Comprehensive Index and Errors Reference

Microsoft® C/C++

Microsoft® C/C++

Version 7.0

Comprehensive Index and Errors Reference

For MS-DOS® and Windows™ Operating Systems

Microsoft Corporation

Document No. LN24779-0192

10 9 8 7 6 5 4 3 2 1

Contents

Introduction .... **v**

Scope of This Book v
Document Conventions v

Part 1 The Comprehensive Index

How to Use the Comprehensive Index **3**

Comprehensive Index **5**

Part 2 Error Messages

Overview **277**

BSCMAKE Error Messages 278
Compiler Error Messages 282
CVPACK Error Messages 441
CodeViewError Messages 446
Command-Line Error Messages 472
HELPMAKE Error Messages 479
IMPLIB Error Messages 485
LINK Error Messages 488
Floating-Point Math Error Messages 513
MPC Error Messages 517
PWB Error Messages 524
Run-Time Error Messages 526
SBRPACK Error Messages 535
NMAKE, EXEHDR, and LIB Error Messages 537

Introduction

Scope of This Book

Comprehensive Index and Errors Reference is divided into two parts. Part 1, "The Comprehensive Index," is a compilation of all of the indexes in the documentation set for Microsoft C/C++ version 7.0. In this index, you can look up any topic covered in the Microsoft C/C++ books.

Part 2, "Error Messages," lists Microsoft C/C++ error and warning messages in alphanumeric order. Each message includes an explanation of what went wrong and what action to take to correct the problem. Error messages can also display the input file and line number where the error occurred.

Document Conventions

Part 2, "Error Messages," uses the following typographic conventions:

Example	Description
STDIO.H	Uppercase letters indicate filenames, segment names, registers, and terms used at the operating-system command level.
char, **_setcolor**, **__far**	Bold type indicates keywords, operators, language-specific characters, and library routines. Within discussions of syntax, bold type indicates that the text must be entered exactly as shown. Many functions and constants begin with either a single or double underscore. These are part of the name and are mandatory. For example, to have the **__cplusplus** manifest constant be recognized by the compiler, you must enter the leading double underscore.
expression	Words in italics indicate placeholders for information you must supply, such as a filename.
[[*option*]]	Items inside double square brackets are optional.

Example	Description
#pragma pack {**1** | **2**}	Braces and a vertical bar indicate a choice among two or more items. You must choose one of these items unless double square brackets ([[ ]]) surround the braces.
`#include <io.h>`	This font is used for examples, user input, program output, and error messages in text.
CL [[*option...*]] *file...*	Three dots (an ellipsis) following an item indicate that more items having the same form may appear.

`while()`
`{`
`  .`
`  .`
`  .`
`}`
	A column or row of three dots tells you that part of an example program has been intentionally omitted.

CTRL+ENTER	Small capital letters are used to indicate the names of keys on the keyboard. When you see a plus sign (+) between two key names, you should hold down the first key while pressing the second.
	The carriage-return key, sometimes marked as a bent arrow on the keyboard, is called ENTER.
“argument”	Quotation marks enclose a new term the first time it is defined in text.
`"C string"`	Some C constructs, such as strings, require quotation marks. Quotation marks required by the language have the form `" "` and `' '` rather than “ ” and ’ ’.
Color Graphics Adapter (CGA)	The first time an acronym is used, it is usually spelled out.

The Comprehensive Index

Part 1

How to use the Comprehensive Index 3

Comprehensive Index 5

How to Use the Comprehensive Index

This Comprehensive Index is a compilation of all of the indexes in the documentation set for Microsoft C/C++. In this index, you can look up any topic covered in the C/C++ books.

The Comprehensive Index is used like any other index, with one exception: after finding your desired topic, you must determine the book in which that topic is located. Page-number references are grouped by book; different books are separated by semicolons. For example, the following entry tells you that you will find information on statements on pages 44 and 89 of the *C Language Reference* and on page 233 of the *C++ Language Reference*:

> Statements, LR 44, 89; LR+ 233

The following key identifies the two- and three-letter book codes which precede the groups of page-number references. This key is repeated on alternating pages of the Comprehensive Index.

Code	Book Title
ET	*Environment and Tools*
LIB	*Run-Time Library Reference*
LR	*C Language Reference*
LR+	*C++ Language Reference*
PT	*Programming Techniques*
TUT	*C++ Tutorial*
XRF	*Class Libraries Reference*
XUG	*Class Libraries User's Guide*

Note *Getting Started* and "Error Messages," Part 2 of this book, are referenced by their full titles.

Comprehensive Index

Key

The following identifies C/C++ manuals:

ET	Environment and Tools
LIB	Run-Time Library Reference
LR	C Language Reference
LR+	C++ Language Reference
PT	Programming Techniques
TUT	C++ Tutorial
XRF	Class Libraries Reference
XUG	Class Libraries User's Guide

A

A command, CodeView, ET 422, 424–426, 436–437
\a (escape sequence), bell, LR 18
\a formatting attribute, HELPMAKE, ET 718, 721
/A option
  CL, PT 39
  NMAKE, ET 648
abort function, LIB 76–77; LR 253
abort function, C++
  described, LR+ 43
  immediate termination, effects, LR+ 45
AbortDoc member function, CDC class, XRF 164–165
Aborting
  abort function, LIB 76–77
  assert function, LIB 92–93
  CDC::AbortDoc, XRF 164
  default termination function, AfxAbort, XRF 61
  print job installing procedure, CDC::SetAbortProc, XRF 235–237
About command
  CodeView, ET 374–375
  PWB, ET 78
  QuickWin, dialog box, customizing, PT 151, 157
About dialog boxes, XUG 105–107, 110
abs function, LIB 78–79
Absolute value, calculating, LIB 78–79
Abstract classes
  design issues, TUT 179
  overview, TUT 128
Abstract declarators
  arrays, LR+ 199–202
  default arguments, LR+ 210–212; LR+ 187
  described, LR 88; LR+ 265–266, 280–281
  function, LR+ 203–210
  in parameter declarations, LR 86
  mixing with parameter declarations, LR 180
  pointers, LR+ 188–190
  pointers to members, LR+ 196–198
  references, LR+ 190–196
Abstraction
  base classes, TUT 169
  classes, TUT 164–165
  data, TUT 163–164
  defined, TUT 161
  overview, TUT 93
  procedural, TUT 162–163
/AC option
  CL, described, ET 488–490, PT 62–63
  HELPMAKE, ET 712
Accelerator key translation, XRF 638
Accelerator keys. *See* Shortcut keys
Accelerator table, Foundation classes tutorial, XUG 109
Accelerator table resource template, XUG 198
Access control, CodeView, ET 409
Access control, C++
  base classes, TUT 131
  design issues, TUT 169
  friends, LR+ 290–295
  members, TUT 45
  member-selection operator overloading, LR+ 363
  multiple-inheritance paths, LR+ 297
  overview, LR+ 285–286
  protected members, LR+ 295–296
  specifiers
    base classes, LR+ 287–290
    described, LR+ 286–287
  virtual functions, LR+ 296–297
_access function, LIB 80–81
accessDenied member function, CFileException class, CFileException::m_cause, XRF 327

Accessing
- files, LR 251
- nested structures, LR 66
- variable-argument lists, va_arg, va_end and va_start functions, LIB 836–839

acos function, LIB 82–83; PT 11

_acosl function, LIB 82–83; PT 11

Activating, windows, PWB, ET 262–263

Active window, QuickWin, PT 152, 161–162

Actual arguments, C++
- defined, LR+ 21

Actual parameters. *See* Arguments

/Ad option, CL, PT 72

Adapters, ET 803

Adapters, graphics (list), PT 167–168

Add command, LIB, ET 703–704

Add member function, CObArray class, XRF 453

Add Watch command, CodeView, ET 364, 423

Add Watch dialog box, CodeView, ET 364

Add Watch Expression command, CodeView, ET 460

AddHead member function, CObList class, XRF 480

Adding
- breakpoints, CodeView, ET 367
- CMainWindow class declaration, XUG 200–202
- commands, PWB Run menu, ET 125–127
- constructors, Phone Book sample program, XUG 202
- custom sections, PWB, ET 58–59
- CWnd to clipboard viewer chain, XRF 807
- dialog boxes, Phone Book sample program, XUG 153–161
- element to array, CObArray::Add, XRF 453
- elements to lists, CObList::AddTail, XRF 481
- filenames
  - to list box of combo box, CComboBox::Dir, XRF 147
  - to list boxes, CListBox::Dir, XRF 360
- files, PWB, ET 44, 47–48, 50
- keyboard and mouse interfaces, Phone Book sample program, XUG 230–233
- lists or elements to lists, CObList::AddHead, XRF 480
- member functions, scrolling, Foundation classes tutorial, XUG 227–229
- memory to heaps, _heapadd functions, LIB 406–409
- menu items, CMenu::InsertMenu, XRF 428–430
- message handlers
  - creation and sizing, XUG 224–226
  - for menu commands, XUG 205–224
- message maps, Foundation classes tutorial, XUG 199–200

Adding (*continued*)
- objects, to a list, XUG 41
- Program Item, PWB, ET 66
- run-time class information, Foundation classes cookbook, XUG 264
- serialization support, Foundation classes cookbook, XUG 265
- size to CSize, CSize::operator+=, XRF 560
- strings
  - to list box of combo box, CComboBox::AddString, XRF 142
  - to list box of combo box, CComboBox::InsertString, XRF 151
  - to list boxes, CListBox::AddString, XRF 355
  - to list boxes, CListBox::InsertString, XRF 366
- time spans
  - CTimeSpan::operator+,-, XRF 625
  - CTimeSpan::operator+=,-=, XRF 626
- utility member functions, Phone Book sample program, XUG 238–241
- watch expressions, CodeView, ET 364, 460
- WM_PAINT message handlers, XUG 235–237, 241

Addition assignment operator (+=), LR 138

Addition operator
- binary-operator expressions, LR+ 104–106
- CRect class, XRF 534
- CString class, XRF 594
- CTime class, XRF 615–616
- CTimeSpan class, XRF 625
- overloading, LR+ 359

Additive operators, LR 128–129

Address ranges
- CodeView expressions, ET 402–403, 420–421
- defined, ET 803

Address space, portability guidelines, PT 283–284

Addresses
- array variables, mixed-language programming, PT 266–267
- CodeView expressions, ET 401–402, 419–420
- common blocks, mixed-language programming, PT 268–269
- defined, ET 803
- normalizing, LR 149
- overloaded function, returned when, LR+ 351
- overriding, LR 56, 169
- pointers
  - described, PT 55
  - portability guidelines, PT 281–283
- register array, warning, LR 266
- variables, debugging assembly code, ET 414

Addressing memory, with based pointers, LR 79

Addressing modes
  based
    described, PT 58
    functions, PT 88–90
    member functions, C++, PT 107–109
    pointers, PT 78–86
  far, PT 57
  huge, PT 57–58
  indirect, portability guidelines, PT 281–283
  keywords, PT 64–66
  near, PT 56–57
  p-code instructions, PT 48
  return objects, PT 101–102
  this pointer, PT 100–101
  v-table pointers, PT 103
Address-of operator (&)
  described, LR 122–124
  in bit-field components, LR 69
  placement and association, LR 122
  with register objects, LR 48
Address-of operator, C++
  overloading, LR+ 355
  unary-operator expressions, LR+ 92–93
AddString member function
  CComboBox class, XRF 142
  CListBox class, XRF 355
AddTail member function, CObList class, XRF 481
adjustfield data member, ios class, XRF 867
/Af option, CL, PT 71
AFLAGS options macro, NMAKE, ET 676
AfxAbort, XRF 61
AfxCheckMemory, XRF 47
AfxDoForAllClasses, XRF 48
AfxDoForAllObjects, XRF 49
afxDump object
  Foundation classes cookbook, XUG 287
  Foundation classes tutorial, XUG 36, 50, 59
AfxEnableMemoryTracking, XRF 49; XUG 291
AfxGetApp, XRF 36
AfxGetAppName, XRF 36
AfxGetInstanceHandle, XRF 36
AfxGetResourceHandle, XRF 37
AfxIsMemoryBlock, XRF 50
AfxIsValidAddress, XRF 51
afxMemDF, XRF 46; XUG 291
afx_msg modifier, Foundation classes tutorial, XUG 90
afx_msg prefix, CMainWindow class, XUG 203–205
AfxRegisterWndClass, XRF 37
AfxRegisterWndClass function
  Foundation classes tutorial, XUG 113
  Foundation classes cookbook, XUG 325
AfxSetAllocHook, XRF 51
AfxSetAllocStop, XRF 52
AfxSetTerminate, XRF 61
AfxTerminate, XRF 62
AfxThrowArchiveException, XRF 63
AfxThrowFileException, XRF 63
AfxThrowMemoryException, XRF 64
AfxThrowNotSupportedException, XRF 64
AfxThrowResourceException, XRF 64
AFXWIN.H file, Foundation classes tutorial, XUG 86, 203
Aggregate types, initializing
  C, LR 93–95, 97
  C++, LR+ 219–222
Aggressive optimization, enabling, PT 20
/AH option, CL, ET 488–490; LR 60; PT 63
/Ah option, CL, PT 71
AH register, CodeView syntax, ET 419, 450
/AL option, CL, ET 488–490; LR 60; PT 63
/Al option, CL, PT 70–71
AL register
  CodeView syntax, ET 419, 450
  fastcall functions, LR 170–171
  for 1-byte return values, LR 171
Aliases, TUT 25–27
Aliasing
  optimization assumptions, PT 13–18
  optimization, CL options, ET 531–532, 536
ALIGN directive, support, PT 113
Aligning
  structures, LR 70–71
  tabs, PWB, ET 297

- Alignment
  - bit fields, LR 69
  - described, LR 123
  - structure members, LR 70, 243
- All Files command, PWB, ET 72
- All Windows command, PWB, ET 77
- _alloca function, LIB 84–85
- allocate member function, streambuf class, XRF 922
- Allocated functions, overlaid DOS programs, ET 599
- Allocating
  - based data
    - dynamic, PT 81–83
    - static, PT 86–88
  - bit fields, LR 69
  - local variables, p-code, PT 52
  - registers
    - CL options, ET 533
    - optimization, PT 21–22
    - portability guidelines, PT 289
  - storage for variables, LR 27
- Allocating memory, LR 253
  - _alloca function, LIB 84–85
  - arrays, calloc functions, LIB 131–132
  - blocks
    - _dos_allocmem function, LIB 183–184
    - _halloc function, LIB 400–401
    - malloc functions, LIB 479–482
  - C++
    - failed, testing for, LR+ 321–323
    - new operator, LR+ 318–320
  - dynamic, LR 35
  - freeing huge memory blocks, _hfree function, LIB 422–423
  - heaps, _bheapseg function, LIB 107–109
  - MOVE, ET 602–603
  - virtual memory blocks, _vmalloc function, LIB 857–858
- alloc_text pragma, PT 77–78, 90
  - described, LR 210
  - module-definition files, ET 626
  - precompiled header compilation, effect on, PT 41
  - replaced by __based keyword, LR 57
  - replacing, LR 173
  - space, LINK, ET 577–578, 599
- alloc_text pragma, C++, LR+ 386
- Alphabetic characters, predefined expression syntax, ET 778, 780, 785
- Alphanumeric characters, predefined expression syntax, ET 778, 780, 785
- ALT key
  - CWnd::OnSysKeyDown, XRF 791–792
  - CWnd::OnSysKeyUp, XRF 792–794
- Alternate floating-point math package
  - described, PT 132
  - option, CL, PT 136
- Alternation, regular expression syntax, ET 778, 780–781, 787
- ALTGR key, enabling, ET 276
- alloc_text pragma
- /AM option, CL, ET 488-490; LR 60, PT 63
- Ambient memory models, C++ classes
  - described, PT 98–99
  - overriding default, PT 99–100
- Ambiguitiy
  - argument matching, LR+ 344–345
  - conversion, TUT 152–155
  - gray expressions, LR+ 130
  - multiple inheritance class names, LR+ 271–274
  - scope rules, LR+ 282
  - this pointer, memory-model specifiers, LR+ 401–402
  - type conversions, LR+ 187
  - user-defined conversions, LR+ 313, 316
- _amblksize variable, LIB 61
- Ampersand (&)
  - C address operator, ET 414–415
  - CodeView, ET 405
  - operations line, extending, ET 702
- /An option, CL, PT 71
- Analysis functions, presentation graphics, PT 202
- Analyzing
  - pie chart data, _pg_analyzepie function, LIB 552
  - scatter chart data, _pg_analyzescatter functions, LIB 553–554
  - series of data, _pg_analyzechart functions, LIB 549–551
- AND operator, bitwise, C++. *See* Bitwise AND operator, C++
- AND operator, logical, C++. *See* Logical AND operator, C++
- AND_CATCH macro, XRF 65; XUG 51, 63
- Angle brackets (< >)
  - command buttons, PWB, ET 80, 82
  - inline files, NMAKE, ET 664–665
- Animate command, CodeView, ET 362–363, 369, 422, 432–433, 453
- Animated graphics functions (list), PT 192–193
- AnimatePalette member function, CPalette class, XRF 503
- ANNUITY1.C sample program, ET 29
- Anonymous allocation, ET 803

Anonymous class types, LR+ 229–230
Anonymous structures, LR 67
Anonymous unions, LR+ 250
ANSI
  additional C features, ET 550–552
  compatibility, LIB x
  converting characters to OEM character set,
      CString::AnsiToOem, XRF 576
  defined, ET 803
  escape sequence, CodeView expressions, ET 408
AnsiToOem member function
  CString class, XRF 576
API defined, ET 803
Apostrophe ('). *See* Escape sequences
Append mode, LR 250
Appending
  characters of strings, strncat and _fstrncat
      functions, LIB 765–766
  DOS device names, CL option, ET 497–498
  menu items, CMenu::AppendMenu, XRF 416–418
  strings, strcat and _fstrcat functions, LIB 738–739
AppendMenu member function
  CMenu class, XRF 416–418
  CWnd::GetSystemMenu, XRF 699
Application class, HELLO sample program, XUG 87
Application object
  creating, Foundation classes tutorial, XUG 85–86
  described, Foundation classes tutorial, XUG 89
  HELLO sample program, XUG 87
Application programming interface, ET 803
Applications
  accessing device facilities unavailable through
      GDI, CDC::Escape, XRF 184–185
  allowing access to Control menu for copying and
      modification, CWnd::GetSystemMenu, XRF 699
  called when creating CWnd object,
      CWnd::OnCreate, XRF 724–726
  CDataBase class, XUG 140
  creating and displaying messages,
      CWnd::MessageBox, XRF 711–714
  CWnd, XRF 735–736
  fonts, called upon changing,
      CWnd::OnFontChange, XRF 738–739
Applications (*continued*)
  inserting text, module-definition files, ET 613–614
  list boxes, returning on application response,
      CWnd::OnCharToItem, XRF 720
  main class described, XRF 5
  owner's, called when destroyed,
      CWnd::OnRenderAllFormats, XRF 780–781
  redrawing or preventing redrawing of changes,
      CWnd::SetRedraw, XRF 811
  returning handle to current instance
    AfxGetInstanceHandle, XRF 36
    AfxGetResourceHandle, XRF 37
  returning name, AfxGetAppName, XRF 36
  sessions, ending, CWnd::OnEndSession, XRF 736
  specifying
    module-definition files, ET 611–612
    protected mode, ET 617
    real mode, ET 617
  specifying the action performed in response to
      message, CWnd::OnVKeyToItem, XRF 795–796
  specifying whether given window is visible,
      CWnd::IsWindowVisible, XRF 710
  Windows
    accessing command-line arguments entered at start,
      CWinApp::m_lpCmdLine, XRF 639
    cleaning up at termination,
      CWinApp::ExitInstance, XRF 631
    constructor, CWinApp::CWinApp, XRF 631
    CWinApp class described, XRF 628
    filtering messages,
      CWinApp::PreTranslateMessage, XRF 638
    handle to current instance,
      CWinApp::m_hInstance, XRF 639
    handle to previous instance,
      CWinApp::m_hPrevInstance, XRF 639
    idle-time processing, CWinApp::OnIdle, XRF 637
    instance initializing, CWinApp::InitInstancei,
      XRF 632
    last message retrieved, CWinApp::m_msgCur,
      XRF 639
    loading cursor resource, CWinApp::LoadCursor,
      XRF 633

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

Applications (*continued*)
  Windows (*continued*)
    loading predefined cursor resource
      CWinApp::LoadOEMCursor, XRF 634
      CWinApp::LoadStandardCursor, XRF 635
    loading predefined icon resource
      CWinApp::LoadOEMIcon, XRF 634
      CWinApp::LoadStandardIcon, XRF 636
    loading specified icon resource, CWinApp::LoadIcon, XRF 633
    making main window visible, CWinApp::m_nCmdShow, XRF 640
    name, CWinApp::m_pszAppName, XRF 640
    one-time initializing, CWinApp::InitApplication, XRF 632
    providing default message loop, CWinApp::Run, XRF 638
    storing pointer to main window object, CWinApp::m_pMainWnd, XRF 640
APPLOADER statement, module-definition files, ET 609, 615
_arc function, LIB 86–87; PT 189
Arc member function, CDC class, XRF 165–166
Arccosines, calculating, acos functions, LIB 82–83
Architecture, segmented, effects on addresses in arrays, LR 130
Archives
  CArchive class, XRF 93–94
  CArchiveException class, XRF 104
  data
    determining if loading, CArchive::IsLoading, XRF 98
    determining if storing, CArchive::IsStoring, XRF 98
  flushing buffer to file, CArchive::Flush, XRF 97
  getting CFile pointer, CArchive::GetFile, XRF 97
  loading object or primitive type, CArchive::operator, XRF 102
  reading
    from object data, CArchive::ReadObject, XRF 99
    from specified number of bytes, CArchive::Read, XRF 99
  serialization exceptions, constructing objects, CArchiveException::CArchiveException, XRF 105
  specifying cause, CArchiveException::m_cause, XRF 106
  storing
    object or primitive type, CArchive::operator, XRF 103
    objects, CArchive::WriteObject, XRF 101
  writing, specified number of bytes to, CArchive::Write, XRF 100
Arcs
  determining viewpoint coordinate endpoints, _getarcinfo function, LIB 344
  drawing, _arc functions, LIB 86–87
  elliptical, drawing, CDC::Arc, XRF 165–166
Arcsines, calculating, asin functions, LIB 90–91
Arctangents, calculating, atan functions, LIB 94–95
_arc_w function, LIB 86–87
_arc_wxy function, LIB 86–87; PT189
Arg function, PWB, ET 150, 156–157
  executing, ET 106–108
  getting help, ET 762
  replacing text, ET 94
argc, ET 803
argc argument, C++
  main function syntax, LR+ 38
argc parameters, passing information to main, LR 31
Argument lists, variable length, LIB 59
Argument passing conventions, CString class, XRF 599
Argument types, C++
  differentiation in overloaded functions, LR+ 340
Arguments
  See also Parameters
  ANSI compatibility, LR 233
  character, LR 171
  CodeView
    entering, ET 352
    format, ET 352–353
    setting, ET 363–364
  command line, ET 337
  command-line rules, LR 30–33
  defined, ET 803; LR 30, 179
  evaluating
    ASSERT macro, XRF 53
    VERIFY macro, XRF 57
  floating-point, calculating absolute value, fabs and _fabsl functions, LIB 258–259
  function calls, LR 119, 185–187
  functions, PWB, ET 156–157
  in pragmas, LR 209
  inserting into streams, XRF 906
  lists, variable, portability limitations, PT 289
  module statement syntax, ET 610
  numeric in LINK, ET 576
  passed by value, LR 185
  prototypes, LR 167
  returning variable number as printf uses them, TRACE macro, XRF 56
  side effects, LR 194
  side effects, portability guidelines, PT 290–291

- Arguments (*continued*)
  - to main, LR 234
  - type checking, LIB x, 8
  - variable, accessing lists, va_arg, va_end and va_start functions, LIB 836–839
- Arguments, C++
  - actual
    - defined, LR+ 21
  - command-line, parsing, LR+ 40–42
  - default
    - declarators, LR+ 210–212
    - overloaded functions, TUT 22
    - overview, TUT 11–12
    - scope, LR+ 212, 283
  - defined, LR+ 21
  - formal
    - defined, LR+ 21
    - scope, LR+ 33
  - function
    - names, LR+ 283
  - handling, _setargv function, LR+ 39–41
  - matching, overloaded functions, LR+ 344–350
- argv, ET 803
- argv argument, C++
  - main function syntax, LR+ 38
- argv parameters, passing information to main, LR 31
- Arithmetic conversions, LR 115–116; LR+ 69–71
- Arithmetic modes
  - portability guidelines, PT 280–281
- Arithmetic, pointer
  - huge memory model effect, PT 60–61
  - mixed memory model effect, PT 64
  - speed, PT 57–58
- Arrange command
  - CodeView, ET 373–374
  - PWB, ET 77; ET 145
- Arrange Icons command, QuickWin, PT 150
- ArrangeIconicWindows member function, CWnd class, XRF 659
- Arrangewindow function, PWB, ET 150, 157
- Arranging, minimized document child windows, CMDIFrameWnd::MDIIconArrange, XRF 406
- Array classes, XRF 27
- Array declarations, LR 74–76
- Array type, incomplete, LR 100
- Arrays
  - 16-bit words, CWordArray class, XRF 823
  - adding element to, CObArray::Add, XRF 453
  - allocating with new, TUT 66, 95
  - as arguments, LR 185
  - bytes, dumping hexadecimally formatted, CDumpContext::HexDump, XRF 276
  - characters, initializing, LR 97–98
  - classes, XRF 27
  - CObject pointers, CObArray class, XRF 450
  - constructors, LR+ 305
  - conversion to pointers, LR+ 73–74
  - CString objects, CStringArray class, XRF 601
  - deallocating with delete, TUT 66, 96, 99
  - debugging assembly language, ET 415
  - declarations, LR 74–76
  - declarators, LR+ 199–202
  - declaring, LR 42; PT 263–264
  - defined, LR 53
  - destroying, CObArray::~CObArray, XRF 454
  - double words, CDWordArray class, XRF 280
  - dynamic, of bytes, CByteArray class, XRF 135–136
  - element types, LR 21
  - elements
    - adding at end, CObArray::Add, XRF 453
    - deleting, Foundation classes cookbook, XUG 274
    - inserting in specified index, CObArray::InsertAt, XRF 457
    - removing elements, CObArray::RemoveAt, XRF 458–459
    - returning at specified index, XRF 455
    - returning reference to pointer, CObArray::ElementAt, XRF 454–455
    - setting at specified index, CObArray::SetAt, XRF 459–461
  - establishing size, CObArray::SetSize, XRF 461
  - expanding and contracting, CodeView, ET 367–368, 478–479
  - freeing extra memory, CObArray::FreeExtra, XRF 455
  - huge arithmetic operations, LR 95

Arrays (*continued*)
  in expressions, LR 107
  indexes, setting elements to specified, CObArray::SetAt, XRF 459–460
  indexing
    See also Pointer arithmetic
    mixed-language programming, PT 263–264
  initializing, LR 93–97; LR+ 222–223, 328–329; TUT 95
  internal character, returning pointer to, ostrstream::str, XRF 914
  iteration of, Foundation classes cookbook, XUG 272
  multidimensional
    declaring, LR 75
    described, LR 117–119
  of bit fields, not allowed, LR 69
  of objects
    declaring, TUT 94
    initialization, TUT 95
  of pointers, LR 75
  one-dimensional, LR 116–117
  pointer comparison, LR 133
  removing pointers from, CObArray::RemoveAll, XRF 458
  returning size of, CObArray::GetSize, XRF 456
  returning upper bound, CObArray::GetUpperBound, XRF 456
  returns reference to element pointer, CObArray::ElementAt, XRF 454–455
  searching, bsearch function, LIB 127–128
  size, LR 57, 94, 241
  sorting, qsort function, LIB 605–606
  storage
    by row, LR 75
    of arrays, LR 76
  strstreambuf objects, preventing memory deletion, strstreambuf::freeze, XRF 944
  subscript operators, CObArray::operator<>, XRF 462
  types, LR+ 52
  unsized
    as last member of structure, LR 68, 125
    declaring in member lists, LR+ 239
  using huge, with library functions, LIB 16
  void pointers, CPtrArray class, XRF 517
  v-table pointers, PT 102–103
  zero-sized, as last member of structure, LR 68
Arrow keys, Phone Book sample program, message handlers, XUG 231
AS command macro, NMAKE, ET 675
/AS option, CL, ET 488–490; LR 60; PT 63
/As option, CL, PT 70–71
ASCII, ET 804; LR 191; PT 284–285;
  HELPMAKE format, ET 728
  memory format, CodeView, ET 356–357
asctime function, LIB 88–89
asin functions, LIB 90–91; PT 11
_asinl function, LIB 90–91; PT 11
Askexit switch, PWB, ET 263, 267
Askrtn switch, PWB, ET 263, 267
__asm blocks
  described, PT 112
  __fastcall calling convention limitations, PT 120–121
  features, PT 113–115
  function calls
    C, PT 122
    C++, PT 123
  labels, PT 121–122
  language elements, using, PT 115–119
  macros, defining as, PT 123–124
  optimization, effects on, PT 124–125
  registers, PT 120–121
.ASM files, ET 804
__asm keyword, LR 174; PT 112
Aspect-ratio filter, retrieving setting, CDC::GetAspectRatioFilter, XRF 192
Assemble command, CodeView, ET 422, 424–426, 436–437
Assembling, 8086 instruction mnemonics, ET 424–426
Assembly files, creating listing, CL, ET 501–505
Assembly groups, PT 112
Assembly, inline. *See* Inline assembly
Assembly language
  debugging, ET 412–415
  inline assembly. *See* Inline assembly
  mixed-language programming, PT 248–257
  specifying, CL option, ET 542
  translating source code, CL, ET 501–505
Assembly mode, ET 804
assert function, LIB 92–93; LR 247
ASSERT macro, XRF 53
  Foundation classes cookbook, XUG 288–289
  Foundation classes tutorial, XUG 127, 138
  Phone Book sample program, XUG 203
  validity testing, Foundation classes tutorial, XUG 33
assert macro, C++, LR+ 43
Assertions, Phone Book sample program, XUG 203
ASSERT_VALID macro, XRF 53
  Foundation classes tutorial, XUG 126, 138
  validity testing, Foundation classes tutorial, XUG 32

AssertValid member function
  CObject class, XRF 466
    debugging, used for, XUG 138
    described, XUG 32
    overriding, XUG 289
    Phone Book sample program, XUG 133–138
Assign function, PWB, ET 150, 157–158
  executing, ET 108
  key assignment, changing, ET 121
  switch settings, changing, ET 124
Assigning attributes and behavior, TUT 174
Assignment
  class objects, TUT 77–78
  copying objects, LR+ 333–337
  initialization, TUT 77–78
Assignment conversions, LR 141–147
Assignment of addition operator, CRect class, XRF 532
  CString class, XRF 595
  CTime class, XRF 616
  CTimeSpan class, XRF 626
Assignment of intersection operator, CRect class, XRF 533
Assignment of union operator, CRect class, XRF 533–534
Assignment operator
  binary-operator expressions, LR+ 112–116
  copy constructors, TUT 79
  CRect class, XRF 531
  CSize class, XRF 560–561
  CString class, XRF 592
  CTime class, XRF 615
  CTimeSpan class, XRF 625
  default behavior for objects, TUT 69
  described, LR 138
  istream class, XRF 889
  ostream class, XRF 910
  overloading, LR+ 359–360; TUT 71–76
Assignment operator, overloaded
  CPerson class, XUG 29–30
  Foundation classes tutorial, XUG 221
  Phone Book sample program, XUG 129
Assistance, customer. *See* the Product Assistance Request Form in LIB
Associating, attributes, LR 27
Associativity
  C operators, LR 23
  C++ operators, LR+ 11–14, 91
  defined, LR 112
  operator overloading, TUT 138
Assumptions
  aliasing, optimization options, PT 13–18
  compiler, effect on portability, PT 285
  hardware, effect on portability, PT 271
/A*string* options, CL, PT 70
Asterisk (*)
  Comment command, CodeView, ET 423, 471
  Copy command, LIB, ET 705
  declaring pointer return type, LR 87
  deleting watch expressions, CodeView, ET 461
  filename macros, NMAKE, ET 672–673
  hyperlink, Microsoft Advisor, ET 764
  in identifiers, LR 55
  modifying identifiers, LR 42
  regular expressions, PWB, ET 93
  SBRPACK syntax, ET 740
  specifying filenames and pathnames, LR 31
  wildcard operator
    HELPMAKE syntax, ET 711
    NMAKE, ET 653–654
    UNDEL syntax, ET 749
/AT option, CL, ET 488–490; PT 63
At sign (@)
  appended to name by __stdcall, LR 58
  BSCMAKE syntax, ET 738
  command files, NMAKE, ET 651
  command modifier, NMAKE, ET 661
  filename macros, NMAKE, ET 672–673
  LINK syntax, ET 573
  local contexts, HELPMAKE, ET 720
  naming registers, CodeView, ET 401, 419
  Redraw command, CodeView, ET 424, 479
atan function, LIB 94–95; PT 11
atan2 function, LIB 94–95; PT 11
_atan2l function, LIB 94–95; PT 11

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

_atanl function, LIB 94–95; PT 11
atexit function, LIB 96–97; LR 253
atexit function, C++, LR+ 45
atof function, LIB 98–100
atoi function, LIB 98–100
atol function, LIB 98–100
_atold function, LIB 98–100
Attach member function
  CGdiObject class, XRF 344
  CMenu class, XRF 419
  CWnd class, XRF 659
  filebuf class, XRF 832
  fstream class, XRF 838
  ifstream class, XRF 846
  ofstream class, XRF 894
Attaching
  filebuf objects to specified open file, filebuf::attach, XRF 832
  labels to charts, PT 202
  stream to specified open file, fstream::attach, XRF 838
  streams
    to already open file, ostream::attach, XRF 894
    to specified open file, ifstream::attach, XRF 846
  Windows GDI object to CGdiObject, CGdiObject::Attach, XRF 344
Attributes
  colors, selecting, PT 175–176
  formatting in HELPMAKE, ET 718, 721
  of a class, TUT 174
  of functions, LR 181, 183
  overriding defaults, LR 168
  relationships between classes, TUT 176
  segment, defining with module statements, ET 618–619
/Au option, CL, LR 175; PT 72
Auto option, Language command, CodeView, ET 370
auto keyword, C++
  declaration statements, LR+ 149–151
  declarations, use in, LR+ 157
  described, LR+ 46
auto storage class
  default, LR 47
  local variables, LR 181
auto storage-class specifier
  external level, invalid, LR 45
  lifetime, LR 37
  local lifetime, LR 44
  nonterminal, LR 43
  omitted, LR 44
auto storage-class specifier (*continued*)
  precompiled header compilation, effect on, PT 41
  visibility, LR 37–38
AUTOEXEC.BAT
  See also Getting Started
  HELPFILES environment variable, ET 771
  PWB configuration, ET 137
auto_inline pragma
  described, LR 210
  new in version 7.0, LR 258
auto_inline pragma, C++, LR+ 386
Autoload switch, PWB, ET 263, 268
Automatic indentation, PWB, C switches, ET 312
Automatic storage class, C++
  declaration statements, LR+ 149–151
  specifiers, LR+ 157–158
Automatic variables, C++
  described, LR+ 46
  initialization, LR+ 47–49
Autosave switch, PWB, ET 122, 263, 269
Autostart entry, TOOLS.INI file, CodeView, ET 330
Autostart macro, PWB, ET 224
AUX, CL options, appending to, ET 497–498
Available memory
  defined, ET 804
  overlaid DOS programs, ET 597
/Aw option, CL, PT 72–74
AX register
  changed, LR 58
  CodeView syntax, ET 419, 450
  fastcall functions, LR 170
  for 2-byte return values, LR 171
Axes, presentation graphics
  described, PT 204
  structure types, PT 221–223
_axistype structures, presentation graphics, PT 221–223

B

\b (escape sequence), backspace, LR 18
\b formatting attribute, HELPMAKE, ET 719, 721, 727
/B option
  CodeView, ET 338–339
  LINK, ET 576
  NMAKE, ET 648
/B1 option, CL, LR 267
/B2 option, CL, LR 267
/B3 option, CL, LR 267

- Background
  - CWnd, called when needing erasing, CWnd::OnEraseBkgnd, XRF 737–738
  - mode, getting, CDC::GetBkMode, XRF 193
- Background colors
  - getting, _getbkcolor function, LIB 345
  - setting current, _setbkcolor function, LIB 652–653
- Backing up files, PWB, ET 95, 303, 747–750
- Backslash (\)
  - arguments, usage in, LR 32
  - continuation character, LR 19
  - escape, regular expression syntax, ET 780, 786
  - escape sequence, LR 18
  - HELPMAKE syntax, ET 720–721
  - line concatenation operator, LR 194
  - line continuation character
    - NMAKE, ET 655, 660, 669
    - PWB, ET 115–117, 136
  - line splicing, LR 191
  - LINK syntax, ET 567
  - match character, regular expression syntax, ET 779
  - regular expressions, PWB, ET 96
  - Screen Exchange command, CodeView, ET 424, 479
- Backspace (\b), (escape sequence), LR 18
- Backtab function, PWB, ET 127–128, 150, 159
- Backup files
  - creating, ET 747–750
  - setting number, in PWB, ET 303
- Backup switch, PWB, ET 263, 269–270
- bad member function
  - ios class, XRF 855
  - ofstream class, iostream classes tutorial, XUG 377
- .BAK files, ET 804
- Bar charts
  - described, PT 203
  - sample program, PT 208, 211–212
  - styles, PT 204–205
- BAR.C sample presentation graphics program, PT 208, 211–212
- .BAS files, ET 804
- Base classes
  - abstraction, TUT 169
  - access from derived classes, TUT 112, 130–131
  - access specifiers, LR+ 287–290
  - conversion to derived class, TUT 118
  - described, LR+ 266
  - direct, TUT 113, 132
  - indirect, TUT 113, 132
  - initializating, LR+ 329–333; TUT 116
  - multiple
    - described, LR+ 267–271
    - name ambiguities, LR+ 271–274
  - pointers to, conversion
    - from pointers to classes, LR+ 72–73
    - to pointers to derived classes, LR+ 76
  - private, TUT 131
  - public, TUT 111, 131
  - references to, conversion from references to classes, LR+ 75
  - tutorial information, TUT 111
  - virtual
    - design issues, TUT 180
    - described, LR+ 268–271
    - overview, TUT 133–134
  - virtual functions, LR+ 275–279
- Base initializer, TUT 116
- base member function, streambuf class, XRF 922
- Base names
  - Curfilenam predefined macro, PWB, ET 225
  - defined, ET 804
  - Shortnames switch, PWB, ET 296
- Base operator (:>), LR 121
- Base operator (:<), CodeView precedence, ET 406
- Base operator, C++, LR+ 406
- Based addressing
  - C++, LR+ 405–406
  - described, PT 58
  - functions, PT 88–90
  - member function, C++, PT 107–109

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

- Based addressing (*continued*)
  - pointers
    - described, PT 78–79
    - fixed base, PT 79–80
    - __self keyword, PT 85
    - variable base, PT 80–84
    - __void keyword, PT 86
  - supported by base operator, LR 121
  - uses, LR 79
- Based data
  - dynamically allocated, PT 81–83
  - statically allocated, PT 86–88
- Based functions, C++, LR+ 411–412
- Based heap, PT 82
- __based keyword, PT 64–66, 78–79
  - /A options, CL, ET 489
  - allocated functions, ET 599
  - compiler errors, LR 258
  - described, LR 55–57, 59, 79
  - enabling, CL options, ET 550
  - offset, LR 121
  - ordering functions, module-definition files, ET 626
  - overriding addressing modes, LR 169
  - replaces same_seg pragma, LR 266
  - segmenting data, CL option, ET 530
  - specifying function in specific segment, LR 169
- __based keyword, C++
  - described, LR+ 405–412
- Based objects, C++, LR+ 411–414
- Based pointers
  - C++, LR+ 74, 406–411
  - defined, LR 57
  - described, LR 79–84; PT 58, 78–79
  - fixed base, PT 79–80
  - normalized, LR 242
  - null, LR 149
  - __self keyword, PT 85
  - variable base, PT 80–84
  - __void keyword, PT 86
- Based variables, declaring, PT 65–66
- basefield data member, ios class, XRF 867
- BASIC, mixed-language programming, PT 240–243
- Basic Compiler
  - NMAKE command macro, ET 675
  - NMAKE options macro, ET 676
- .BAT file, ET 804
- Batch files
  - backing up C files, PWB, ET 95
  - building browser database, PWB, ET 104–105
  - defined, ET 804
  - executing CL, ET 490
- /BATCH option, LINK, ET 576
- /batch option, CL, ET 490
- BC command, CodeView, ET 422, 426–427
- BC command macro, NMAKE, ET 675
- _bcalloc function, LIB 131–132
- BD command, CodeView, ET 422, 427–428
- _bdos function, LIB 101–102
- BE command, CodeView, ET 422, 428–429
- Beep switch, PWB, ET 263, 270
- Begfile function, PWB, ET 150, 159
- BEGIN_MESSAGE_MAP macro
  - Foundation classes cookbook, XUG 313–315
  - Foundation classes tutorial, XUG 95
- BeginPaint member function
  - CPaintDC::m_ps, XRF 500
  - CWnd class, XRF 660
- Begline function, PWB, ET 150, 159–160
- Behavior
  - of a class, TUT 175
  - relationships between classes, TUT 176
- Bell character (\a), LR 18
- Bessel functions, LIB 103–105
- _bexpand function, LIB 255–257
- BFLAGS options macro, NMAKE, ET 676
- _bfree function, LIB 306–308
- _bfreeseg function, LIB 106
- BH register, CodeView syntax, ET 419, 450
- _bheapadd function, LIB 406–409
- _bheapchk function, LIB 410–412
- _bheapmin function, LIB 413–414
- _bheapseg function, LIB 107–109; PT 82
- _bheapset function, LIB 415–417
- _bheapwalk function, LIB 418–421
- Binary file operations, CFile class, Foundation classes cookbook, XUG 277
- Binary files, ET 804
- Binary numbers
  - floating-point, storing as normalized numbers, PT 128–129
  - processor arithmetic modes, PT 280–281
- Binary operators, C++
  - additive, LR+ 104–106
  - assignment, LR+ 112–116
  - bitwise AND, LR+ 110–111
  - bitwise exclusive OR, LR+ 110–111
  - bitwise inclusive OR, LR+ 110–111
  - bitwise shift, LR+ 106–107
  - comma, LR+ 116–117
  - equality, LR+ 107–110
  - (list), LR+ 102–103
  - logical AND, LR+ 111–112

- Binary operators, C++ (*continued*)
  - logical OR, LR+ 112
  - multiplicative, LR+ 103–104
  - overloading, LR+ 358–363
  - relational, LR+ 107–110
- Binary operators defined, ET 804; LR 112
- Binary output files, output streams, iostream classes tutorial, XUG 378–379
- Binary/text mode, setting
  - filebuf objects, filebuf::setmode, XRF 835
  - stream's filebuf object, ifstream::setmode, XRF 851
  - streams, ios& binary, XRF 868
  - streams, ofstream::setmode, XRF 899
- Binding
  - *See also* Precedence
  - defined, LR 112
  - done when, XRF 25
  - expressions, LR 114
  - postfix operators, LR 116
- BIOS
  - calling time and date services, _bios_timeofday function, LIB 125–126
  - communications services, _bios_serialcom function, LIB 122–124
  - defined, ET 804
  - disk services, _bios_disk function, LIB 110–113
  - equipment-list service, _bios_equiplist function, LIB 114–115
  - interface services routines, LIB 55
  - keyboard services, _bios_keybrd function, LIB 116–118
  - memory-size service, _bios_memsize function, LIB 119
  - preprocessing directives, NMAKE, ET 691
  - printer services, _bios_printer function, LIB 120–121
- _bios_disk function, LIB 110–113
- _bios_equiplist function, LIB 114–115
- _bios_keybrd function, LIB 116–118
- _bios_memsize function, LIB 119
- _bios_printer function, LIB 120–121
- _bios_serialcom function, LIB 122–124
- _bios_timeofday function, LIB 125–126
- Bit fields
  - described, LR 68; LR+ 252–254
  - portability guidelines, PT 279–280
  - signed, LR 243
  - storage, LR 243
- Bit patterns, creating for specified device, CDC::PatBlt, XRF 218–219
- Bit rate, remote debugging, ET 396
- bitalloc member function, ios class, XRF 855
- BitBlt member function
  - CDC class, XRF 167–169
    - CBitmap::CreateBitmap, XRF 110
    - CBitmap::CreateBitmapIndirect, XRF 110
- Bitmaps
  - associating with menu items, CMenu::SetMenuItemBitmaps, XRF 435–436
  - constructing CBitmap objects, CBitmap::CBitmap, XRF 109
  - copying
    - bit pattern to buffer, CBitmap::GetBitmapBits, XRF 113
    - to current device context, CDC::BitBlt, XRF 167–169
  - creating
    - discardable, CBitmap::CreateDiscardableBitmap, XRF 112
    - device-dependent memory bitmap, CBitmap::CreateBitmap, XRF 109–110
    - having lpBitmap structure, CBitmap::CreateBitmapIndirect, XRF 110
  - GDI, CBitmap class, XRF 107–108
  - getting characters, _pg_getchardef function, LIB 564
  - initializing, compatible with device specified by pDC, CBitmap::CreateCompatibleBitmap, XRF 111
  - moving, CDC::StretchBlt, XRF 255–257
  - predefined, loading, CBitmap::LoadOEMBitmaps, XRF 115–116
  - resource, loading, CBitmap::LoadBitmap, XRF 114
  - returning pointer to CBitmap object, CBitmap::FromHandle, XRF 113
  - setting bits to specified values, CBitmap::SetBitmapBits, XRF 116
  - setting characters, _pg_setchardef function, LIB 573

Bitmaps (*continued*)
  stretching mode, retrieving, CDC::GetStretchBltMode, XRF 204
  width, height, getting, CBitmap::GetBitmapDimension, XRF 114
  width, specifying, CBitmap::SetBitmapDimension, XRF 116–117
Bits, rotating
  _lrotl and _lrotr functions, LIB 468
  _rotl and _rotr functions, LIB 633–634
Bitwise AND assignment operator (&=), LR 138
Bitwise AND operator (&), LR 134
Bitwise AND operator, C++
  binary-operator expressions, LR+ 110–111
  overloading, LR+ 358
Bitwise complement operator, C++
  unary-operator expressions, LR+ 94
Bitwise exclusive OR assignment operator(^=), LR 138
Bitwise exclusive OR operator (^), LR 134
Bitwise exclusive OR operator, C++
  binary-operator expressions, LR+ 110–111
  overloading, LR+ 359
Bitwise inclusive OR assignment operator(|=), LR 138
Bitwise inclusive OR operator (|), LR 134
Bitwise inclusive OR operator, C++
  binary-operator expressions, LR+ 110–111
  overloading, LR+ 359
Bitwise NOT operator (~), LR 124
Bitwise operators, LR 130–132, 134–135
Bitwise shift (>>) operator
  portability guidelines, LR 130–133; PT 287
Bitwise shift operators, C++
  binary-operator expressions, LR+ 106–107
  overloading, LR+ 359
BL command, CodeView, ET 422, 429
BL register
  CodeView, ET 419, 450
  fastcall functions, LR 171
Black, color value, ET 273
_BLACK constant, PT 180
BLACKRECT structure, CStatic::Create, XRF 564
Blank lines, command lines, NMAKE, ET 660
blen member function, streambuf class, XRF 922
Block scope
  function declarations, LR 266
  linkage rules, LR+ 34
  rules, LR 36
Blocks
  See also Compound statements
  __asm. *See* __asm blocks
  defined, LR 28
  mixed-language programming, PT 267–268
  virtual memory
    dirty vs. clean, specifying, PT 92
    loading, PT 92
    locking and unlocking, PT 92–93
Blue, color value, ET 273
_BLUE constant, PT 180
/Bm option, CL, ET 491; LR 260
_bmalloc function, LIB 479–482; PT 83
.BMP files, ET 804
BN_CLICKED message
  Foundation classes cookbook, XUG 316
  Foundation classes tutorial, XUG 98
  naming conventions, Foundation classes tutorial, XUG 205
Bold text, HELPMAKE formatting
  QuickHelp, ET 721
  rich text format, ET 726
B_OnExit QuickBASIC function, PT 242
Books of interest, LIB xiii
Boolean switches, PWB, ET 122, 266
Borders, drawing
  around rectangles, CDC::FrameRect, XRF 191
  around regions, CDC::FrameRgn, XRF 192
BottomRight member function, CRect class, XRF 523
Bounding rectangles
  See also Rectangles
  coordinate systems, PT 185
Box Mode command, PWB, ET 73, 143
Boxes, buttons
  getting
    check state, CButton::GetCheck, XRF 132
    current state, CButton::GetState, XRF 132
  setting check state, CButton::SetCheck, XRF 134
Boxes, command execution, PWB, ET 80
Boxes, setting highlighting, CButton::SetState, XRF 134
BP command, CodeView, ET 384–385, 422, 429–432
  display radix, ET 444–445
  line numbers, ET 400
BP register, CodeView syntax, ET 419, 450

Braces ({ })
  __asm blocks, PT 112
  block delimiters, LR 28
  compound statement delimiters, LR 151
  context operator, CodeView, ET 405–407, 421–422
  enclosing __asm blocks, LR 174
  in initializer lists, LR 95–96
  key box, PWB, ET 120
  RTF formatting codes, ET 726
  specifying search path, NMAKE, ET 660
Brackets ([ ])
  array declarations, LR 88
  balancing, PWB, ET 192–193
  call tree, PWB, ET 100
  character class, regular expression syntax, ET 778, 780–781, 786
  double, use of, LIB xv; XUG viii
  in declarations, LR 42, 74
  in identifiers, LR 55
  inline assembly, using in, PT 116
  match character
    regular expression syntax, ET 779
break statements
  See also continue statements
  described, LR 152
  example, LR 152
  preferred over goto statements, LR 158
  terminate for statements, LR 156
  transfer control, LR 161
  within switch statements, LR 162
break statements, C++
  jump statements, LR+ 147
  selection statements, LR+ 141–142
Breakpoint Clear command, CodeView, ET 422, 426–427
Breakpoint Disable command, CodeView, ET 422, 427–428
Breakpoint Enable command, CodeView, ET 422, 428–429
Breakpoint List command, CodeView, ET 422, 429
Breakpoint Set command, CodeView, ET 384–385, 422, 429–432, 444–445
Breakpoints
  CodeView
    clearing, ET 426–427
    disabling, ET 427–428
    enabling, ET 428–429
    listing, ET 429
    saving, ET 344
    setting, ET 326, 365–366, 384–385, 429–432
  defined, ET 804
_brealloc function, LIB 613–615
Bright Back check box, PWB, customizing colors, ET 125
Bright Fore check box, PWB, customizing colors, ET 125
_BRIGHTWHITE constant, PT 180
BringWindowToTop member function, CWnd class, XRF 661
Browcase switch, PWB, ET 309
Browdbase switch, ET 309–310
Brown, color value, ET 273
_BROWN constant, PT 180
Browse menu
  PWB, ET 76
  functions, ET 145, 200
BROWSE option, BSCMAKE, ET 732
Browse Options command, PWB, ET 75
Browser Database Maintenance Utility. *See* BSCMAKE
Browser database, PWB
  building, ET 58, 101
    BSCMAKE, ET 731–734
    combined, ET 106
    non-PWB projects, ET 104–106
  BSCMAKE error messages. *See* “Error Messages,” Part 2 of this book
  creating, ET 97–98
  estimating file size, ET 103–104
  finding symbol definitions, ET 101–103
  makefiles, ET 61
  menu commands, ET 76
  overview, ET 731

Browser database, PWB (*continued*)
  SBRPACK error messages. *See* "Error Messages," Part 2 of this book
  specifying switches, ET 310
Browser Information Compactor. *See* SBRPACK
Browser information files, PWB
  browser database, ET 97–98
  building browser database, non-PWB, ET 105
  estimating size, ET 103–104
Browser Output command, PWB, ET 77
Browser Output pseudofile, PWB, ET 102
Browser, source. *See* Source browser
Brushes
  available in device context, enumerating, CDC::EnumObjects, XRF 182–184
  CBrush class, XRF 118
  CBrush object, returning pointer to, CBrush::FromHandle, XRF 125
  constructing uninitialized, CBrush::CBrush, XRF 119–120
  current, retrieving origin, CDC::GetBrushOrg, XRF 193
  filling
    display area with current, CDC::ExtFloodFill, XRF 187–188, CDC::FloodFill, XRF 190
    rectangles with specified, CDC::FillRect, XRF 189–190
    regions with, CDC::FillRgn, XRF 190
    specified region, CDC::PaintRgn, XRF 218
  initializing
    DIB-specified pattern, CBrush::CreateDIBPatternBrush, XRF 121–122
    hatch pattern and color, CBrush::CreateHatchBrush, XRF 122–124
    LOGBRUSH-specified pattern, CBrush::CreateBrushIndirect, XRF 120–121
    pattern specified by bitmap, CBrush::CreatePatternBrush, XRF 123
    solid color, CBrush::CreateSolidBrush, XRF 124
  predefined
    retrieving handle to, CGdiObject::CreateStockObject, XRF 345–346
    selecting, CMetaFileDC::SelectStockObject, XRF 442
  resetting CGdiObject::UnrealizeObject, XRF 350
  setting origin for GDI assignment, CDC::SetBrushOrg, XRF 239
.BSC files
  defined, ET 804
  PWB, ET 97, 103–104
BSCMAKE
  building a database, ET 731–734
  command line, ET 735–738
  error messages. *See* "Error Messages," Part 2 of this book
  options, ET 732, 736–737
  overview, ET 731
  response files, ET 738
  syntax, ET 735
  system requirements, ET 734
BSCMAKE command, building browser database
  non-PWB, ET 105
  PWB, ET 97–98
BSCMAKE.EXE, ET 734
BSCMAKEV.EXE, ET 734
bsearch function, LIB 127–128
Buffer-deletion flags, assigning value for stream, ios::delbuf, XRF 856
Buffering
  output streams
    effects, XUG 377–378
  setting state for streambuf object, stream::unbuffered, XRF 937
Buffer-manipulation routines, LIB 18
Buffers
  archive, flushing
    CArchive::Close, XRF 97
    CArchive::Flush, XRF 97
  CodeView command window, ET 353
  committing contents to disk, LIB 37
  controlling, setting size, setvbuf function, LIB 688–689
  decompression, specifying size, ET 332
  defined, ET 804
  filling with data that defines object, CGdiObject::GetObject, XRF 348
  flushing
    files, CFile::Flush, XRF 309
    to dump context, CDumpContext::CDumpContext, XRF 275
  internal character, returning pointer for CString object, CString::GetBuffer, XRF 582–583
  moving one to another, memmove and _fmemmove functions, LIB 510–512
  QuickWin, getting screen buffer size, _wgetscreenbuf function, LIB 875–876
  setting to specified character, memset and _fmemset functions, LIB 513–514
  size, QuickWin, PT 161
  stream control, setbuf function, LIB 654–655
  streams, flushing, ostream::flush, XRF 902

- Buffers (*continued*)
  - writing data to CFile object file, CFile::Write, XRF 320
  - writing to files, _dos_write function, LIB 232–233
- Bugs. *See* Debugging
- Build command, PWB, ET 74, 144
- Build directories, Foundation classes tutorial, XUG 9
- Build Results command, PWB, ET 77
- Build Results window, PWB, ET 261
  - clearing, ET 162
  - Nextmsg function, ET 188–189
  - retaining results, ET 289
- Build switch, PWB, ET 263
- Building
  - browser database, PWB, ET 97–98, 101
    - BSCMAKE, ET 731–734
    - combined, ET 106
    - non-PWB projects, ET 104–106
  - canceling, _pwbcancelbuild macro, ET 229–230
  - customized PWB projects, ET 58–61
  - debug mode, XUG 12
  - described, PWB, ET 56–58
  - menu commands, PWB, ET 74
  - multimodule programs, PWB, ET 45–46
  - programs, NMAKE, ET 646
  - _pwbbuild macro, ET 229
  - targets
    - NMAKE, ET 648
    - PWB, ET 163–164
- Building DLLs, module-definition files, LR 172
- Build:message switch, tagged expressions, ET 784
- Button control
  - calling owner when visual aspect of control or menu changes, CWnd::OnDrawItem, XRF 732–735
  - check-marking, dimming, CWnd::CheckDlgButton, XRF 661
  - determining check-marking, CWnd::IsDigButtonChecked, XRF 709
- Buttons
  - as child windows, XRF 16
  - check-marking, XRF 662
  - command execution, PWB, ET 80–82
- Buttons (*continued*)
  - creating
    - constructor, CButton::CButton, XRF 128
    - control, CButton::Create, XRF 128–131
  - hyperlinks
    - index screens, ET 764
    - navigating with, ET 759–761
  - styles
    - changing, CButton::SetButtonStyle, XRF 133
    - getting, CButton::GetButtonStyle, XRF 131
- Buttons, boxes
  - called when control created, CWnd::OnMeasureItem, XRF 754–756
  - calling owner when visual aspect of control or menu changes, CWnd::OnDrawItem, XRF 732–735
  - check-marked, getting ID of radio button, CWnd::GetCheckedRadioButton, XRF 686
- BX register
  - changed, LR 58
  - CodeView syntax, ET 419, 450
  - fastcall functions, LR 170
- BY operator, CodeView, ET 405, 414–415
- Bypassing buffers, flushing file to disk, LR 260
- Byte order
  - portability guidelines, PT 277–278, 292–293
- Bytes
  - displaying, CodeView, ET 356–357
  - dumping array of hexadecimally formatted, CDumpContext::HexDump, XRF 276
  - dynamic array support, CByteArray class, XRF 135–136
  - extracting from streams, istream::read, XRF 883
  - file length, CFile::GetLength, XRF 309
  - inputting from port, _inp and _inpw functions, LIB 428
  - locking range in open file, CFile::LockRange, XRF 312
  - locking, unlocking, _locking function, LIB 460–462
  - outputting at port, _outp and _outpw functions, LIB 542–544
  - swapping, _swab function, LIB 801–802
  - writing, to streams, ostream::write, XRF 905

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

C

- C calling conventions, PT 28–29
- :c command
  - HELPMAKE, ET 722
  - PWB, ET 785
- C Compiler
  - NMAKE command macro, ET 675
  - NMAKE options macro, ET 676
- C Compiler Options command, PWB, ET 75
- C expression evaluator
  - choosing, ET 403–404
  - defined, ET 399
  - overview, ET 399
  - using, ET 404–407
- .C files, ET 804
- C functions, C++ linkage, LR+ 36–37
- C macros, defining as __asm blocks, PT 123–124
- /C option
  - CL, ET 325–491
  - CodeView, ET 338, 340
  - HELPMAKE, ET 712
  - NMAKE, ET 648
- /c option, CL
  - described, ET 491
  - option interactions, ET 496–497
- c option, optimize pragma, PT 23
- C run-time functions, CString functions, comparison to, Foundation classes cookbook, XUG 258
- C++
  - accommodates Windows messaging, XRF 9
  - and Windows, XRF 8
  - constructors, XRF 15
  - file translation order, LR+ 1–2
  - fundamental elements, LR+ 1
  - global objects, Foundation classes tutorial, XUG 89
  - Microsoft C version 7.0 support for, LR 257
  - overview, LR+ 17
  - techniques
    - classes, deriving, XUG 24
    - constructors, XUG 30
    - Foundation classes tutorial, XUG 38
    - Windows programs, creating, XUG 81
  - v-tables
    - Foundation classes tutorial, XUG 203
    - message maps, similarity to, XUG 96
  - Windows objects, constructing, XRF 15
- C++ class libraries
  - advantages, XUG ii
  - class source code, modification of, XUG iv
  - classes and objects, direct use of, XUG iii
  - derivation of new classes, XUG iii
  - documentation, how to use, XUG vii
  - introduction, XUG 1
- C++ Compiler, NMAKE command macro, ET 676
- C++ Compiler Options command, PWB, ET 75
- C++ expression evaluator
  - choosing, ET 403–404
  - overview, ET 399
  - using, ET 404–415
- C++ header code
  - generating for CObject class
    - serializable, DECLARE_SERIAL macro, XRF 468
    - with run-time information, DECLARE_DYNAMIC macro, XRF 468
  - generating for dynamic CObject-derived class with run-time access to class name and position IMPLEMENT_DYNAMIC, XRF 471
- _cabs function, LIB 129–130
- _cabsl function, LIB 129–130
- Caches, overlaid DOS programs, ET 598, 602–603
- CALCRECT structure, CDC::DrawText, XRF 177–179
- Calculating
  - absolute value
    - arguments, abs function, LIB 78–79
    - complex numbers, _cabs and _cabsl functions, LIB 129–130
    - floating point arguments, fabs and _fabsl functions, LIB 258–259
    - long integers, labs function, LIB 445–446
  - arccosines, acos functions, LIB 82–83
  - arcsines, asin functions, LIB 90–91
  - arctangents, atan functions, LIB 94–95
  - ceilings of values, ceil and _ceill functions, LIB 133–134
  - cosines, cos functions, LIB 163–164
  - exponentials, exp and _expl functions, LIB 253–254
  - floating-point remainders, fmod and _fmodl functions, LIB 288–289
  - floors of values, floor and _floorl functions, LIB 285–286
  - height of CRect, CRect::Height, XRF 525
  - hypotenuses, _hypot and _hypotl functions, LIB 424–425

- Calculating *(continued)*
  - logarithms, log functions, LIB 463–464
  - nonclient area, CWnd::OnNcCalcSize, XRF 762
  - square roots, sqrt and _sqrtl functions, LIB 727–728
  - tangents, tan functions, LIB 805–806
  - time used by calling process, clock function, LIB 154–155
  - width of CRect, CRect::Width, XRF 530
- Call gates, ET 804
- Call Tree command, PWB, ET 76, 145
- Call trees
  - PWB, ET 86
  - showing, ET 99–101
- Callback function, XRF 183, 211, 236
- Calling
  - BIOS
    - communications services, _bios_serialcom function, LIB 122–124
    - disk services, _bios_disk function, LIB 110–113
    - equipment-list service, _bios_equiplist function, LIB 114–115
    - keyboard services, _bios_keybrd function, LIB 116–118
    - memory-size service, _bios_memsize function, LIB 119
    - printer services, _bios_printer function, LIB 120–121
    - time and date services, _bios_timeofday function, LIB 125–126
  - library routines, LIB 5–6
  - processes, terminating, exit and _exit functions, LIB 251–252
- Calling conventions
  - C, PT 29
  - CL options, ET 516–520
  - far, PT 64, 66–68, 88–89
  - __fastcall, PT 30–32
  - FORTRAN/Pascal, PT 29
  - mixed-language programming, PT 234–235, 258
  - overview, PT 29
  - register, PT 30
  - specifying, LR 169–170
  - unsupported, LR 266
- Calling conventions, C++
  - __cdecl keyword, LR+ 416–418
  - __fastcall keyword, LR+ 417–418
  - __fortran keyword, LR+ 418
  - linkage specification effects, LR+ 178–179
  - modifiers, LR+ 415–416
  - __pascal keyword, LR+ 418
  - __stdcall keyword, LR+ 419
- Calling destructors, LR+ 310–311
- Calling functions
  - *See also* Function calls
  - CodeView expressions, ET 405
- calloc functions, LIB 131–132; TUT 101
- Calls menu, CodeView, ET 372–373
- Calls to emulator option, floating-point math, PT 135
- Calls to math coprocessor option, floating-point math, PT 134–136
- Cancel button, overriding in dialog boxes, CModalDialog::DoModal, XRF 448
- Cancel function, PWB, ET 150, 160
- Canceling
  - background search, _pwbcancelsearch macro, ET 230–231
  - builds, _pwbcancelbuild macro, ET 229–230
  - print operations, _pwbcancelprint macro, ET 230
- Cancelsearch function, PWB, ET 150, 161
- CanUndo member function, CEdit class, XRF 285
- Captions
  - dialog boxes, retrieving, CWnd::GetDlgItemText, XRF 691
  - setting, to specified text, CWnd::SetWindowText, XRF 816
- CArchive class
  - described, XRF 93–94; XUG 20
  - member functions
    - CArchive, XRF 95–96
    - ~CArchive, XRF 96
    - Close, XRF 96–97
    - Flush, XRF 96–97, 309
    - GetFile, XRF 97
    - IsLoading, XRF 98, 473
    - IsStoring, XRF 98, 473

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

CArchive class (*continued*)
member functions (*continued*)
Read, XRF 99
ReadObject, XRF 99–100, 473
Write, XRF 100
WriteObject, XRF 101, 473
object persistence described, XRF 23
operators, XRF 102–103
serialization, Foundation classes cookbook, XUG 279
CArchive constructor, XRF 95–96
CArchive destructor, XRF 96
CArchive object, closing and disconnecting from file, CArchive::Close, XRF 97
creating
CArchive::CArchive, XRF 95
Foundation classes cookbook, XUG 284
described, Foundation classes tutorial, XUG 61
destroying, CArchive::CArchive, XRF 96
extraction operator, Foundation classes tutorial, XUG 33
insertion operator, Foundation classes tutorial, XUG 33
I/O, XUG 33, 61
serialization, Foundation classes tutorial, XUG 33
CArchiveException class
data members, m_cause, XRF 106
described, XRF 104
member functions, CArchiveException, XRF 105
CArchiveException constructor, XRF 105
CArchiveException exception handler, Foundation classes cookbook, XUG 298
Carets (^)
See also Bitwise-exclusive OR operator
character ranges, regular expression syntax, ET 779
coordinates, retrieving, CWnd::GetCaretPos, XRF 686
displaying
after gaining input focus, CWnd::OnSetFocus, XRF 783
at current position, CWnd::ShowCaret, XRF 816–817
gray, creating, CWnd::CreateGrayCaret, XRF 671
hiding, CWnd::HideCaret, XRF 705
line beginning, regular expression syntax, ET 778–781, 786
literal characters, NMAKE, ET 653
moving to position specified by point, CWnd::SetCaretPos, XRF 806–807
regular expressions, PWB, ET 92, 95
Carets (^) (*continued*)
solid, creating, CWnd::CreateSolidCaret, XRF 672
system, creating new shape, CWnd::CreateCaret, XRF 665
user-defined macros, NMAKE, ET 669
Carriage-return escape sequence (\r), LR 18
Cascade command
PWB
described, ET 77
predefined macros, ET 145
QuickWin, PT 150, 152–153
Cascading, window arrangements, _pwbcascade macro, ET 231
Case, matching, CodeView search option, ET 361
case labels, in switch statements, LR 161–164
Case sensitivity
browser database, ET 309
CodeView
commands, ET 368, 371, 417
expression evaluators, ET 405
options, ET 445–447
defined, ET 804
global searches, in Microsoft Advisor, ET 766
identifiers, LR 6, 235
IMPLIB, ET 747
labels, inline assembler, PT 121–122
LIB options, ET 701
operating systems, LIB 9
PWB
options, ET 141–142
search functions, ET 270–271
case statements, C++, labels, LR+ 135–136, 139–142
Case switch, PWB, ET 263, 270–271
Case translation, portability guidelines, PT 285
Cast
See also Type casts
long values to unsigned short type, LR 100
operators, LR 126
operators, C++. *See* Conversion functions, C++
segment values, LR 83
types, LR 64
Casting operator, CString::operator const char* (), XRF 592
CATCH macro, XRF 65
exception object argument, XUG 63
exception type argument, XUG 63
Foundation classes cookbook, XUG 298–302
Foundation classes tutorial, XUG 51, 63
Categories, presentation graphics, PT 202
.category command, HELPMAKE, ET 722

CBitmap class
  described, XRF 107–108
  member functions
    CBitmap, XRF 109
    CreateBitmap, XRF 109–110, 120, 123, 665
    CreateBitmapIndirect, XRF 110, 120, 123
    CreateCompatibleBitmap, XRF 111, 120, 123
    CreateDiscardableBitmap, XRF 112
    FromHandle, XRF 113
    GetBitmapBits, XRF 113, 349
    GetBitmapDimension, XRF 114
    LoadBitmap, XRF 114, 120, 123, 665
    LoadOEMBitmap, XRF 115–116
    SetBitmapBits, XRF 116
    SetBitmapDimension, XRF 116–117
CBitmap constructor, XRF 109
CBitmap object, copying bit pattern to buffer, CBitmap::GetBitmapBits, XRF 113
  creating, CBitmap::CBitmap, XRF 109
  returning pointer, CBitmap::FromHandle, XRF 113
CBitmap, width, height, getting, CBitmap::GetBitmapDimension, XRF 114
CBrush class
  described, XRF 118
  member functions
    CBrush, XRF 119–120
    CreateBrushIndirect, XRF 120–121
    CreateDIBPatternBrush, XRF 121–122
    CreateHatchBrush, XRF 122–123
    CreatePatternBrush, XRF 123–124
    CreateSolidBrush, XRF 124
    FromHandle, XRF 125
    UnrealizeObject, XRF 738
CBrush constructor, XRF 119–120
CBrush object, creating uninitialized, CBrush::CBrush, XRF 119–120
CButton class
  described, XRF 126–127
  Foundation classes cookbook, XUG 335
CButton class (*continued*)
  member functions
    CButton, XRF 128
    Create, XRF 128–131
    GetButtonStyle, XRF 131
    GetCheck, XRF 132
    GetState, XRF 132–133
    SetButtonStyle, XRF 133
    SetCheck, XRF 134
    SetState, XRF 134
  setting highlight state, CButton::SetState, XRF 134
CButton constructor, XRF 128
CButton object
  control, CButton::Create, XRF 128–131
  creating, CButton::CButton, XRF 128
CByteArray class, XRF 135–136
CC command macro, NMAKE, ET 675
CClientDC class
  data members, m_hWnd, XRF 138
  described, XRF 137
  member functions
    CClientDC, XRF 138
    ~CClientDC, XRF 138
CClientDC constructor, XRF 138
CClientDC destructor, XRF 138
CClientDC objects
  creating, CClientDC::CClientDC, XRF 138
  destroying, CClientDC::~CClientDC, XRF 138
  handles, CClientDC::m_hWnd, XRF 138
CComboBox class
  described, XRF 139–141
  edit control
    copies current selection to clipboard, CComboBox::Copy, XRF 143
    deleting selection, CComboBox::Clear, XRF 143
    inserting clipboard data into, CComboBox::Paste, XRF 152
    selecting characters in, CComboBox::SetEditSel, XRF 154
    text, limiting length, CComboBox::LimitText, XRF 152

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

- CComboBox class (*continued*)
  - Foundation classes cookbook, XUG 335
  - items
    - getting number of, CComboBox::GetCount, XRF 149
    - retrieving associated application-supplied 32-bit value, CComboBox::GetItemData, XRF 150
    - setting associated with 32-bit value, CComboBox::SetItemData, XRF 155
  - list box
    - adding string to, CComboBox::AddString, XRF 142
    - removing all items from, CComboBox::ResetContent, XRF 153
    - searching for string in, CComboBox:Select String, XRF 153
    - selecting string in, CComboBox::SetCurSel, XRF 154
    - showing or hiding specified, CComboBox::ShowDropDown, XRF 155
  - member functions
    - AddString, XRF 142
    - CComboBox, XRF 142
    - Clear, XRF 143
    - Copy, XRF 143
    - Create, XRF 143–146
    - Cut, XRF 146
    - DeleteString, XRF 147, 729
    - Dir, XRF 147–148
    - FindString, XRF 148
    - GetCount, XRF 149
    - GetCurSel, XRF 149
    - GetEditSel, XRF 149
    - GetItemData, XRF 150
    - GetLBText, XRF 150
    - GetLBTextLen, XRF 151
    - InsertString, XRF 151
    - LimitText, XRF 152
    - Paste, XRF 152
    - ResetContent, XRF 153, 729
    - SelectString, XRF 153
    - SetCurSel, XRF 154
    - SetEditSel, XRF 154–155
    - SetItemData, XRF 155
    - ShowDropDown, XRF 155
- CComboBox constructor, XRF 142
- CComboBox object
  - creating, CComboBox::Create, XRF 143–144, 146
  - ending print job page, CDC::EndPage, XRF 181
  - printing, terminating job, CDC::EndDoc, XRF 180
- CDataBase class
  - applications, Foundation classes tutorial, XUG 140
  - member functions
    - database, XUG 135
    - file handling, XUG 136
  - overview, Foundation classes tutorial, XUG 134–140
  - Phone Book sample program, XUG 123
- CDataBase constructor, Foundation classes tutorial, XUG 135
- CDataBase object, member variables, Foundation classes tutorial, XUG 136
- CDC class
  - allowing applications to access device facilities, CDC::Escape, XRF 184–185
  - bit pattern, creating, CDC::PatBlt, XRF 218–219
  - bitmaps, moving, CDC::StretchBlt, XRF 255, 257
  - bitmap-stretching mode, retrieving, CDC::GetStretchBltMode, XRF 204
  - CGdiObject object, selecting, CDC::SelectStockObject, XRF 234–235
  - character strings, computing width and height, CDC::GetTabbedTextExtent, XRF 204–205
    - writing, with tab stops, CDC::TabbedTextOut, XRF 258–259
  - clipping region
    - creating, CDC::ExcludeClipRect, XRF 185
    - selecting given region as current, CDC::SelectClipRgn, XRF 231
  - colors
    - retrieving current text, CDC::GetTextColor, XRF 207
    - retrieving RGB value of specified pixel, CDC::GetPixel, XRF 202
    - returning closest to device capability, CDC::GetNearestColor, XRF 202
    - setting background, CDC::SetBkColor, XRF 238
    - setting text, CDC::SetTextColor, XRF 248
    - updating client area with current, CDC::UpdateColors, XRF 260
  - converting logical to device points, CDC::LPtoDP, XRF 215
  - copying bitmap, CDC::BitBlt, XRF 167–169
  - creating
    - bit pattern on device, CDC::PatBlt, XRF 218–219
    - clipping region, CDC::IntersectClipRect, XRF 212
  - current position, retrieving, CDC::GetCurrentPosition, XRF 195
  - described, XRF 156–163

CDC class (*continued*)
device contexts
creating for specified device, CDC::CreateDC, XRF 172–173
deleting, CDC::DeleteDC, XRF 174
obtaining final translation origin, CDC::GetDCOrg, XRF 196
saving current state, CDC::SaveDC, XRF 227
display device, getting information on, CDC::GetDeviceCaps, XRF 196, 201
drawing
dimmed text, CDC::GrayString, XRF 210–212
ellipses, CDC::Ellipse, XRF 179–180
elliptical arcs, CDC::Arc, XRF 165–166
filled polygons, CDC::PolyPolygon, XRF 222
icons, CDC::DrawIcon, XRF 176
line segments, CDC::Polyline, XRF 222
lines to points, CDC::LineTo, XRF 214
pie-shaped wedge, CDC::Pie, XRF 219–220
polygons consisting of points, CDC::Polygon, XRF 221
rectangles in focus style, CDC::DrawFocusRect, XRF 175
rectangles with current pen, CDC::Rectangle, XRF 224
rectangles with rounded corners, CDC::RoundRect, XRF 226–227
setting current mode, CDC::SetROP2, XRF 243–244
drawing mode, retrieving, CDC::GetROP2, XRF 203
dynamic runtime checking supported, XRF 156
filling
display area with current brush, CDC::ExtFloodFill, XRF 187–188
display area with current brush, CDC::FloodFill, XRF 190
rectangle using specified brush, CDC::FillRect, XRF 189–190
specified region with brush, CDC::PaintRgn, XRF 218

CDC class (*continued*)
fonts
altering mapper, CDC::SetMapperFlags, XRF 241
copying typeface name into buffer, CDC::GetTextFace, XRF 208
retrieving character widths, CDC::GetCharWidth, XRF 194
retrieving metrics for current, CDC::GetTextMetrics, XRF 208
information contexts, creating, CDC::CreateIC, XRF 173
intercharacter spacing setting, CDC::GetTextCharacterExtra, XRF 206
mapping mode, retrieving, CDC::GetMapMode, XRF 201
mapping point coordinates, CDC::DPtoLP, XRF 175
member functions
AbortDoc, XRF 164–165
Arc, XRF 165–166
BitBlt, XRF 110, 167–169
CDC, XRF 169
~CDC, XRF 170
Chord, XRF 170–171
CreateCompatibleDC, XRF 171–172
CreateDC, XRF 172–173
CreateIC, XRF 173–174
DeleteDC, XRF 174–175
DPtoLP, XRF 175
DrawFocusRect, XRF 175–176
DrawIcon, XRF 176
DrawText, XRF 177–179
Ellipse, XRF 179–180
EndDoc, XRF 180–181
EndPage, XRF 181–182
EnumObjects, XRF 182–184
Escape, XRF 184–185
ExcludeClipRect, XRF 185–186
ExcludeUpdateRgn, XRF 186
ExtTextOut, XRF 188–189
FillRect, XRF 189–190

CDC class *(continued)*
member functions *(continued)*
FillRgn, XRF 190
FloodFill, XRF 190–191
FrameRect, XRF 191
FrameRgn, XRF 192
GetAspectRatioFilter, XRF 192
GetBkColor, XRF 193
GetBkMode, XRF 193
GetBrushOrg, XRF 193–194
GetCharWidth, XRF 194
GetClipBox, XRF 195
GetCurrentPosition, XRF 195
GetDCOrg, XRF 196
GetDeviceCaps, XRF 196–201
GetMapMode, XRF 201
GetNearestColor, XRF 202
GetPixel, XRF 202–203
GetPolyFillMode, XRF 203
GetROP2, XRF 203
GetStretchBltMode, XRF 204
GetTabbedTextExtent, XRF 204–205
GetTextAlign, XRF 205–206
GetTextCharacterExtra, XRF 206
GetTextColor, XRF 207
GetTextExtent, XRF 207
GetTextFace, XRF 208
GetTextMetrics, XRF 208
GetViewportExt, XRF 209
GetViewportOrg, XRF 209
GetWindowExt, XRF 209
GetWindowOrg, XRF 210
GrayString, XRF 210–212
IntersectClipRect, XRF 212–213
InvertRect, XRF 213
InvertRgn, XRF 214
LineTo, XRF 214
LPtoDP, XRF 215
MoveTo, XRF 215–216
OffsetClipRgn, XRF 216
OffsetViewportOrg, XRF 217
OffsetWindowOrg, XRF 217
PaintRgn, XRF 218
PatBlt, XRF 218–219
Pie, XRF 219–220
PlayMetaFile, XRF 221, 440
Polygon, XRF 221
Polyline, XRF 222
PolyPolygon, XRF 222–223
PtVisible, XRF 223
RealizePalette, XRF 224, 350, 507

CDC class *(continued)*
member functions *(continued)*
Rectangle, XRF 224–225
RectVisible, XRF 225
RestoreDC, XRF 226
RoundRect, XRF 226–227
SaveDC, XRF 227–228
ScaleViewportExt, XRF 228
ScaleWindowExt, XRF 229
ScrollDC, XRF 230
SelectClipRgn, XRF 231
SelectGdiObject, XRF 110, 112, 334
SelectObject, XRF 110, 112, 232–233, 334
SelectPalette, XRF 233–234
SelectStockObject, XRF 234–235
SetAbortProc, XRF 235–237
SetBkColor, XRF 238
SetBkMode, XRF 238–239
SetBrushOrg, XRF 239, 350
SetMapMode, XRF 240–241
SetMapperFlags, XRF 241
SetPixel, XRF 242
SetPolyFillMode, XRF 242–243
SetROP2, XRF 243–244
SetStretchBltMode, XRF 245
SetTextAlign, XRF 246–247
SetTextCharacterExtra, XRF 247
SetTextColor, XRF 248
SetTextJustification, XRF 248–249
SetViewportExt, XRF 250
SetViewportOrg, XRF 251
SetWindowExt, XRF 252
SetWindowOrg, XRF 253
StartDoc, XRF 254
StartPage, XRF 254
StretchBlt, XRF 255–257
TabbedTextOut, XRF 258–259
TextOut, XRF 259–260
UpdateColors, XRF 260
metafile, playing on device, CDC::PlayMetaFile, XRF 221
modifying
viewport extents, CDC::ScaleViewportExt, XRF 228
window origin, CDC::OffsetWindowOrg, XRF 217
windows extents, CDC::ScaleWindowExt, XRF 229
moving clipping region, CDC::OffsetClipRgn, XRF 216

CDC class (*continued*)
palettes, logical
mapping entries to system palette, CDC::RealizePalette, XRF 224
selecting, CDC::SelectPalette, XRF 233
pens, brushes available, enumerating, CDC::EnumObjects, XRF 182–184
pixels, setting at specified point, CDC::SetPixel, XRF 242
polygon-filling mode
retrieving, CDC::GetPolyFillMode, XRF 203
setting, CDC::SetPolyFillMode, XRF 242
position, current, moving to point, CDC::MoveTo, XRF 215
print job, informing device driver of new, CDC::StartDoc, XRF 254
printer driver, preparing to receive data, CDC::StartPage, XRF 254
printing
installing abort procedure for job, CDC::SetAbortProc, XRF 235–237
terminating current job, CDC::AbortDoc, XRF 164
rectangles
bounding, retrieving dimensions around clipping boundary, CDC::GetClipBox, XRF 195
determining if within clipping region, CDC::RectVisible, XRF 225
drawing borders around, CDC::FrameRect, XRF 191
drawing text in, CDC::DrawText, XRF 177–179
drawing with rounded corners, CDC::RoundRect, XRF 226–227
inverting contexts, CDC::InvertRect, XRF 213
scrolling, CDC::ScrollDC, XRF 230
regions
drawing border around, CDC::FrameRgn, XRF 192
filling with specified brush, CDC::FillRgn, XRF 190
filling with specified brush, CDC::PaintRgn, XRF 218
inverting contents, CDC::InvertRgn, XRF 214

CDC class (*continued*)
regions (*continued*)
preventing drawing within invalid area, CDC::ExcludeUpdateRgn, XRF 186
writing character strings within, CDC::ExtTextOut, XRF 188–189
restoring Windows device context to previous state, CDC::RestoreDC, XRF 226
retrieving
aspect-radio filter setting, CDC::GetAspectRatioFilter, XRF 192
current brush origin, CDC::GetBrushOrg, XRF 193
window origin coordinates, CDC::GetWindowOrg, XRF 210
returning
background mode, CDC::GetBkMode, XRF 193
current background color, CDC::GetBkColor, XRF 193
selecting object into current device context, CDC::SelectObject, XRF 232–233
setting
background mode, CDC::SetBkMode, XRF 238
bitmap-stretching mode, CDC::SetStretchBltMode, XRF 245
current drawing mode, CDC::SetROP2, XRF 243–244
intercharacter spacing, CDC::SetTextCharacterExtra, XRF 247
mapping mode, CDC::SetMapMode, XRF 240–241
specifying next brush origin, CDC::SetBrushOrg, XRF 239
terminating print job
CDC::AbortDoc, XRF 164
CDC::EndDoc, XRF 180
text
computing line dimensions, CDC::GetTextExtent, XRF 207
setting alignment flags, CDC::SetTextAlign, XRF 246–247
setting color, CDC::SetTextColor, XRF 248

- CDC class (*continued*)
  - text (*continued*)
    - setting justification, CDC::SetTextJustification, XRF 248–249
  - text-alignment flag status, retrieving, CDC::GetTextAlign, XRF 205
  - viewports
    - modifying origin, CDC::OffsetViewportOrg, XRF 217
    - retrieving device contexts' extents, CDC::GetViewportExt, XRF 209
    - retrieving origin coordinates associated with device context, CDC::GetViewOrg, XRF 209
    - setting origins of device context, CDC::SetViewportOrg, XRF 251
    - setting x- and y-extents of device context, CDC::SetViewportExt, XRF 250
  - windows, XRF 253
    - retrieving coordinates associated with device context, CDC::GetWindowExt, XRF 209
    - setting x- and y-extents, CDC::SetWindowExt, XRF 252
  - writing character strings at specified location, CDC::TextOut, XRF 259
- CDC constructor, XRF 169
- CDC destructor, XRF 170
- CDC objects, creating,
  - CDC::CreateCompatibleDC, XRF 171
  - constructor, CDC::CDC, XRF 169
  - destroying, CDC::CDC, XRF 170
- __cdecl keyword
  - calling convention, LR 55, 57–58
  - calling conventions, CL options, ET 516–518
  - enabling, CL options, ET 550
  - example, LR 170
  - invalid with fastcall, LR 171
  - specifying, LR 169–170
  - symbol format, CodeView, ET 409
- __cdecl keyword, C++
  - calling convention, LR+ 416–417
  - described, LR+ 7
- Cdelete function, PWB, ET 150, 161
- CDialog class
  - deriving dialog classes, Foundation classes cookbook, XUG 332
  - deriving simple classes, Foundation classes cookbook, XUG 333
  - described, XRF 261–263
- CDialog class (*continued*)
  - dialog boxes, modeless, XUG 162, 165
  - focus control
    - CDialog::GotoDlgCtrl, XRF 268
    - CDialog::NextDlgCtrl, XRF 270
    - previous, CDialog::PrevDlgCtrl, XRF 272
  - font control, XRF 262, 271
  - member functions
    - CDialog, XRF 264
    - Create, XRF 264–265
    - CreateIndirect, XRF 266–267
    - EndDialog, XRF 267
    - GetDefID, XRF 268
    - GotoDlgCtrl, XRF 268
    - IsDialogMessage, XRF 268–269
    - MapDialogRect, XRF 270
    - NextDlgCtrl, XRF 270
    - OnInitDialog, XRF 271
    - OnSetFont, XRF 271–272
    - PrevDlgCtrl, XRF 272
    - SetDefID, XRF 272
  - message-checking, CDialog::IsDialogMessage, XRF 268–269
  - message-handler member functions, XRF 261
  - modeless dialog box, creating, CDialog::CDialog, XRF 264
  - push button control, default, CDialog::SetDfID, XRF 272
  - screen unit conversion control, CDialog::MapDialogRect, XRF 270
  - standard dialog box procedure, CDialog::OnInitDialog, XRF 271
  - standard Foundation dialog box procedure, XRF 267
- CDialog constructor, XRF 264
- CDumpContext class
  - described, XRF 273–274
  - member functions
    - CDumpContext, XRF 275
    - Flush, XRF 275
    - GetDepth, XRF 276
    - HexDump, XRF 276
    - SetDepth, XRF 277
  - operators, XRF 278–279
- CDumpContext constructor, XRF 275
- CDumpContext object
  - creating,CDumpContext::CDumpContext, XRF 275
  - DMTEST sample program, XUG 35
- CDWordArray class, XRF 280–281

- CEdit class
  - described, XRF 282–284
  - Foundation classes cookbook, XUG 335
  - member functions
    - CanUndo, XRF 285
    - CEdit, XRF 285
    - Clear, XRF 285
    - Copy, XRF 286
    - Create, XRF 286–289
    - Cut, XRF 289
    - EmptyUndoBuffer, XRF 290
    - FmtLines, XRF 290
    - GetHandle, XRF 291
    - GetLine, XRF 291–292
    - GetLineCount, XRF 292
    - GetModify, XRF 292–293
    - GetRect, XRF 293
    - GetSel, XRF 293
    - LimitText, XRF 294
    - LineFromChar, XRF 294
    - LineIndex, XRF 295
    - LineLength, XRF 295–296
    - LineScroll, XRF 296
    - Paste, XRF 297
    - ReplaceSel, XRF 297
    - SetHandle, XRF 297–298
    - SetModify, XRF 298
    - SetPasswordChar, XRF 299
    - SetRect, XRF 299–300
    - SetRectNP, XRF 300
    - SetSel, XRF 301
    - SetTabStops, XRF 301–302
    - Undo, XRF 302
- CEdit constructor, XRF 285
- CEdit object, creating
  - attaching, CEdit::Create, XRF 286–289
  - constructor, CEdit::CEdit, XRF 285
- CEditDialog class, Foundation classes tutorial, XUG 163–165
- ceil function, LIB 133–134; PT 11
- _ceill function, LIB 133–134; PT 11
- cerr, TUT 7
  - predefined stream object, XRF 900
- CException class, XRF 303
- _cexit function, LIB 135
- CFile class, XUG 20, 277
  - data members
    - m_hFile, XRF 321
  - described, XRF 304–305
  - family, XRF 26
  - member functions
    - CFile, XRF 306–308
    - ~CFile, XRF 308
    - Close, XRF 97, 308
    - Duplicate, XRF 309
    - Flush, XRF 309
    - GetLength, XRF 309
    - GetPosition, XRF 96, 310
    - GetStatus, XRF 310–312
    - LockRange, XRF 312–313
    - Open, XRF 313–314, 568
    - Read, XRF 314, 570
    - Remove, XRF 315
    - Rename, XRF 315–316
    - Seek, XRF 316–317
    - SeekToBegin, XRF 317
    - SeekToEnd, XRF 317
    - SetLength, XRF 318
    - SetStatus, XRF 318–319
    - UnlockRange, XRF 319
    - Write, XRF 320, 570
- CFile constructor, XRF 306–308
    - CArchive::CArchive, XRF 96
    - CStdioFile::CStdioFile, XRF 568
- CFile destructor, XRF 308
- CFile objects
  - closing associated file, CFile::Close, XRF 308
  - creating
    - constructor, CFile::CFile, XRF 306
    - safe method, CFile::Open, XRF 313
  - described, Foundation classes tutorial, XUG 60
  - destroying, CFile::~CFile, XRF 308
  - duplicating, CFile::Duplicate, XRF 309
  - reading data into buffer, CFile::Read, XRF 314
  - retrieving file status, CFile::GetStatus, XRF 310–311

- CFileException class
  - data members
    - m_cause, XRF 327–328
    - m_lOsError, XRF 328
    - OsErrorToException, XRF 325
    - ThrowErrno, XRF 325–326
    - ThrowOsError, XRF 326
  - described, XRF 322–323
  - enumerators, XRF 327
  - member functions
    - CFileException, XRF 324
    - ErrnoToException, XRF 324–325
  - operating system error codes, XRF 324–326, 328
- CFileException exception handler, Foundation classes cookbook, XUG 298
- CFileException object
  - creating, throwing exception, CFileException::CFileException, XRF 324
  - Foundation classes tutorial, XUG 60
- CFindDialog class
  - dialog boxes, XUG 163–164
  - Foundation classes tutorial, XUG 153, 163–164
- CFLAGS options macro, NMAKE, ET 676
- CFont class
  - described, XRF 329
  - member functions
    - CFont, XRF 330
    - CreateFont, XRF 330–333
    - CreateFontIndirect, XRF 334
    - FromHandle, XRF 335
- CFont data member, CFont class, XRF 330
- CFont objects
  - creating, CFont::CFont, XRF 330
  - handle to, CFont::FromHandle, XRF 335
  - initializing
    - LOGFONT-specified characteristics, CFont::CreatFontIndirect, XRF 334
    - specified characteristics, CFont::CreateFont, XRF 330–333
- CFrameWnd class
  - accelerator table, loading, CFrameWnd::LoadAccelTable, XRF 340
  - data members
    - m_hAccelTable, XRF 341
    - rectDefault, XRF 341
  - described, XRF 336–337
  - Foundation classes cookbook, XUG 311, 323–325
  - HELLO sample program, XUG 83
- CFrameWnd class (*continued*)
  - member functions
    - CFrameWnd, XRF 338
    - ~CFrameWnd, XRF 338
    - Create, XRF 338–339
    - GetChildFrame, XRF 339
    - GetParentFrame, XRF 340
    - LoadAccelTable, XRF 340
  - overridable member function
    - CFrameWnd::GetParentFrame, XRF 340
- CFrameWnd constructor, XRF 338
- CFrameWnd destructor, XRF 338
- CFrameWnd object
  - creating
    - attaching, CFrameWnd::Create, XRF 338–339
    - constructor, CFrameWnd::CFrameWnd, XRF 338
  - destroying, CFrameWind::~CFrameWnd, XRF 338
- CGA, ET 805
- CGA displays, CodeView option, ET 341
- CGdiObject class
  - data members
    - m_hObject, XRF 351
  - described, XRF 342–343
  - member functions
    - Attach, XRF 344
    - CGdiObject, XRF 344
    - ~CGdiObject, XRF 345
    - CreateStockObject, XRF 345–346
    - DeleteObject, XRF 272, 346–347
    - DeleteTempMap, XRF 347
    - Detach, XRF 347
    - FromHandle, XRF 348
    - GetObject, XRF 348–349
    - GetSafeHandle, XRF 349
    - UnrealizeObject, XRF 350
  - resetting brush origin or logical palette, CGdiObject::UnrealizeObject, XRF 350
- CGdiObject constructor, XRF 344
- CGdiObject destructor, XRF 345
- CGdiObject objects
  - brushes, resetting, CGdiObject::UnrealizeObject, XRF 350
  - creating, CGdiObject::CGdiObject, XRF 344
  - deleting
    - objects, CGdiObject::DeleteObject, XRF 346
    - temporary, CGdiObject::DeleteTempMap, XRF 347
  - destroying, CGdiObject::~CGdiObject, XRF 345
  - detaching Windows GDI object from, CGdiObject::Detach, XRF 347

CGdiObject objects (*continued*)
  filling buffer with data definition,
    CGdiObject::GetObject, XRF 348
  handles, CGdiObject::m_hObject, XRF 351
  palettes, resetting, CGdiObject::UnrealizeObject,
    XRF 350
  retrieving handle to,
    CGdiObject::CreateStockObject, XRF 345–346
  returning handle to, CGdiObject::GetSafeHandle,
    XRF 349
  returning pointer with GDI object handle,
    CGdiObject::FromHandle, XRF 348
  selecting, CDC::SelectObject, XRF 232–233
  selecting, CDC::SelectStockObject, XRF 234–235
_cgets function, LIB 136–137
CH register, CodeView syntax, ET 419, 450
Chaining interrupts between handlers, _chain_intr
    function, LIB 138–139
ChangeClipboardChain member function
    CWnd class, XRF 661
Changing
  current drives, _chdir function, LIB 142–143
  CWnd object position and dimensions,
    CWnd::MoveWindow, XRF 714–715
  directories, _chdir function, LIB 140–141
  file size, _chsize function, LIB 146–147
  file-permission settings, _chmod function,
    LIB 144–145
  font text output orientation
    _getgtextvector function, LIB 366
    _setgtextvector function, LIB 665
  fonts
    called when, CWnd::OnFontChange,
      XRF 738–739
  memory
    block size, _expand functions, LIB 255–257
    segment size, _dos_setblock function,
      LIB 216–217
  menu items, CMenu::ModifyMenu, XRF 432–433
  palettes, graphics, PT 175–176
  position
    relative to stream beginning, streambuf::seekpos,
      XRF 930

Changing (*continued*)
    stream, ostream::seekp, XRF 904
    streambuf objects, streambuf::seekoff, XRF 929
Chaining, file length, CFile::SetLength, XRF 318
char type
  conversion, LR 142
  described, LR+ 50–51
  integral promotion, portability guidelines,
    PT 285–287
  portability guidelines, PT 272
  size, LR+ 51–52
  usage, LR 98
Character arrays
  initializing, LR+ 222–223
  returning pointer to string stream's, istrstream::str,
    XRF 892
  scanning strings for characters, strpbrk and
    _fstrpbrk functions, LIB 776–777
  table, LR 236
Character classification and conversion functions
    (list), LIB 19
Character classification functions, portability
    guidelines, PT 284–285
Character constants
  C++, LR+ 16–19
  described, LR 16
  unrepresented, LR 267
Character devices, checking, _isatty function,
    LIB 441
Character range, regular expression syntax, ET 779
Character sets
  ANSI, converting to OEM, CString::AnsiToOem,
    XRF 576
  compared, LR 18
  mapping translation phase, LR 191
  OEM, converting to ANSI, CString::OemToAnsi,
    XRF 588
  portability guidelines, PT 284–285
Character strings
  defined, ET 805
  getting from console, _cgets function, LIB 136–137
Character-font functions, LIB 22

Characters
  appending from strings, strncat and _fstrncat functions, LIB 765–766
  ASCII compatibility, LR 236
  ASCII, defined, ET 804
  backslash (\), LR 18
  backspace escape sequence (\b), LR 18
  bell (\a), LR 18
  carriage-return escape sequence (\r), LR 18
  case distinction, LINK option, ET 585
  changing type, CL option, ET 526–527
  classes
    PWB, ET 92
    regular expression syntax, ET 778, 780–781, 786–787
  comparing
    from two strings, strncmp and _fstrncmp functions, LIB 767–769
    in two buffers, case-sensitive, _memicmp and _fmemicmp functions, LIB 506–507
    in two buffers, memcmp and _fmemcmp functions, LIB 500–502
    of two strings, _strnicmp and _fstrnicmp functions, LIB 772–773
  controlling, HELPMAKE, ET 712
  converting
    between uppercase and lowercase, LIB 19
    multibyte to wide, mbtowc and _fmbtowc functions, LIB 491–493
    series of wide to multibyte, wcstombs and _fwcstombs functions, LIB 867–868
    __toascii, tolower, toupper functions, LIB 817–819
    wide to multibyte, wctomb and _fwctomb functions, LIB 869–870
  copying
    between buffers, memcpy and _fmemcpy functions, LIB 503–505
    from buffers, _memccpy and _fmemccpy functions, LIB 496–497
  dead-key, returning value, CWnd::OnDeadChar, XRF 727–729
  deleting
    PWB, ET 161, 167–168, 209–210
  double-quotation-mark escape sequence (\"), LR 18
  edit control, getting starting, ending character positions, XRF 293

Characters (*continued*)
  extracting
    from stream and discarding, istream::ignore, XRF 880
    putting back into stream, istream::putback, XRF 882
  fill, setting for stream, setfill, XRF 869
  finding
    in buffers, memchr and _fmemchr functions, LIB 498–499
    in strings, strchr and _fstrchr functions, LIB 740–742
  formatting and printing to console, _cprintf function, LIB 165–166
  formfeed escape sequence (\f), LR 18
  getting from console, _getch and _getche functions, LIB 348–349
  getting pixel bitmaps for specified characters, _pg_getchardef function, LIB 564
  hexadecimal escape sequence, LR 19–20
  horizontal tab escape sequence (\t), LR 18
  index
    of line, CEdit::LineIndex, XRF 295
    retrieving line number from multiple-line edit control, CEdit::LineFromChar, XRF 294
  inserting, PWB, ET 173–174
  inserting into output stream, ostream::put, XRF 903
  international, LR 8
  makefiles, NMAKE, ET 653
  matching, ET 779
  mnemonic, called when user presses, CWnd::OnMenuChar, XRF 756–757
  moving to another segment, _movedata function, LIB 525–526
  multibyte, LR 8, 236
    converting to wide, mbstowcs and _fmbstowcs functions, LIB 489–490
    getting length, determining validity, mblen and _fmblen functions, LIB 487–488
  newline, inserting into output streams, ostream& endl, XRF 907
  nongraphic control, LR 1
  null, LR 249
  null-terminator, inserting into output streams, ostream& ends, XRF 907
  number of positions to scroll horizontally, CEdit::LineScroll, XRF 296
  octal escape sequences, LR 19–20

Characters (*continued*)
of a string, initializing to given characters, _strnset and _fstrnset functions, LIB 774–775
passwords, setting or removing in edit control, CEdit::SetPasswordChar, XRF 299
pixel bitmaps, setting, _pg_setchardef function, LIB 573
predefined expression syntax, ET 778, 785
presentation graphics character pool, PT 219
pushing back
last read from console, _ungetch function, LIB 829–830
onto streams, ungetc function, LIB 827–828
range of values, LR 238
reading from streams
fgetc and _fgetchar functions, LIB 273–274
getc and getchar functions, LIB 346–347
retrieving current font width, CDC::GetCharWidth, XRF 194
retrieving intercharacter spacing, CDC::GetTextCharacterExtra, XRF 206
returning count of characters in CString object, CString::GetLength, XRF 584
returning without extracting, istream::peek, XRF 882
reversing in strings, _strrev and _fstrrev functions, LIB 780–781
scanning strings
for last occurrence, strrchr and _fstrrchr functions, LIB 778–779
for specified character sets, strpbrk and _fstrpbrk functions, LIB 776–777
searching for first match in string, CString::FindOneOf, XRF 581
selecting, combo box edit control, CComboBox::SetEditSel, XRF 154
setting
buffers to specified, memset and _fmemset functions, LIB 513–514
characters in strings to, _strset and _fstrset functions, LIB 782–783
intercharacter spacing, CDC::SetTextCharacterExtra, XRF 247
range in edit control, CEdit::SetSel, XRF 301

Characters (*continued*)
single-quotation-mark escape sequence (\'), LR 18
soft line-break, inserting in multiple-line edit control, CEdit::FmtLines, XRF 290
special, NMAKE, ET 668
streams, returning number extracted by last unformatted input function, istream::gcount, XRF 877
string literals, LR 20
strings
computing width, height, CDC::GetTabbedTextExtent, XRF 204–205
returning character specified by index, CString::GetAt, XRF 581
testing
for specified conditions, is functions, LIB 437–440
individual, LIB 19
types, LR 17
unformatted input function, istream::gcount, XRF 877
vertical-tab escape sequence (\v), LR 18
white-space, LR 2
wide, LR 8, 237
writing
strings to regions, CDC::ExtTextOut, XRF 188–189
strings to specified location, XRF 258–259
to console, _putch function, LIB 595–596
to streams, fputc and _fputchar functions, LIB 301–302
CHAR_BIT constant, C++
integral limits, LR+ 62
Charizing operator
C, LR 195–197
C++, LR+ 372
CHAR_MAX constant, C++
integral limits, LR+ 62
CHAR_MIN constant, C++
integral limits, LR+ 62
Chart windows, presentation graphics, PT 204
_chartenv structures, presentation graphics, PT 219–220, 226–227

- Charts
  - *See also* Presentation graphics
  - axes
    - described, PT 204
    - structure types, PT 221–223
  - bar charts
    - described, PT 203
    - sample program, PT 208–212
    - styles, PT 204–205
  - categories, PT 202
  - chart windows, PT 204
  - column charts
    - described, PT 203
    - sample program, PT 208–212
    - style, PT 204–205
  - data series, PT 202
  - data windows, PT 204
  - displaying
    - pie, _pg_chartpie function, LIB 558–559
    - series or multiseries, _pg_chart functions, LIB 555–557
  - environment variables, PT 219–227
  - features described, PT 202–205
  - initializing environment, _pg_defaultchart function, LIB 562–563
  - labels, PT 202
  - legends
    - described, PT 205
    - structure types, PT 225
  - line graphs
    - described, PT 203
    - sample program, PT 208–212
    - styles, PT 204–205
  - pie charts
    - described, PT 203
    - sample program, PT 206–208
    - styles, PT 204–205
  - presentation-graphics, LIB 30
  - scatter. *See* Scatter charts
  - scatter diagrams
    - described, PT 203–204
    - sample program, PT 212–214
    - styles, PT 204–205
  - styles
    - described, PT 204–205
    - pool, PT 216
  - types described, PT 202–205
  - values, PT 203
- Charts (*continued*)
  - windows
    - chart, PT 204
    - data, PT 204
    - structure types, PT 223–224
- _CHAR_UNSIGNED macro, C++, LR+ 374
- _chdir function, LIB 140–141
- _chdrive function, LIB 142–143
- Check boxes
  - getting check state, CButton::GetCheck, XRF 132
  - PWB, ET 82
  - setting check state, CButton::SetCheck, XRF 134
- Check marks, in pop-up menu, CMenu::CheckMenuItem, XRF 419–420
- check_pointer pragma
  - described, LR 210
  - /ZR option, CL, ET 556–557
- check_pointer pragma, C++, LR+ 387
- check_stack pragma, PT 21
  - described, LR 210
  - removing stack probes, ET 518–520
- check_stack pragma, C++, LR+ 387
- CheckDlgButton member function, CWnd class, XRF 661–662
- CheckForSave member function, Foundation classes tutorial, XUG 240
- Checking
  - character device, _isatty function, LIB 441
  - console for keyboard input, _kbhit function, LIB 444
  - equality between sizes, CSize::operator==, XRF 560
  - heaps, _heapset functions, LIB 415–417
  - inequality between sizes, CSize::operator!=, XRF 560
  - video configuration, PT 171–172
- Checking stacks, disabling, PT 21
- Check-mark control, CWnd button control, CWnd::IsDigButtonChecked, XRF 709
- Check-marking, button control, CWnd::CheckDlgButton, XRF 661
- CheckMenuItem member function, CMenu class, XRF 419–420
- Checkpoint member function, CMemoryState class, memory leaks, detecting, XUG 292
- CheckRadioButton member function, CWnd class, XRF 662
- Child processes
  - creating, executing, _spawn functions, LIB 717–722
  - defined, ET 805; LIB 51
  - loading and executing, _exec functions, LIB 246–250

Child windows
  activating, next child,
    CMDIFrameWnd::MDINext, XRF 407
  buttons as, XRF 16
  called on activation or deactivation,
    CWnd::OnMDIActivate, XRF 753–754
  called upon creation or destruction,
    CWnd::OnParentNotify, XRF 774–775
  called when about to be drawn,
    CWnd::OnCtlColor, XRF 726–727
  called when changing size or position,
    CWnd::OnChildActivate, XRF 720
  changing
    parent, CWnd::SetParent, XRF 810
    size, position, ordering, CWnd::SetWindowPos,
      XRF 814–816
  classes (list), XRF 6
  CMDIChildWnd class described, XRF 395
  creating
    and attaching, CMDIChildWnd::Create, XRF 397
    constructor, CMDIChildWnd::CMDIChildWnd,
      XRF 397
  determining which contains specified point,
    CWnd::ChildWindowFromPoint, XRF 662–663
  flashing once, CWnd::FlashWindow, XRF 683–684
  handling activation message,
    CMDIChildWnd::MDIActivate, XRF 398
  identifying, CWnd::IsChild, XRF 709
  MDI
    activating, CMDIFrameWnd::MDIActivate,
      XRF 405
    arranging in cascade,
      CMDIFrameWnd::MDICascade, XRF 406
    arranging in tiled format,
      CMDIFrameWnd::MDITile, XRF 409
    destroying, CMDIChildWnd::MDIDestroy,
      XRF 399
    maximizing, CMDIChildWnd::MDIMaximize,
      XRF 399, CMDIFrameWnd::MDIMaximize,
      XRF 407

Child windows (*continued*)
  MDI (*continued*)
    restoring, CMDIChildWnd::MDIRestore,
      XRF 399,
      CMDIFrameWnd::MDIRestore, XRF 408
    returning active child,
      CMDIFrameWnd::GetChildFrame, XRF 405
    returning current,
      CMDIFrameWnd::MDIGetActive, XRF 406
  messages, differentiating, Foundation classes
    cookbook, XUG 317
  minimized, arranging,
    CWnd::ArrangeIconicWindows, XRF 659
  notification messagesm, Foundation classes
    cookbook, XUG 316–317
  QuickWin
    closing, PT 162
    displaying, PT 147–148
    (list), PT 150
    opening, PT 151, 157–158
    reading from, PT 159–160
    sizing, positioning, PT 153, 160
    writing to, PT 159–160
  returning parent MDI frame,
    CMDIChildWnd::GetParentFrame, XRF 398
  top-level, searching for, CWnd::GetTopWindow,
    XRF 700
  Windows, creating
    attaching to CWnd object, CWnd::Create,
      XRF 664–665
    constructor, CWnd::CWnd, XRF 673
ChildWindowFromPoint member function, CWnd
    class, XRF 662–663
_chmod function, LIB 144–145
Choosing
  between functions and macros, LIB 9–11
  QuickWin menu items, _wmenuclick function,
    LIB 880–881
Chord member function, CDC class, XRF 170–171
Chords, drawing, CDC::Chord, XRF 170–171

_chsize function, LIB 146–147
cin, TUT 8, predefined stream object, XRF 875
CL
  batch files, executing, ET 490
  calling conventions, ET 516–518, registers, ET 520
  command line, ET 485–486
  compiling, without linking, ET 491
  constants, ET 491–493
  data allocation, ET 523–524
  data threshold, setting, ET 522
  debugging, ET 553
  DOS device names, ET 497–498
  entry/exit codes, optimizing, ET 515, 522–523
  environment variables
    NO87, ET 513
    specifying options, ET 557–559
  error messages. *See* "Error Messages," Part 2 of this book
  extensions, specifying, ET 496
  filename extensions, processing, ET 486
  files
    assembly, listing, ET 501–503, 505
    browser, generating, ET 507–508
    executable, creating, ET 486–487
    machine-code, listing, ET 502–503, 505
    map, creating, ET 505–507
    optimizing size, ET 538
    renaming, ET 498–499
    source, listing, ET 501
  floating-point math operations, ET 508–513
  function prototypes, ET 552–553
  function-level linking, ET 524
  intrinsic functions, generating, ET 534–535
  language specification, ET 542
  linker-control options, ET 527
  macros, defining, ET 491–493
  memory capacity, increasing, ET 491
  naming conventions, ET 518
    Pascal, ET 552
    segments, ET 528–529
  optimizing
    aliasing, ET 531–532
    execution time, ET 539
    exit sequence, ET 538
    file size, ET 538
    float consistency, ET 537–538
    frame sorting, ET 539
    inline expansion control, ET 532
    intrinsic function generation, ET 534–535
    loops, ET 535–537
    maximum, ET 539
CL (*continued*)
  optimizing (*continued*)
    p-code quoting, ET 533
    post-code generation, ET 536–537
    register allocation, ET 533
    subexpressions, ET 533
    turning off, ET 533
  options, ET 488–557
    debugging considerations, ET 323–324
    interaction between, ET 496, 512
    linker control, ET 527
    specifying, ET 557–559
  output files, setting alternates, ET 495
  paths, specifying, ET 496
  p-code, removing entry points, ET 520–521, 533
  preprocessing
    copying output, ET 493–494
    output file, creating, ET 540
    preserving comments, ET 491
  processor-specific instructions, ET 514–515
  segments, ET 529–530
  specifying entry tables, ET 521
  stacks
    checking, ET 518–520
    size, setting, ET 494
  syntax, ET 488, 557
  warning level, setting, ET 544–545
CL options
  /AH, determining pointer size, LR 60
  /AL, determining pointer size, LR 60
  /AM, determining pointer size, LR 60
  /AS, determining pointer size, LR 60
  /Au, has same effect as __loadds attribute, LR 175
  /B1, unsupported, LR 267
  /B2, unsupported, LR 267
  /B3, unsupported, LR 267
  /Bm, set memory amount, LR 260
  /D, defining macros/constants, LR 195, 206, 267
  /f, specifying precompiled header filename, LR 260
  /Gc, force calling conventions, LR 170, 175, 235
  /Gd, force __cdecl calling convention, LR 170
  /GD, optimize entry/exit code, LR 260
  /GE, optimize entry/exit code, LR 260
  /Gi, unsupported, LR 267
  /Gn, saving bytes, LR 212, 260
  /Gp, specify maximum number of entry tables, LR 260
  /Gq, specify real mode Windows, LR 260
  /Gr, cause functions to compile as fastcall, LR 170, 175
  /Gx, assume all data is near, LR 260

CL options (*continued*)
/Gy, enable function-level linking, LR 260
/Gz, specify __stdcall, LR 170
/H, restrict external name length, LR 6, 235
/J, changing default char type, LR 52, 98
/Lc, unsupported, LR 267
/Ld, control library selection for DLLs, LR 260
/Li, unsupported, LR 267
/Lp, unsupported, LR 267
/Lu, link without C run-time startup code, LR 260
/Lw, control library selection for applications, LR 260
/MD, unsupported, LR 267
/ML, unsupported, LR 267
/MT, unsupported, LR 267
/ND, name-data segment, LR 175
new in version 7.0, LR 261
/NQ, name temporary segment for p-code, LR 260
/NV, name temporary segment, LR 260
/O, optimizations, LR 212
/Ob, control inline expansion, LR 260
/Ob2, inhibit expansion, LR 210–211
/Of, p-code quoting, LR 261
/Oi, intrinsic functions, LR 211
/Oo, post-code-generation optimizing, LR 261
/Oq, turn on p-code optimization, LR 260
/Ov, sort local variables by frequency of use, LR 261
/P, debugging macros, LR 196
/qc, restrictions, LR 211
/Sl, specifying number of characters per line, LR 212
/Tp, specify C++ source file, LR 261
/u, turn off every defined identifer, LR 267
/W4, generating name length warning, LR 6
/Yc, create precompiled header, LR 261
/Yd, debugging information, LR 261
/Yu, use precompiled header file, LR 261
/Za, disabling Microsoft extensions, LR 3–4, 169, 200, 261
/Zc, ignore case, LR 7, 235
/Ze, __STDC__ macro, LR 199, 261, 265

CL options (*continued*)
/Zf, accept __far keyword, LR 261
/Zn, turn off SBRPACK utility, LR 261
/Zp, controlling packing, LR 70, 212
/Zr, checking pointer dereference, LR 210
CL register, CodeView syntax, ET 419, 450
Class declarations
CMainWindow, adding, XUG 200–202
friends, defining in, LR+ 295
Class hierarchies
abstract classes, TUT 128
overview, TUT 113
Class Hierarchy command, PWB
described, ET 76
function, ET 145
CLASS keyword, module-definition files, ET 619–620
class keyword, C++
class type declaration, LR+ 228
Class scope, C++
described, LR+ 29, 282–284
linkage rules, LR+ 34
type names in, LR+ 257
Class Tree command, PWB, ET 76, 145
class type names, C++
introduction by declaration statements, LR+ 149
Classes
See also Foundation classes
abstract, LR+ 265–266, 280–281; TUT 128
abstraction, TUT 164–165
ambient memory models
described, PT 98–99
overriding default, PT 99–100
anonymous, LR+ 229–230
arrays of objects, TUT 94–95
base, TUT 111
access specifiers, LR+ 287–290
described, LR+ 266
multiple, LR+ 267–271
name ambiguities, LR+ 271–274
virtual, LR+ 268–271

Classes (*continued*)
- CDataBase
  - applications, XUG 140
  - Foundation classes tutorial, XUG 123, 134–140
  - member functions, XUG 135–136
- CEditDialog, Foundation classes tutorial, XUG 163–164
- CFindDialog
  - dialog boxes, XUG 163–164
  - Foundation classes tutorial, XUG 153, 163
- characters, regular expression syntax, ET 778, 780–781, 786–787
- CMainWindow, XUG 91, 121, 199
- CodeView accessibility, ET 409–411
- compared with C structures, TUT 41
- composed
  - construction, LR+ 305
- composite
  - design issues, TUT 176
  - overview, TUT 58–60
- concrete, TUT 128
- CPerson, XUG 129
- customized memory management, TUT 103–104, 106
- data members, TUT 44
- declaring, LR+ 228, 238; TUT 42
- defined, TUT 41
- defining, LR+ 228–230
- derivation, XRF 10
- derivative types, LR+ 58
- derived, TUT 111
  - described, LR+ 259
  - multiple inheritance, LR+ 264–265, 267–274, 297
  - single inheritance, LR+ 259–264
  - virtual functions, LR+ 265
- described, LR+ 227–228
- design issues
  - assigning responsibilities, TUT 174–175
  - common mistakes, TUT 175
  - hierarchies, TUT 177–180
  - identifying, TUT 173–174
  - overview, TUT 172
  - relationships, TUT 176–177
- empty
  - declaring, LR+ 232
- friends, TUT 89–91, 93
  - declaring as, LR+ 293–294
- general purpose, categorized by function (list), XRF 21

Classes (*continued*)
- grammar summary, LR+ 431–433
- header and source files, TUT 60–62
- hierarchies
  - design issues, TUT 169–170, 178–180
  - overview, TUT 177–178
- inheritance, TUT 110
- lifetime of objects, TUT 49–50
- member functions, TUT 45
- members. *See* Members
- memory models
  - *See also* ambient memory models, PT 98
  - overview, PT 97–98
  - return object addressing modes, PT 101–102
  - this pointer, overloading, PT 100–101
  - v-table pointers, PT 102–103
- names, LR+ 31–32, 232–234
- nested, LR+ 254–257
- pointers to, conversion, to pointers to base classes, LR+ 72–73
- references to, conversion to references to base classes, LR+ 75
- requiring destructors, TUT 69
- run-time information, supplying, DECLARE_DYNAMIC macro, XRF 38–39
- scope, *See* Class scope
- storage. *See* Storage classes, C++
- Windows classes
  - described, XRF 5
  - registering, AfxRegisterWndClass, XRF 37
  - summary of, XRF 5

Class-type objects, C++, LR+ 230–231

Cleanup operations during process, _cexit and _c_exit functions, LIB 135

Clear member function
- CComboBox class, XRF 143
- CEdit class, XRF 285

clear member function
- ios class, XRF 856
- ofstream class, iostream, XUG 377

Clear Paste command, QuickWin, PT 150

_clear87 function, LIB 148–149

clearerr function, LIB 150–151

Clearing
- breakpoints in CodeView, ET 367, 426–427
- edit control
  - selection, CEdit::Clear, XRF 285
  - undo flag, CEdit::EmptyUndoBuffer, XRF 290
- error-bits, ios::clear, XRF 856

- Clearing (*continued*)
  - floating-point status word, _clear87 function, LIB 148–149
  - format flags
    - ios::unsetf, XRF 864
    - streams, XRF 869
  - screen area, _clearscreen function, LIB 152–153
- Clearmsg function, PWB, ET 150, 162
- _clearscreen function, LIB 152–153; PT 187
- Clearsearch function, PWB, ET 150, 162
- Click, defined, ET 805
- Client, coordinates, converting to screen coordinates, CWnd::ClientToScreen, XRF 663
- Client areas
  - called after size changed, CWnd::OnSize, XRF 784–785
  - called when needing repainting, CWnd::OnPaintClipboard, XRF 772–773
  - called when size changed and clipboard contains data, CWnd::OnSizeClipboard, XRF 785
  - calling windows, CClientDC class, XRF 137
  - CClientDC class described, XRF 137
  - converting screen coordinates of point or rect to client coordinates, CWnd::ScreenToClient, XRF 802
  - CWnd, copying client coordinates into specified structure, CWnd::GetClientRect, XRF 686–687
  - device contexts
    - creating, CClientDC::CClientDC, XRF 138
    - destroying, CClient::~CClientDC, XRF 138
  - invalidating
    - CWnd::Invalidate, XRF 706
    - entire, CWnd::Invalidate, XRF 707
    - within given rectangle, CWnd::InvalidateRect, XRF 707
    - within given region, CWnd::InvalidateRgn, XRF 708
  - painting
    - information, PAINTSTRUCT structure, XRF 84–85
    - window associated with CPaintDC object, XRF 500
- Client areas (*continued*)
  - scrolling, CWnd::ScrollWindow, XRF 803–804
  - updating
    - CWnd::UpdateWindows, XRF 819
    - matching colors, CDC::UpdateColors, XRF 260
  - validating within given region, CWnd::ValidateRgn, XRF 820
- Client windows, QuickWin, user interface, PT 147–148
- CLIENTCREATESTRUCT structure, CMDIFrameWnd::CreateClient, XRF 404
- ClientToScreen member function, CWnd class, XRF 663
- Clipboard
  - called for each window in viewer chain when contents change, CWnd::OnDrawClipboard, XRF 731–732
  - calling owner when emptied, CWnd::OnDestroyClipboard, XRF 731
  - combo box edit control
    - copying current selection to, CComboBox::Copy, XRF 143
    - copying deleted selection to, CComboBox::Cut, XRF 146
    - inserting data, CComboBox::Paste, XRF 152
  - copying edit control selection to, CEdit::Copy, XRF 286
  - CWnd, called with event in vertical scroll bar, CWnd::OnVScrollClipboard, XRF 797–798
  - defined, ET 805
  - format, specifying, CWnd::OnRenderFormat, XRF 781
  - formats (list), XRF 86–87
  - opening, CWnd::OpenClipboard, XRF 799
  - owner
    - called when application is destroyed, CWnd::OnRenderAllFormats, XRF 780–781
    - retrieving, CWnd::GetClipboardOwner, XRF 687
  - QuickWin
    - copying, PT 148–149
    - pasting, PT 149

Clipboard Results command, PWB, ET 77
Clipboard viewers
called for displaying clipboard contents, CWnd::OnAskCbFormatName, XRF 716–717
called for each window in chain when clipboard contents change, CWnd::OnDrawClipboard, XRF 731–732
called when client area needs repainting, CWnd::OnPaintClipboard, XRF 772–773
called when client area size changed, CWnd::OnSizeClipboard, XRF 785
chain
adding current CWnd to, CWnd::SetClipboardViewer, XRF 807
retrieving first window in, CWnd::GetClipboardViewer, XRF 687
horizontal scrolling, CWnd::OnHScrollClipboard, XRF 742–743
removing CWnd from chain, CWnd::ChangeClipboardChain, XRF 661
removing windows from chain, CWnd::OnChangeCbChain, XRF 717–718
Clipping region
creating
CDC::IntersectClipRect, XRF 212
CDC::ExcludeClipRect, XRF 185
device contexts, specifing whether point is within, CDC::PtVisible, XRF 223
moving, CDC::OffsetClipRgn, XRF 216
rectangles, determining if within, CDC::RectVisible, XRF 225
selecting given region as current, CDC::SelectClipRgn, XRF 231
smallest bounding rectangle dimensions, CDC::GetClipBox, XRF 195
CList boxes, setting, tab-stop positions, CListBox::SetTabStops, XRF 371
CListBox class
adding strings, CListBox::AddString, XRF 355
bounding rectangle, retrieving dimensions, CListBox::GetItemRect, XRF 363
described, XRF 352–354
Foundation classes cookbook, XUG 335
getting string length, CListBox::GetTextLen, XRF 366
CListBox class (*continued*)
items
deleting, CListBox::DeleteString, XRF 359
ensuring visibility, CListBox::SetTopIndex, XRF 372
removing, CListBox::ResetContent, XRF 367
CListBox::GetCurSel, XRF 362
retrieving index of, CListBox::GetItemData, XRF 363
retrieving index of first visible, CListBox::GetTopIndex, XRF 366
retrieving number of, CListBox::GetCount, XRF 361
retrieving selection state, CListBox::GetSel, XRF 364
retrieving total selected, CListBox::GetSelCount, XRF 364
retrieving zero-based index of currently selected,
searching for matching, CListBox::SelectString, XRF 367
selecting consecutive, CListBox::SelItemRange, XRF 368
setting associated 32-bit values, CListBox::SetItemData, XRF 370
member functions
AddString, XRF 355
CListBox, XRF 355
Create, XRF 356–359
DeleteString, XRF 359, 729
Dir, XRF 360
FindString, XRF 361
GetCount, XRF 361
GetCurSel, XRF 362
GetHorizontalExtent, XRF 362
GetItemData, XRF 363
GetItemRect, XRF 363
GetSel, XRF 364
GetSelCount, XRF 364
GetSelItems, XRF 364–365
GetText, XRF 365
GetTextLen, XRF 366
GetTopIndex, XRF 366
InsertString, XRF 366–367
ResetContent, XRF 367, 729
SelectString, XRF 367–368
SelItemRange, XRF 368
SetColumnWidth, XRF 369

- CListBox class (*continued*)
  - member functions (*continued*)
    - SetCurSel, XRF 369
    - SetHorizontalExtent, XRF 370
    - SetItemData, XRF 370
    - SetSel, XRF 371
    - SetTabStops, XRF 371–372
    - SetTopIndex, XRF 372
  - multicolumn list box, selecting width, CListBox::SelColumnWidth, XRF 369
  - retrieving horizontal scroll event, CListBox::GetHorizontalExtent, XRF 362
  - scrolling
    - selected strings, CListBox::SelCurSel, XRF 369
    - setting width, CListBox::SetHorizonalExtent, XRF 370
  - selecting strings in multiple-selection, CListBox::SetSel, XRF 371
- CListBox constructor, XRF 355
- CListBox objects, creating
  - constructor, CListBox::CListBox, XRF 355
  - specifying style, CListBox::Create, XRF 356–359
- clock function, LIB 154–155; LR 255
- clog, predefined stream object, XRF 900
- Close All command, PWB
  - described, ET 77
  - predefined macros, ET 145
- Close command
  - CodeView, ET 373–374
  - PWB, ET 72, 77, 142–145
- _close function, LIB 156–157
- Close member function
  - CArchive class, XRF 96–97
  - CFile class, XRF 308
    - CArchive::Flush, XRF 97
    - Foundation classes cookbook, XUG 278
  - CMetaFileDC class, XRF 440
- close member function
  - filebuf class, XRF 832
  - fstream class, XRF 838
  - ifstream class, XRF 846
  - istream class, XUG 376–377
  - ofstream class, XRF 894; XUG 391
- Close Project command, PWB, ET 74
- Closefile function, PWB, ET 150, 163
- CloseWindow member function
  - CWnd class, XRF 664
- Closing
  - child windows, QuickWin, PT 162
  - file associated with filebuf object, fstream::close, XRF 838
  - files
    - associated with filebuf object, XRF 894
    - attached to filebuf object, filebuf::close, XRF 832
    - CFile object, CFile::Close, XRF 308
    - _close function, LIB 156–157
    - _dos_close function, LIB 185–186
    - filebuf objects, ifstream::close, XRF 846
    - Foundation classes cookbook, XUG 278
    - memory, CMemFile::~CMemFile, XRF 412
    - PWB, ET 72, 233
  - Help files
    - PWB, ET 213–214
    - QuickHelp, ET 769
  - menus, PWB, ET 79
  - operating system file, CFile::~CFile, XRF 308
  - projects, PWB, ET 234
  - streams, fclose and _fcloseall functions, LIB 260–261
  - windows, PWB, ET 220, 232–233
- CLRFILE.CVW
  - CodeView, ET 343–344
    - and Colors command, ET 370–371
    - saving, ET 360
- CMainWindow class
  - database, Phone Book sample program, XUG 121
  - member functions, Phone Book sample program, XUG 199
  - message handlers, Phone Book sample program, XUG 199
- CMainWindow constructor
  - HELLO sample program, XUG 88–91
  - windows, creating, XUG 89–93
- CMapPtrToPtr class, XRF 373–374

- CMapPtrToWord class, XRF 375–376
- CMapStringToOb class
  - described, XRF 377–378
  - member functions
    - CMapStringToOb, XRF 379
    - ~CMapStringToOb, XRF 379
    - GetCount, XRF 380
    - GetNextAssoc, XRF 380–381
    - GetStartPosition, XRF 381
    - IsEmpty, XRF 382
    - Lookup, XRF 382
    - RemoveAll, XRF 383
    - RemoveKey, XRF 383–384
    - SetAt, XRF 384–385
  - operators, XRF 386
- CMapStringToOb constructor, XRF 379
- CMapStringToOb destructor, XRF 379
- CMapStringToOb objects
  - constructing, CMapStringToOb::CMapStringToOb, XRF 379
  - destroying, CMapStringToOb::~CMapStringToOb, XRF 379
- CMapStringToPtr class, XRF 387–388
- CMapStringToString class, XRF 389–390
- CMapWordToOb class, XRF 391–392
- CMapWordToPtr class, XRF 393–394
- CMDBOOK sample program, Foundation classes tutorial, XUG 6, 140
- CMDIChildWnd class
  - data members, m_pMDIFrameWnd, XRF 400
  - described, XRF 395–396
  - member functions
    - CMDIChildWnd, XRF 397
    - Create, XRF 397–398
    - GetParentFrame, XRF 398
    - MDIActivate, XRF 398
    - MDIDestroy, XRF 399
    - MDIMaximize, XRF 399
    - MDIRestore, XRF 399
- CMDIChildWnd constructor, XRF 397
- CMDIChildWnd object, Foundation classes cookbook, XUG 311, 323–325
- CMDIFrameWnd class
  - data members, m_hWndMDIClient, XRF 410
  - described, XRF 401–402
  - member functions
    - CMDIFrameWnd, XRF 403
    - Create, XRF 403–404
    - CreateClient, XRF 404
    - GetChildFrame, XRF 405
    - MDIActivate, XRF 405, 754
- CMDIFrameWnd class (*continued*)
  - member functions (*continued*)
    - MDICascade, XRF 406
    - MDIGetActive, XRF 406
    - MDIIconArrange, XRF 406, 659
    - MDIMaximize, XRF 407
    - MDINext, XRF 407
    - MDIRestore, XRF 408
    - MDISetMenu, XRF 408–409
    - MDITile, XRF 409
- CMDIFrameWnd constructor, XRF 403
- CMDIFrameWnd objects
  - creating, CMDIFrameWnd::CMDIFrameWnd, XRF 403
  - Foundation classes cookbook, XUG 311, 323–325
  - Windows MDI frame window for, creating, CMDIFrameWnd::Create, XRF 403
- !CMDSWITCHES preprocessing directive, NMAKE, ET 688–689
- CMemFile class
  - described, XRF 411
  - member functions
    - CMemFile, XRF 412
    - ~CMemFile, XRF 412
- CMemFile constructor, XRF 412
- CMemFile destructor, XRF 412
- CMemoryException class
  - described, XRF 413
  - member function, CMemoryException, XRF 413
- CMemoryException constructor, XRF 413
- CMemoryException exception handler, Foundation classes cookbook, XUG 298
- CMemoryException objects, creating, CMemoryException::CMemoryException, XRF 413
- CMemoryState objects
  - comparing two, CMemoryState::Difference, XRF 55
  - creating, CMemoryState::CMemoryState, XRF 54
- CMemoryState::DumpAllObjectsSince, XRF 55
- CMemoryState::DumpStatistics, XRF 56
- CMenu class
  - described, XRF 19, 414–415
  - member functions
    - AppendMenu, XRF 416–418, 699
    - Attach, XRF 419
    - CheckMenuItem, XRF 419–420
    - CMenu, XRF 420
    - ~CMenu, XRF 421
    - CreateMenu, XRF 421
    - CreatePopupMenu, XRF 421–422
    - DeleteMenu, XRF 422

- CMenu class (*continued*)
  - member functions (*continued*)
    - DestroyMenu, XRF 423, 810
    - Detach, XRF 423
    - EnableMenuItem, XRF 423–424
    - GetMenuItemCount, XRF 424
    - GetMenuItemID, XRF 425
    - GetMenuState, XRF 425–426
    - GetMenuString, XRF 427
    - GetSubMenu, XRF 428
    - InsertMenu, XRF 428–430, 699
    - LoadMenu, XRF 430
    - LoadMenuIndirect, XRF 431
    - ModifyMenu, XRF 432–433, 699
    - RemoveMenu, XRF 434
    - SetMenuItemBitmaps, XRF 435–436
    - TrackPopupMenu, XRF 436–437
  - Phone Book sample program, XUG 119
- CMenu constructor, XRF 420
- CMenu destructor, XRF 421
- CMenu object
  - creating, CMenu::CMenu, XRF 420
  - destroying menus, CMenu::DestroyMenu, XRF 423
  - detaching menus, CMenu::Detach, XRF 423
  - loading, CMenu::LoadMenu, XRF 430
  - retrieving from pop-up, CMenu::GetSubMenu, XRF 428
- CMetaFileDC class
  - described, XRF 438–439
  - member functions
    - Close, XRF 440
    - CMetaFileDC, XRF 440
    - Create, XRF 441
    - SelectObject, XRF 110, 112, 334, 441–442
    - SelectStockObject, XRF 442
- CMetaFileDC constructor, XRF 440
- CMetaFileDC object
  - creating, XRF 440–441
- CModalDialog class
  - Cancel button action, overriding, CModalDialog::OnCancel, XRF 448
  - described, XRF 443–444
  - dialog boxes
    - described, XUG 106
    - modal, XUG 162–165
  - Foundation classes cookbook, XUG 329
  - HELLO sample program, XUG 83
  - invoking dialog box and returning result, CModalDialog::DoModal, XRF 447
  - member functions
    - CModalDialog, XRF 446
    - CreateIndirect, XRF 446–447
    - DoModal, XRF 447
    - OnCancel, XRF 448
    - OnOK, XRF 448
  - OK button, overriding in dialog boxes, CModalDialog::OnOK, XRF 448
- CModalDialog constructor, XRF 446
- CModalDialog objects
  - creating, XRF 446–447
  - Help dialogs, XUG 163
  - Phone Book sample program, XUG 151
- CNotSupportedException class
  - described, XRF 449
  - member function, XRF 449
- CNotSupportedException object, creating, XRF 449
- CNotSupportException exception handler, Foundation classes cookbook, XUG 298
- /CO option
  - debugging considerations, ET 324–325
  - LINK, ET 577, 743
- .COB files, ET 805
- CObArray class
  - described, XRF 450–452
  - member functions
    - Add, XRF 453
    - CObArray, XRF 454
    - ~CObArray, XRF 454
    - ElementAt, XRF 454–455
    - FreeExtra, XRF 455
    - GetAt, XRF 455
    - GetSize, XRF 456
    - GetUpperBound, XRF 456
    - InsertAt, XRF 456–457
    - operator, XRF 462

- CObArray class (*continued*)
  - member functions (*continued*)
    - RemoveAll, XRF 458
    - RemoveAt, XRF 458–459
    - SetAt, XRF 459–460
    - SetAtGrow, XRF 460–461
    - SetSize, XRF 461
- CObArray constructor, XRF 454
- CObArray destructor, XRF 454
- COBFLAGS options macro, NMAKE, ET 676
- CObject class
  - AssertValid member function, Foundation class tutorial, XUG 32
  - basic functionality, using, Foundation classes cookbook, XUG 264
  - deriving classes from, XUG 263
  - described, XRF 463–465
    - Foundation classes cookbook, XUG 263
    - Foundation classes tutorial, XUG 19
  - DMTEST sample program, Foundation classes tutorial, XUG 24, 28
  - executing specified iteration function for derived objects, AfxDoForAllObjects, XRF 49
  - functionality, levels of, Foundation classes cookbook, XUG 263
  - getting run-time structure, CObject::GetRunTimeClass, XRF 470
  - giving serialization capability, DECLARE_SERIAL macro, XRF 39–40
  - implementation files, Foundation classes cookbook, XUG 263
  - interface files, Foundation classes cookbook, XUG 263
  - IsKindOf function, using, Foundation classes cookbook, XUG 266
  - macros
    - DECLARE_DYNAMIC, XUG 264–266
    - IMPLEMENT_DYNAMIC, XUG 264–266
    - RUNTIME_CLASS, XUG 265–266
  - member functions
    - AssertValid, XRF 466
    - CObject, XRF 467
    - ~CObject, XRF 467
    - Dump, XRF 279, 469–470
    - GetRuntimeClass, XRF 470–471
    - IsKindOf, XRF 100, 472
    - IsSerializable, XRF 473
    - operator, XRF 475–476
    - Serialize, XRF 473
  - object diagnostics, XRF 24
- CObject class (*continued*)
  - performing optimal memory allocation, CObject::operator new, XRF 475
  - persistence described, XRF 23
  - run-time class information, XRF 25; XUG 264–267
  - serialization, Foundation classes cookbook, XUG 265, 279
  - services provided, XRF 23–26
  - validity checking, XRF 25
- CObject collection
  - deleting all objects, Foundation classes cookbook, XUG 273
- CObject constructor, XRF 467
- CObject destructor, XRF 467
- CObject objects
  - creating, CObject::CObject, XRF 467
  - destroying, CObject::~CObject, XRF 467
  - dumping to, CObject::Dump, XRF 469–470
  - reading or writing to archive, CObject::Serialize, XRF 473–474
  - testing
    - for class, CObject::IsKindOf, XRF 472
    - if eligible for serialization, CObject::IsSerializable, XRF 473
  - validity checking, CObject::AssertValid, XRF 466
- CObject pointer lists
  - adding element after specified position, CObList::InsertAfter, XRF 491–492
  - adding element before specified position, CObList::InsertBefore, XRF 492
  - creating, CObList::CObList, XRF 482–483
  - destroying, CObList::~CObList, XRF 483
  - getting number of elements in, CObList::GetCount, XRF 485
  - getting pointer representing head element, CObList::GetHead, XRF 486–487
  - getting position of head element, CObList::GetHead, XRF 487
  - getting position of next element, CObList::GetNext, XRF 488
  - getting position of previous element, CObList::GetPrev, XRF 489
  - getting tail element position, CObList::GetTailPosition, XRF 491
  - getting tail element, CObList::GetTail, XRF 490
  - indicating if containing no elements, CObList::IsEmpty, XRF 493
  - removing all elements from, CObList::RemoveAll, XRF 493
  - removing head element from, CObList::RemoveHead, XRF 495

- CObject pointer lists (*continued*)
  - removing specified element from, CObList::RemoveAt, XRF 494
  - removing tail element from, CObList::RemoveTail, XRF 496
  - retrieving pointer to given position, CObList::GetAt, XRF 485
  - searching for first matching pointer, CObList::Find, XRF 483
  - writing pointer to specified position, CObList::SetAt, XRF 496
- CObject pointers
  - arrays
    - adding element to end of, CObArray::Add, XRF 453
    - constructing empty, CObArray::CObArray, XRF 454
    - destroying, CObArray::~CObArray, XRF 454
  - CObArray class described, XRF 450
- CObList class
  - described, XRF 477–479
    - Foundation classes tutorial, XUG 20
  - member functions
    - AddHead, XRF 480
    - AddTail, XRF 481
    - CObList, XRF 379, 466, 482–483
    - ~CObList, XRF 483
    - Find, XRF 483–484
    - FindIndex, XRF 484
    - GetAt, XRF 485
    - GetCount, XRF 485–486
    - GetHead, XRF 486–487
    - GetHeadPosition, XRF 487
    - GetNext, XRF 488–489
    - GetPrev, XRF 489–490
    - GetTail, XRF 490
    - GetTailPosition, XRF 491
    - InsertAfter, XRF 491–492
    - InsertBefore, XRF 492–493
    - IsEmpty, XRF 493
    - RemoveAll, XRF 493–494
    - RemoveAt, XRF 494–495
    - RemoveHead, XRF 495
- CObList class (*continued*)
  - member functions (*continued*)
    - RemoveTail, XRF 496
    - SetAt, XRF 496–497
  - Phone Book sample program, XUG 127
- CObList constructor, XRF 482–483
  - CMapStringToOb::CMapStringToOb, XRF 379
  - CObject::AssertValid, XRF 466
- CObList destructor, XRF 483
- COBOL command macro, NMAKE, ET 675
- COBOL Compiler
  - NMAKE command macro, ET 675
  - NMAKE options macro, ET 676
- _CODE constant, C++, LR+ 415
- Code, inline, debugging, ET 322
  - inserting, HELPMAKE rich text format, ET 726
  - invariant, removing, ET 535–536
  - memory model, ET 489–490
  - p-code
    - native entry points, ET 520–521
    - optimizing, ET 538
    - quoting, ET 533
    - specifying, ET 521
  - searching, PWB, ET 85
  - source, displaying, ET 350, 457–460
  - tracing, p-code to native code, ET 391–392
- Code links, HELPMAKE, ET 718
- Code listings
  - DMTEST sample program, XUG 66–79
  - HELLO sample program, XUG 83, 107, 113–116
  - Phone Book sample program
    - database, XUG 141–149
    - dialog boxes, XUG 168–194
    - message handlers, XUG 243–246
- Code pointers. *See* Pointers
- Code segments
  - defining attributes
    - module-definition files, ET 618, 620
  - memory models, CL options, ET 488–490
  - naming, CL option, ET 528–530
  - naming, custom memory models, PT 76–77
  - overlaid DOS programs, ET 598, 603–605
  - packing, PT 26–27

- Code segments (*continued*)
  - pointers, PT 56
  - specifying, custom memory models, PT 77–78
- code_seg pragma
  - described, LR 210
  - new in version 7.0, LR 258
  - precompiled header compilation, effect on, PT 41
- code_seg pragma, C++, LR+ 387
- CODE statement, module-definition files, ET 609, 618, 620
- Code symbols, ET 805
- CodeView
  - access control, ET 409
  - animating, ET 432–433
  - arguments, ET 352, 363–364
  - breakpoints
    - clearing, ET 426–427
    - disabling, ET 427–428
    - editing, ET 367
    - enabling, ET 428–429
    - listing, ET 429
    - setting, ET 326, 365–366, 384–385, 429–432
  - case sensitivity, ET 371
    - expression evaluators, ET 405
    - options, ET 445–447
  - CL options, ET 553
  - command line, ET 336–343
  - commands
    - copying text, ET 353
    - described, ET 460
    - entering, ET 352
    - executing, ET 340, 433–434
  - compacting files with CVPACK, ET 743–744
  - compatibility, MOVE, ET 603
  - configuring, ET 329–330
  - contracting elements, ET 367–368, 478–479
  - CURRENT.STS, PWB, ET 138
  - CVW
    - commands, ET 382–385
    - compared to CV, ET 377–382
    - multiple applications, ET 380–382
    - multiple instances, ET 379–380
    - running, ET 378–379
    - techniques, ET 386–388
  - debugging
    - assembly language, ET 412–415
    - p-code, ET 389–393
    - PWB programs, ET 28–34

- CodeView (*continued*)
  - displays
    - black-and-white, ET 339
    - line-display mode, ET 339
    - overview, ET 345–347
    - redrawing, ET 479
    - screen exchange, ET 341, 371, 445–447, 479
    - specifying, ET 338–339
    - suppressing snow, ET 341
  - dynamic-link libraries, loading, ET 328, 342, 363–364
  - editing, ET 360
  - error messages. *See* "Error Messages," Part 2 of this book
  - execution
    - controlling, ET 386
    - speed of, ET 453
    - terminating, ET 387–388
  - expanding elements, ET 367–368, 478–479
  - expression evaluators
    - choosing, ET 403–404, 454–455
    - defined, ET 399
    - listing, ET 370
    - numbers, ET 407–408
    - operators, ET 405–407
    - string literals, ET 408
    - symbol format, ET 409
  - expressions
    - *See also* Expressions
    - addresses, ET 401–403, 420–421
    - C++, ET 409–415
    - line numbers, ET 400, 418
    - overview, ET 399
    - registers, ET 401
  - flags, changing values, ET 450–452
  - functions
    - listing, ET 435–436
    - tracing, ET 452–453
  - Help menu, ET 757
  - Help
    - *See also* Microsoft Advisor
    - getting, ET 756–765
    - structure, ET 755
  - identifying bugs, ET 325–326
  - information in object files, overriding, PT 38–39
  - inline assembly code, debugging, PT 115
  - installing, ET 327–329
  - interrupt trapping, ET 341–342

CodeView (*continued*)
- interrupting execution, ET 387
- LINK option, ET 577
- loading symbolic information, ET 342
- locating bugs, ET 326
- memory
  - comparing, ET 437–438
  - dumping, ET 438–439
  - entering data, ET 440–441
  - filling, ET 441–442
  - format, ET 356–357
  - management of, ET 336
  - moving blocks of, ET 442–443
  - searching, ET 443–444
  - viewing, ET 455–457
- menus
  - Calls menu, ET 372–373
  - Data menu, ET 364–368
  - Edit menu, ET 360
  - File menu, ET 358–360
  - Help menu, ET 374–375
  - Options menu, ET 368–372
  - Run menu, ET 362–364
  - Search menu, ET 361–362
  - Windows menu, ET 373–374
- module statement keywords, ET 621
- modules
  - configuring, ET 363–364
  - listing, ET 463
- mouse
  - disabling, ET 342
- options, ET 338–343, 445–447
  - remote debugging, ET 396
  - setting, ET 344
- p-code, PT 45
- preparing programs, ET 321–325
- printing, ET 359
- PWB menu commands, ET 74
- quitting, ET 360
- radix, ET 444–445
- registers, changing values, ET 450–452

CodeView (*continued*)
- remote debugging
  - overview, ET 393
  - requirements, ET 393–398
  - syntax, ET 396
- restarting, ET 362, 436–437
- searching, ET 361–362
- shell escape, ET 468–469
- slow motion execution, ET 363, 369
- source code, displaying, ET 457–460
- source files
  - loading, ET 359
  - opening, ET 358
- state file
  - overview, ET 344
  - toggling status, ET 343
- syntax, ET 336–343
  - CVW commands, ET 382–385
  - expressions, ET 400–403, 417–421
  - regular expressions, ET 779
- TOOLS.INI file entries, ET 330–336
- trace speed, ET 453
- variables
  - listing, ET 369
  - local, ET 354
  - program, ET 351
- viewing error messages. *See* "Error Messages," Part 2 of this book
- viewing output, ET 374
- watch expressions
  - adding, ET 364, 460
  - deleting, ET 365, 461
  - listing, ET 465–466
  - setting, ET 326–327
- windows
  - 8087 window, ET 355–356
  - Command window, ET 351–354, 417
  - Help window, ET 357–358
  - Local window, ET 354
  - Memory windows, ET 356–357
  - navigation, ET 349
  - opening, ET 373
  - overview, ET 347–349

Key

ET	Environment and Tools	PT	Programming Techniques
LIB	Run-Time Library Reference	TUT	C++ Tutorial
LR	C Language Reference	XRF	Class Libraries Reference
LR+	C++ Language Reference	XUG	Class Libraries User's Guide

CodeView (*continued*)
windows (*continued*)
program errors, XUG 11
Register window, ET 354–355
Source windows, ET 350
Watch window, ET 350–351
/CODEVIEW option, LINK, ET 577
CodeView Options command, PWB, ET 75
Collate member function, CString class, XRF 576–577
Collection classes
array classes described, XRF 27
arrays, XUG 40
described, XRF 26–27
Foundation classes cookbook, XUG 269–276
list classes described, XRF 27
maps, XUG 40
maps classes described, XRF 27
storing objects of CObject-derived classes, XRF 26
summary, Foundation classes tutorial, XUG 49
Collection objects
described, Foundation classes tutorial, XUG 40
designing, Foundation classes tutorial, XUG 37
Collections
array elements, deleting, Foundation classes cookbook, XUG 274
arrays, iteration of, Foundation classes cookbook, XUG 272
CObject class, Foundation classes cookbook, XUG 273
deriving and extending, Foundation classes cookbook, XUG 271
lists, XUG 272–273
maps, XUG 272–274
members, accessing, Foundation classes cookbook, XUG 272–276
predefined, using, Foundation classes cookbook, XUG 270
queue, creating, XUG 276
shapes, XUG 269
stacks, creating, XUG 275
templates, using, Foundation classes cookbook, XUG 271
type-safe, XUG 270
Colon (:)
appending device names, CL, ET 497–498
CodeView operator, ET 405
Delay command, CodeView, ET 423, 475
dependency, NMAKE, ET 656–657
Colon (:) (*continued*)
dot directives, NMAKE, ET 687
HELPMAKE commands, ET 722
inference rules, NMAKE, ET 681
macro substitution, NMAKE, ET 677
module statement syntax, ET 610
setting off length of bit field from declarator, LR 68
target separator, NMAKE, ET 655
Color entry, TOOLS.INI file, CodeView, ET 330–331
Color graphics adapter, ET 805
Color switch, PWB, ET 263, 271–273
Colors
attributes, selecting, PT 175–176
background
getting, _getbkcolor function, LIB 345
returning current for device context, CDC::GetBkColor, XRF 193
setting, CDC::SetBkColor, XRF 238
setting current, _setbkcolor function, LIB 652–653
brushes, creating, CBrush::CreateSolidBrush, XRF 124
called when child-system control class about to be drawn, CWnd::OnCtlColor, XRF 726–727
CGA palettes, PT 176–177
customizing
PWB, ET 124–125
EGA palettes, PT 179
filling display area with, _floodfill and _floodfill_w functions, LIB 283–284
getting current text, _gettextcolor function, LIB 378
getting current, _getcolor function, LIB 350–351
getting pixel values, _getpixel functions, LIB 374–375
graphics
controlling functions (list), PT 188–189
maximizing, PT 172
inverting in specified region, CDC::InvertRgn, XRF 214
matching current to update client area, CDC::UpdateColors, XRF 260
MCGA palettes, PT 179
Olivetti palettes, PT 177
palettes
See also Palettes
getting, _pg_getpalette function, LIB 565–567
remapping, _remapallpalette and _remappalette functions, LIB 619–623
setting RGB values and flags in logical palette, CPalette::SetPaletteEntries, XRF 506

- Colors (*continued*)
  - presentation graphics
    - color pool, PT 215–216
    - palettes, PT 214–215
  - retrieving, RGB value of specified pixels, CDC::GetPixel, XRF 202
  - returning, closest to specified logical color, CDC::GetNearestColor, XRF 202
  - setting
    - called when change made, CWnd::OnSysColorChange, XRF 788
    - CodeView, ET 370–371
    - current text, _settextcolor function, LIB 678–680
    - current, _setcolor function, LIB 658–659
    - low-level palette routines, LIB 25
    - pixel to current, _setpixel functions, LIB 676–677
  - specifying
    - PWB, ET 271–273
    - PWB switches, ET 313
  - text
    - retrieving current, CDC::GetTextColor, XRF 207
    - setting, CDC::SetTextColor, XRF 248
  - values, ET 272
  - VGA palettes, PT 177–179
- Colors command
  - CodeView, ET 368, 370–371
  - PWB, ET 75
- Colors dialog box, CodeView, ET 370–371
- Column charts
  - described, PT 203
  - sample program, PT 208, 211–212
  - styles, PT 204–205
- .COM files, ET 805
- CombineRgn member function, CRgn class, XRF 539–540
- Combining libraries, ET 704
- Combo boxes
  - called when control created, CWnd::OnMeasureItem, XRF 754, 756
  - CComboBox class described, XRF 139–140
  - comparing items in, CWnd::OnCompareItem, XRF 722, 724
- Combo boxes (*continued*)
  - control, calling owner when visual aspect or menu changes, CWnd::OnDrawItem, XRF 732–735
  - control, filling with directory listing, CWnd::DlgDirListComboBox, XRF 678–679
  - creating
    - attaching, CComboBox:Create, XRF 143–146
    - constructor, CComboBox::CComboBox, XRF 142
  - describing deleted item, DELETEITEMSTRUCT structure, XRF 80
  - destroying, XRF 729–730
  - edit control
    - deleting selection and copying to clipboard, CComboBox::Cut, XRF 146
    - deleting selection, CComboBox::Clear, XRF 143
    - getting position of current selection, CComboBox::GetEditSel, XRF 149
    - inserting clipboard data into, CComboBox::Paste, XRF 152
    - limiting text length, CComboBox::LimitText, XRF 152
    - selecting characters in, CComboBox::SetEditSel, XRF 154
  - items
    - retrieving associated application-supplied 32-bit value, CComboBox::GetItemData, XRF 150
    - setting associated with 32-bit value, CComboBox::SetItemData, XRF 155
  - list boxes
    - adding list of filenames to, CComboBox::Dir, XRF 147
    - adding string to, CComboBox::AddString, XRF 142
    - deleting string in, CComboBox::DeleteString, XRF 147
    - finding first string containing specified prefix, CComboBox::FindString, XRF 148
    - getting string from, CComboBox::GetLBText, XRF 150
    - inserting string into, CComboBox::InsertString, XRF 151
    - items in, getting number, CComboBox::GetCount, XRF 149

Combo boxes (*continued*)
  list boxes (*continued*)
    returning selected items, CComboBox::GetCurSel, XRF 149
    searching for string in, CComboBox::SelectString, XRF 153
    selecting string in, CComboBox::SetCurSel, XRF 154
    showing or hiding specified, CComboBox::ShowDropDown, XRF 155
  listing all items from, CComboBox::ResetContents, XRF 153
  PWB, ET 81–82
  retrieving current selection from list box, CWnd::DlgDirSelectComboBox, XRF 680–681
  supplying identifiers for two items in, COMPAREITEMSTRUCT structure, XRF 77–78
Comma
  argument separator, CodeView, ET 352–353
  CodeView operator, ET 405
  field separation, LIB, ET 699
  in initializer lists, LR 95
  LINK syntax, ET 565
  with context operator, CodeView, ET 421–422
Comma operator
  See also Sequential evaluation operator
  binary-operator expressions, LR+ 116–117
  overloading, LR+ 358
!command
  CodeView, ET 452
  HELPMAKE, ET 719
Command buffer, using CodeView, ET 353
Command button, PWB, ET 82
COMMAND.COM, file handles, ET 808
Command command, CodeView, ET 373–374
Command files
  defined, ET 805
  NMAKE, ET 651
Command line
  error messages. *See* "Error Messages," Part 2 of this book
  floating-point math package options, PT 132–136
  memory model options, PT 62–63, 70, 76–77
  optimization options
    aggressive, enabling, PT 20
    aliasing assumptions, PT 13–18
    calling conventions, PT 29–30
    common subexpression elimination, PT 23
Command line (*continued*)
  optimization options (*continued*)
    entry points, removing, PT 51
    entry tables, specifying, PT 51–52
    floating-point result handling, PT 23–24
    frame sorting, PT 52
    function-level linking, PT 21
    inlining, PT 13
    intrinsic function generation, PT 10–12
    loops, PT 18–19
    loops, disabling unsafe, PT 20
    maximizing efficiency, PT 24–25
    overview, PT 6
    p-code, PT 44
    processor selection, PT 24
    register allocation, PT 21–22
    size, PT 9–10, 24–25
    speed, PT 9–10, 24–25
    stack probe removal, PT 21
  p-code compiling, PT 44
  precompiled header options, PT 34–40
  QuickWin program options, PT 154–155
  running programs, QuickWin, PT 154
  wildcards, C++, LR+ 39
Command lines
  BSCMAKE, ET 735–738
  CL, ET 485–486
  CodeView, ET 336–343
  CVPACK, ET 744
  EXEHDR, ET 629–631
  EXP, ET 750
  IMPLIB, ET 746–747
  LIB, ET 698
  LINK, ET 564–572
  MOVE, ET 604–605
  NMAKE, ET 647
    command file, ET 651
    commands, ET 660
    macros, defining, ET 669–670
    suppressing, ET 688
  PWB, ET 141–142
  RM, ET 748–749
  SBRPACK, ET 740–741
  UNDEL, ET 749
Command-line arguments
  NMAKE, ET 651
  parsing, LR 32–33; LR+ 40–42
  received by main, LR 30

- Command modifiers, NMAKE, ET 661–662
- Command shell, DOS Shell command, CodeView, ET 359
- Command window, CodeView
  - command format, ET 417
  - function, ET 351–353
  - opening, ET 374
  - overview, ET 347
- Command-line processing, LR 33–34
- Commands
  - CodeView, ET 424–479
    - copying text for, ET 353
    - CVW, ET 382–385
    - Data menu, ET 364–368
    - Edit menu, ET 360–362
    - entering, ET 352
    - executing, ET 340, 433–434
    - File menu, ET 358–360
    - for Windows applications, ET 377
    - format, ET 352–353
    - Help menu, ET 374
    - Options menu, ET 368–372
    - Run menu, ET 362–363
    - Windows menu, ET 373
  - defined, ET 805
  - executing, system function, LIB 803–804
  - Help menu, PWB, ET 78
  - HELPMAKE, dot commands, ET 713–714, 722–724
  - LIB, ET 702–705
  - menu, Phone Book sample program, XUG 135
  - NMAKE
    - displaying, ET 649
    - exit codes, ET 662–664
    - inline files, ET 664–667
    - macros, ET 675
    - modifiers, ET 661–662
    - predefined inference rules, ET 684–685
    - suppressing display, ET 650
    - syntax, ET 660–661
  - optional items, LIB xv
- Commands (*continued*)
  - PWB
    - choosing, ET 78–79
    - cursor movement, ET 154–155
    - Edit menu, ET 73
    - executing, ET 78–82, 142, 170, 219
    - File menu, ET 72
    - Options menu, ET 75
    - predefined, ET 142–146
    - Project menu, ET 74
    - Run menu, ET 74, 125, 127
    - Search menu, ET 73
    - Window menu, ET 77
  - QuickHelp, ET 770
  - QuickWin, PT 148–151
- COMMDLG.DLL file
  - dialog classes, XUG 163, 167
  - PrintDlg function, XUG 212
- COMMDLG.H file
  - open dialogs, Foundation classes tutorial, XUG 153, 167
  - print dialogs, Foundation classes tutorial, XUG 153, 167
  - save dialogs, Foundation classes tutorial, XUG 153, 167
- .comment command, HELPMAKE, ET 722
- Comment command, CodeView, ET 423, 471
- Comment line, custom builds in PWB, ET 59
- comment pragma
  - described, LR 210–211
  - precompiled header compilation, effect on, PT 41
- comment pragma, C++, LR+ 387
- Comments
  - C++
    - described, LR+ 3–4
    - format, TUT 9
  - described, LR 2
  - inline assembly, PT 114–116
  - makefiles, ET 654
  - preserving, CL, ET 491
  - PWB, ET 136
  - TOOLS.INI file, ET 329–330

_commit function, LIB 158–159
Committing to disk, _dos_commit function, LIB 187–188
COMMODE.OBJ, LIB 33, 37
Common expressions, optimizing, CL option, ET 533
Common subexpression elimination, PT 8, 23
Communications services, calling BIOS, _bios_serialcom function, LIB 122–124
Compact memory model, ET 805
Compact memory models
  option, CL, PT 63
  null pointers, PT 61–62
Compacting files, CVPACK, ET 743–744
Compare member function, CString class, XRF 577
COMPAREITEMSTRUCT structure, XRF 77–78
CompareNoCase member function
  CString class, XRF 577–578
Comparing
  absolute time, CTime comparison operators, XRF 616
  characters in two buffers
    memcmp and _fmemcmp functions, LIB 500–502
    _memicmp and _fmemicmp functions, LIB 506–507
  characters of two strings
    strncmp and _fstrncmp functions, LIB 767–769
    _strnicmp and _fstrnicmp functions, LIB 772–773
  CMemoryState objects, CMemoryState::Difference, XRF 55
  items in combo boxes, CWnd::OnCompareItem, XRF 722, 724
  strings
    CString::Collate, XRF 576
    CString::Compare, XRF 577
    CString::CompareNoCase, XRF 577–578
    lowercase, _stricmp and _fstricmp functions, LIB 759–760
    null-terminated, strcmp and _fstrcmp functions, LIB 743–745
    using locale-specific information, strcoll function, LIB 746
  time, two relative values, CTimeSpan comparison operators, XRF 626
Comparison operators, CString class, XRF 596
Compatibility, LIB 75
  floating-point math operations, ET 512
  floating-point math options, PT 138–139
  operands, C++, LR+ 130–131
  with special collection classes, XRF 26
Compilation, conditional
  described, LR 202–207
  evaluating expressions, LR 206
  testing code, LR 3
Compilation, conditional, C++
  control, preprocessor directives, LR+ 379
  #if, #elif, #else and #endif directives, LR+ 379–383
  #ifdef and #ifndef directives, LR+ 383–384
Compile command, PWB, predefined macros, ET 144
Compile File command, PWB, ET 74
Compile function, PWB, ET 150, 163–164
Compiler options
  changing, PWB, ET 52–56
  debugging considerations, ET 323–324
  listing, ET 525
Compiler
  changing options, PWB, ET 52–56
  error messages. *See* "Error Messages," Part 2 of this book
  increasing capacity, CL, ET 491
  menu commands, PWB, ET 75
  optimizing, CL options, ET 531–539
  options, ET 56
  warnings. *See* "Error Messages," Part 2 of this book
Compiling
  __based keyword, LR 258
  comments, LR 2–3
  debugging considerations, ET 323
  defined, ET 805
  errors
    due to single-line comments, LR 3
    messages produced by error directives, LR 208
  files, PWB, ET 234
  for debugging, CL option, ET 553
  header file. *See* precompiled headers, using
  mixed-language programming, PT 237
  optimization. *See* Optimization
  overlays, ET 599–600
  p-code
    from PWB, PT 43–44
    from the command line, PT 44
    options, PT 50–52
  portability guidelines, PT 285
  precompiled headers, using, PT 33
  QuickWin programs
    from PWB, PT 156
    from the command line, PT 154–155

- Compiling (*continued*)
  - sample programs
    - DMTEST, XUG 65
    - HELLO, XUG 109
    - PHBOOK, XUG 243
  - speed
    - increasing using precompiled headers, PT 33
    - p-code use, effect on, PT 43
  - translation units, LR 26
  - white space ignored, LR 2
  - without linking, CL, ET 491
  - /Za option, LR 263–265
- Complement operator, C++
  - unary-operator expressions, LR+ 94
- Complex declarators, LR 88–91
- Composed classes, construction, LR+ 305
- Composition
  - compared with inheritance, TUT 178
  - relationship between classes, TUT 176
- Compound assignment operators, LR 139
- Compound statements
  - *See also* Blocks
  - C++, described, LR+ 137
  - defined, LR 168
  - described, LR 153
  - overview, LR 151
  - repeating, LR 156
  - type of block, LR 28
- Compressing
  - Help database, ET 711–712
  - keywords, HELPMAKE option, ET 712–713
- Computing
  - Bessel functions, LIB 103–105
  - quotients and remainders
    - from long integers, ldiv and ldiv_t functions, LIB 449–450
    - of two integer values, div function, LIB 181–182
  - real numbers from mantissa and exponent, ldexp and _ldexpl functions, LIB 447–448
  - string's width, height, CDC::GetTabbedTextExtent, XRF 204–205
  - text's line width, height, CDC::GetTextExtent, XRF 207
- CON
  - CL options, appending to, ET 497–498
- Concatenating
  - Help files, ET 772
  - string literals, LR 21, 195; LR+ 22
- Concatenation operators
  - CString::operator +, XRF 594
  - CString::operator +=, XRF 595
  - Foundation classes cookbook, XUG 257
- Concrete classes, TUT 128
- Conditional branching.
  - *See also* switch statements; if statements
- Conditional breakpoints, ET 805
- Conditional compilation
  - described, LR 202–207
  - evaluating expressions, LR 206
  - testing code, LR 3
- Conditional compilation, C++
  - control, preprocessor directives, LR+ 379
  - #if, #elif, #else and #endif directives, LR+ 379–383
  - #ifdef and #ifndef directives, LR+ 383–384
- Conditional operator (? :)
  - ANSI compatibility, LR 264
  - CodeView, ET 405
  - described, LR 136–138
  - uses, LR 137
- Conditional operator, C++, LR+ 117–118
- CONFIG.SYS
  - *See also Getting Started*
  - editing, PWB, ET 66
  - memory management, CodeView, ET 336
  - PWB configuration, ET 137
- Configuring
  - *See also Getting Started*
  - CodeView
    - modules, ET 363–364
    - TOOLS.INI, ET 329–330
- Consistency
  - floating-point math operations, PT 23–24, 138
  - precompiled header rules, PT 39–41
- Consistency checks
  - heaps, _heapchk functions, LIB 410–412
  - LIB, ET 700

Console
  checking for keyboard input, _kbhit function, LIB 444
  getting characters from
    _cgets function, LIB 136–137
    _getch and _getche functions, LIB 348–349
  I/O routines, LIB 43–44
  putting strings to, _cputs function, LIB 167
  reading data from, _cscanf function, LIB 171–172
  writing characters to, _putch function, LIB 595–596
_CONST constant, C++, LR+ 415
const keyword
  listed, LR 53
  modifying typedef, LR 102
  nonterminal, LR 43
  objects, LR 52
  repeating, LR 264
  restrictions, LR 107
  using, LR 53
const keyword, C++
  member functions, TUT 57
  objects, TUT 56–57
  parameters, TUT 18
  pointers, LR+ 188–190; TUT 18
  this pointer modification, LR+ 246
  variables, TUT 17
Constant expressions
  defined, ET 805
  described, LR 108–109
  restricted, LR 204
  with #if directives, LR 204
Constant expressions, C++
  described, LR+ 118–119
  integral, conversion to null pointer, LR+ 75
Constants
  characteristics, LR 9
  defined, ET 805
  defining, CL, ET 492–493
  described, LR 9–20
  floating-point, LR 10
  graphics (table), PT 169–170
  inline assembly, PT 115–116
  integer types, LR 14
  propagation, PT 9
  symbolic
    graphics, PT 180
    inline assembly, PT 115–116
  types, LR 106
  windows (list), PT 158
Constants, C++
  character
    described, LR+ 16–19
  described, LR+ 14
  enumerators. *See* Enumerators, C++
  floating limits, LR+ 63–64
  floating-point
    described, LR+ 19–20
  integer
    described, LR+ 14–16
  integral limits, LR+ 62
  string literals
    concatenation, LR+ 22
    defined, LR+ 20–23
  types, LR+ 54
Construction of objects, two-phase, XRF 15
Construction order, C++, LR+ 305
Constructors
  adding, Phone Book sample program, XUG 202
  array allocation, TUT 95
  array, described, LR+ 305
  base class initialization, TUT 116
  called by new operator, TUT 64, 95, 102, 105
  calling member functions, TUT 54
  CArchive, XRF 95–96
  CArchiveException, XRF 105
  CBitmap, XRF 109
  CBrush, XRF 119–120
  CButton, XRF 128
  CClientDC, XRF 138
  CComboBox, XRF 142
  CDataBase, Foundation classes tutorial, XUG 135
  CDC, XRF 169
  CDialog, XRF 264
  CDumpContext, XRF 275
  CEdit, XRF 285
  CFile, XRF 306–308
  CFileException, XRF 324
  CFont, XRF 330
  CFrameWnd, XRF 338
  CGdiObject, XRF 344
  CListBox, XRF 355
  CMainWindow
    Foundation classes tutorial, XUG 90–91
    ShowWindow function, XUG 88
    UpdateWindow function, XUG 88
    windows, creating, XUG 89–93
  CMapStringToOb, XRF 379
  CMDIChildWnd, XRF 397

- Constructors (*continued*)
  - CMDIFrameWnd, XRF 403
  - CMemFile, XRF 412
  - CMemoryException, XRF 413
  - CMenu, XRF 420
  - CMetaFileDC, XRF 440
  - CModalDialog, XRF 446
  - CObArray, XRF 454
  - CObject, XRF 467
  - CObList, XRF 482–483
  - conversion, LR+ 313–315; TUT 148–150
  - copy, TUT 78–81, Foundation classes tutorial, XUG 29–30
  - CPaintDC, XRF 499
  - CPalette, XRF 503
  - CPen, XRF 509–510
  - CPerson, Foundation classes tutorial, XUG 28
  - CPersonList, Foundation classes tutorial, XUG 41
  - CPoint, XRF 513
  - CRect, XRF 524
  - CResourceException, XRF 536
  - CRgn, XRF 545
  - CScrollBar, XRF 553
  - CSize, XRF 559
  - CStatic, XRF 563
  - CStdioFile, XRF 568–569
  - CString, XRF 578–579
  - CTime, XRF 608–609
  - CTimeSpan, XRF 620–621
  - CWinApp, XRF 631
  - CWnd, XRF 673
  - declaring, LR+ 302–304
  - default, TUT 54, 59, 94, 116
  - defining, Foundation classes cookbook, XUG 281
  - derived window classes, Foundation classes cookbook, XUG 312
  - described, LR+ 300–301
  - dialog resource, Foundation classes tutorial, XUG 166
  - exceptions, Foundation classes cookbook, XUG 303
  - filebuf, XRF 833
  - Foundation graphics, Foundation classes cookbook, XUG 346
- Constructors (*continued*)
  - frame allocation, Foundation classes cookbook, XUG 252fstream, XRF 839–841
  - global objects, TUT 50
  - ifstream, XRF 847–848
  - in the frame, described, XUG 29, 31, 41
  - in the heap, described, XUG 29, 31, 41
  - initializers, LR+ 284
  - ios, XRF 860
  - Iostream_init, XRF 874
  - iostream, XRF 873
  - istream, XRF 881
  - istream_withassign, XRF 888
  - istrstream, XRF 891
  - member initialization, TUT 58–60
  - ofstream, XRF 895–897
  - ostream, XRF 903
  - ostream_withassign, XRF 909
  - ostrstream, XRF 912
  - overloading, TUT 48, 53
  - overview, TUT 43, 47
  - parameters, with, Foundation classes tutorial, XUG 28
  - parameters, without, Foundation classes tutorial, XUG 29
  - serialization, used for, Foundation classes tutorial, XUG 30
  - static objects, TUT 50
  - stdiobuf, XRF 916
  - stdiostream, XRF 918
  - streambuf, XRF 936
  - strstream, XRF 941
  - strstreambuf, XRF 945–946
  - using C++ expressions, ET 410–411
- Contents command
  - CodeView, ET 374–375
  - PWB, ET 78, 757
    - predefined macros, ET 146
- .context command, HELPMAKE, ET 716–717, 720, 722, 726
- Context operator ({ })
  - CodeView, ET 405
  - function, ET 406–407, 421–422

Context prefixes, HELPMAKE, ET 729
contextstring command, HELPMAKE, ET 718–719
continue statement
  C++, jump statements, LR+ 147–148
  described, LR 154
  preferred over goto statements, LR 158
Contracting, elements in CodeView, ET 367–368, 478–479
Control characters, specifying, HELPMAKE, ET 712
Control classes
  dialog boxes, XUG in, Foundation classes cookbook, XUG 340
  message handler functions, using, Foundation classes cookbook, XUG 338
  overriding, Foundation classes cookbook, XUG 338
  standard, deriving from, Foundation classes cookbook, XUG 337–339
  using, Foundation classes cookbook, XUG 335
  values, setting, Foundation classes cookbook, XUG 339–340
Control library, selecting, CL options, ET 527
Control messages, XRF 14
Control menu
  allowing application access to, CWnd::GetSystemMenu, XRF 699
  called when user selects command from, CWnd::OnSysCommand, XRF 788–790
Control values, setting, Foundation classes cookbook, XUG 339–340
Control Windows classes
  described, XRF 16
  (list), XRF 6
_control87 function, LIB 160–162
Controlling
  frame sorting, p-code, PT 52
  inline expansion, LR 260
  optimization
    from PWB, PT 5–6
    from the command line, PT 6
    from the linker, PT 25–28
    using pragmas, PT 6–7
  QuickWin menus, PT 152
  quoting, p-code, PT 52
  stream buffering, buffer size, setvbuf function, LIB 688–689
  word wrap, _wrapon function, LIB 885–886
Conventional memory
  browser database, ET 733–734, 737
  defined, ET 805
Conventions, CL options
  calling, ET 516–518, 520
  naming, ET 518, 529, 552
Conversion constructors, C++
  described, LR+ 313–315
Conversion functions
  using C++ expressions, ET 410–411
Conversion functions, C++
  declaring, LR+ 317–318
  described, LR+ 315–317
Conversions
  ambiguities, LR+ 274; TUT 152–155
  argument matching, overloaded functions, LR+ 346–350
  arithmetic, LR 115–116; LR+ 69–71
  assignment, LR 126, 141–150
  base to derived class, TUT 118
  C++, described, LR+ 65, 312–313
  constructors, TUT 148–150
  derived to base class, TUT 116
  enumeration types, LR 147
  enumerators, LR+ 177–178
  explicit type conversions
    described, LR+ 119–124
    operator, LR+ 119–121
  floating conversions, LR+ 68–69
  floating to integral, LR+ 69
  floating-point types, LR 145–146
  function call, LR 149–150
  integral conversions
    signed to unsigned, LR+ 67
    standard, LR+ 68
    unsigned to signed, LR+ 67–68
  integral promotions, LR+ 66–67
  integral to floating, LR+ 69
  integral types, LR 141–144
  operators, TUT 150–151
  pointer
    See also Pointer conversions, C++
    described, LR+ 71–76
  pointer types, LR 127, 146–147
  references, LR+ 75
  rules, LR 15, 115, 141–150
  signed integral types, LR 126, 141–144
  type cast, LR 147–149
  types, LR 126
  unsigned integral types, LR 143–144

- Converting
  - between IEEE and MS double values, _dieeetomsbin and _dmsbintoieee functions, LIB 175
  - characters, to ASCII, lower- or uppercase, __toascii, tolower, toupper functions, LIB 817–819
  - characters from ANSI to OEM character set, CString::AnsiToOem, XRF 576
  - characters from OEM to ANSI character set, CString::OemToAnsi, XRF 588
  - client coordinates to screen coordinates, CWnd::ClientToScreen, XRF 663
  - CString object to lowercase, CString::MakeLower, XRF 586
  - CString object to uppercase, CString::MakeUpper, XRF 587
  - data segments, LR 259
  - dialog units of rectangle to screen units, CDialog::MapDialogRect, XRF 270
  - DOS applications to Windows applications. *See* QuickWin
  - double numbers to strings, _ecvt function, LIB 239–240
  - error codes, run-time library to CFileException values, CFileException::ErrnoToException, XRF 324
  - expression-list, LR 185
  - floating-point
    - numbers between IEEE and Microsoft binary format, _fieeetomsbin and _fmsbintoieee functions, LIB 279
    - numbers to strings, _fcvt function, LIB 262–263
    - values, LR 145–146
    - values to strings, _gcvt function, LIB 340–341
  - integers
    - long to strings, _ltoa function, LIB 474–475
    - to strings, _itoa function, LIB 442–443
    - unsigned long to strings, _ultoa function, LIB 823–824
  - logical to device points, CDC::LPtoDP, XRF 215
  - multibyte to wide characters, mbstowcs and _fmbstowcs functions, LIB 489–490

- Converting (*continued*)
  - pointer size, PT 68–69
  - pointers, LR 86
  - points, device into logical, CDC::DPtoLP, XRF 175
  - rectangles between CRect and LPRECT, CRect::operator, XRF 531
  - single multibyte to wide characters, mbtowc and _fmbtowc functions, LIB 491–493
  - strings
    - to double-precision or long-integer values, strtod, strtol, _strtold and strtoul functions, LIB 790–793
    - to lowercase, _strlwr and _fstrlwr functions, LIB 763–764
    - to uppercase, _strupr and _fstrupr functions, LIB 797–798
    - to values, atof, atoi, atol and _atold functions, LIB 98–100
  - time
    - local to calendar, mktime function, LIB 521–522
    - structures to character strings, asctime function, LIB 88–89
    - to character strings, ctime function, LIB 173–174
    - values to structures, gmtime function, LIB 394–395
    - values with zone correction, localtime function, LIB 458–459
  - wide to multibyte characters
    - wcstombs and _fwcstombs functions, LIB 867–868
    - wctomb and _fwctomb functions, LIB 869–870
- Coordinate systems
  - bounding rectangles, PT 185
  - described, PT 180
  - physical coordinates
    - described, PT 180–182
    - using, functions listed, PT 181
  - pixel cursors, PT 186
  - screen locations, PT 185
  - viewport coordinates, described, PT 182–183
  - window coordinates, described, PT 184–185

Coordinates
carets, retrieving, CWnd::GetCaretPos, XRF 686
translating to view, _getviewcoord functions, LIB 386–387
translating views to window coordinates, _getwindowcoord function, LIB 391
Coprocessor, floating-point math. *See* Math coprocessor floating-point math package
Coprocessors
defined, ET 803
displaying registers, CodeView, ET 355–356
floating-point math, ET 509–511, 513
Copy command
CodeView, ET 360
LIB, ET 705
PWB, ET 73, predefined macros, ET 143
QuickWin, PT 149
COPY command, MS-DOS
concatenating Help databases, ET 710–711
concatenating Help files, ET 772
Copy constructors
assignment operator, TUT 79
default, TUT 78
defined, Foundation classes tutorial, XUG 29–30
overview, TUT 78–81
Copy function, PWB, ET 151, 164–165
Copy member function
CComboBox class, XRF 143
CEdit class, XRF 286
Copy Tabs command, QuickWin, PT 149
Copying, XUG 30
allowing application access to Control menu for, CWnd::GetSystemMenu, XRF 699
bitmaps, to current device context, CDC::BitBlt, XRF 167–169
characters
between buffers, memcpy and _fmemcpy functions, LIB 503–505
from buffers, _memccpy and _fmemccpy functions, LIB 496–497
CTimeSpan object, CTimeSpan::operator=, XRF 625
CWnd's caption title into specified buffer, CWnd::GetWindowText, XRF 704
dates to buffers, _strdate function, LIB 751–752
dimensions of bounding rectangle of CWnd object, CWnd::GetWindowRect, XRF 703–704
edit control selection to clipboard, CEdit::Copy, XRF 286
files, PWB, ET 95

Copying (*continued*)
fonts, current, typeface name into buffer, CDC::GetTextFace, XRF 208
menu item label to buffer, CMenu::GetMenuString, XRF 427
object modules, ET 705
objects, LR+ 333–337
preprocessor output, CL, ET 493–494
rectangles
scrRect to CRect, CRect::operator=, XRF 531
to CRect, CRect::CopyRect, XRF 523
regions into CRgn object, CRgn::CopyRgn, XRF 540
scroll bar position range, CWnd::GetScrollRange, XRF 697
scroll-bar position to specified location, CScrollBar::GetScrollRange, XRF 556
strings, strcpy _fstrcpy functions, LIB 747–748
text
CodeView commands, ET 353
Microsoft Advisor, ET 761
QuickHelp, ET 771
time source into CTime object, CTime::operator=, XRF 615
time to buffers, _strtime function, LIB 788–789
to clipboard, combo box edit control selection, CComboBox::Copy, XRF 143
Copying text, QuickWin, PT 148–149
CopyRect member function, CRect class, XRF 523
CopyRgn member function, CRgn class, XRF 540
Copyright message, suppressing. *See* /NOLOGO option
cos function, LIB 163–164; PT 11
cosh function, LIB 163–164; PT 11
_coshl function, LIB 163–164; PT 11
Cosines, calculating, cos functions, LIB 163–164; PT 11
_cosl function, LIB 163–164; PT 11
COUNT
sample program, PWB, ET 41–42, 44, 46–47, 49–50, 52–56, 58–63, 97–103
Counting
bytes stored in stream buffers, ostrstream::pcount, XRF 913
elements in lists, CObList::GetCount, XRF 485
items in list box, CListBox::GetCount, XRF 361
number of elements in maps, CMapStringToOb::GetCount, XRF 380
cout, TUT 5–6, predefined stream object, XRF 900
/CP option, LINK, ET 577–578

CPaintDC class
  described, XRF 498
  HELLO sample program, XUG 83
  member functions
    CPaintDC, XRF 499
    ~CPaintDC, XRF 499
    m_hWnd, XRF 500
    m_ps, XRF 500
CPaintDC constructor, XRF 499
CPaintDC destructor, XRF 499
CPaintDC objects
  creating, CPaintDC::CPaintDC, XRF 499
  destroying, CPaintDC::~CPaintDC, XRF 499
  painting client area, CPaintDC::m_ps, XRF 500
CPalette class
  described, XRF 501–502
  member functions
    AnimatePalette, XRF 503
    CPalette, XRF 503
    CreatePalette, XRF 504
    FromHandle, XRF 504
    GetNearestPaletteIndex, XRF 505
    GetPaletteEntries, XRF 349, 505
    ResizePalette, XRF 506
    SetPaletteEntries, XRF 506–507
CPalette constructor, XRF 503
CPalette objects
  creating, initializing, CPalette::CreatePalette, XRF 504
  resizing logical palette attached to, CPalette::ResizePalette, XRF 506
  returning pointer to, CPalette::FromHandle, XRF 504
/CPARM option, LINK, ET 577–578
/CPARMAXALLOC option, LINK, ET 577–578
CPen class
  described, XRF 508
  member functions
    CPen, XRF 509–510
    CreatePen, XRF 510
    CreatePenIndirect, XRF 510–511
    FromHandle, XRF 511
CPen constructor, XRF 509–510
CPen objects
  creating
    constructor, CPen::CPen, XRF 509
    initializing, CPen::CreatePen, XRF 510
  returning pointer to, CPen::FromHandle, XRF 511
CPerson class
  constructors, Foundation classes tutorial, XUG 27–28
  copying, described, XUG 30
  DMTEST sample program
    declaring, XUG 22
    Foundation classes tutorial, XUG 22, 28
  member functions
    AssertValid, XUG 27
    Dump, XUG 27
    Serialize, XUG 27
  overloaded assignment operator
    defined, XUG 27
    Foundation classes tutorial, XUG 129
  serialization, Foundation classes tutorial, XUG 33
CPerson object
  constructors
    in the frame, XUG 29
    in the heap, XUG 29
  described, Foundation classes tutorial, XUG 28
  Dump, using, Foundation classes tutorial, XUG 35
  serialization, overview, XUG 34
CPersonList class, Foundation classes tutorial, XUG 40
CPersonList object
  constructing, Foundation classes tutorial, XUG 41
  designing, Foundation classes tutorial, XUG 36
  Foundation classes tutorial, XUG 139
  Phone Book sample program, XUG 134
  searching in, Foundation classes tutorial, XUG 45–46
__cplusplus macro, C++, LR+ 374
CPoint class
  adding separate values to x and y members, CPoint::Offset, XRF 513
  described, XRF 20, 512
  member functions
    CPoint, XRF 513
    Offset, XRF 514
    operator, XRF 515–516

CPoint constructor, XRF 513
CPoint objects, creating, CPoint::CPoint, XRF 513
CPP command macro, NMAKE, ET 676
.CPP files, ET 805
CPPFLAGS options macro, NMAKE, ET 676
_cprintf function, LIB 165–166
CPtrArray class, described, XRF 517–518
CPtrList class, described, XRF 519–520
CPU, ET 805
_cpumode variable, LIB 65; LR 259
_cputs function, LIB 167
CRC command macro, NMAKE, ET 676
_creat function, LIB 168–170
Create member function
  CButton class, XRF 128–131
  CComboBox class, XRF 143–146
  CDialog class, XRF 264–265
  CEdit class, XRF 286–289
  CFrameWnd class, XRF 338–339
  CListBox class, XRF 356–359
  CMDIChildWnd class, XRF 397–398
  CMDIFrameWnd class, XRF 403–404
  CMetaFileDC class, XRF 441
  CScrollBar class, XRF 553–555
  CStatic class, XRF 563–566
  CWnd class, XRF 664–665
  Foundation classes cookbook, XUG 312, 336–337
  windows, creating, XUG 92–93
CreateBitmap member function
  CBitmap class, XRF 109–110
  CBrush::CreatePatternBrush, XRF 123
  CWnd::CreateCaret, XRF 665
CreateBitmapIndirect member function
  CBitmap class, XRF 110
  CBrush::CreatePatternBrush, XRF 123
CreateBrushIndirect member function
  CBrush class, XRF 120–121
CreateCaret member function
  CWnd class, XRF 665
CreateClient member function
  CMDIFrameWnd class, XRF 404
CreateCompatibleBitmap member function
  CBitmap class, XRF 111
  CBrush::CreatePatternBrush, XRF 123
CreateCompatibleDC member function, CDC class, XRF 171–172
CreateDC member function, CDC class, XRF 172–173
CreateDIBPatternBrush member function, CBrush class, XRF 121–122
CreateDiscardableBitmap member function, CBitmap class, XRF 112
CreateEllipticRgn member function, CRgn class, XRF 541
CreateEllipticRgnIndirect member function, CRgn class, XRF 541–542
CreateEx member function, CWnd class, XRF 666–670
CreateFont data member, CFont class, XRF 330–333
CreateFontIndirect data member, CFont class, XRF 334
CreateGrayCaret member function, CWnd class, XRF 671
CreateHatchBrush member function, CBrush class, XRF 122–123
CreateIC member function, CDC class, XRF 173–174
CreateIndirect member function, CDialog class, XRF 266–267
CreateMenu member function, CMenu class, XRF 421
CreatePalette member function, CPalette class, XRF 504
CreatePatternBrush member function, CBrush class, XRF 123–124
CreatePen member function, CPen class, XRF 510
CreatePenIndirect member function, CPen class, XRF 510–511
CreatePolygonRgn member function, CRgn class, XRF 542
CreatePolyPolygonRgn member function, CRgn class, XRF 543
CreatePopupMenu member function, CMenu class, XRF 421–422
CreateRectRgn member function, CRgn class, XRF 544
CreateRectRgnIndirect member function, CRgn class, XRF 544
CreateRoundRectRgn member function, CRgn class, XRF 545
CreateSolidBrush member function, CBrush class, XRF 124
CreateSolidCaret member function, CWnd class, XRF 672
CreateStockObject member function, CGdiObject class, XRF 345–346
CREATESTRUCT structure, XRF 78–79
  CMDIFrameWnd::CreateClient, XRF 404
  CWnd::OnCreate, XRF 724–725
  CWnd::OnNcCreate, XRF 762

Creating
  backup files, ET 747–750
  bitmaps
    device-compatible,
      CBitmap::CreateCompatibleBitmap, XRF 111
    device-dependent memory,
      CBitmap::CreateCBitmap, XRF 109–110
    discardable, CBitmap::CreateDiscardableBitmap,
      XRF 112
    specified structure,
      CBitmap::CreateBitmapIndirect, XRF 110
  browser database, PWB, ET 97–98
  brushes
    uninitialized object, CBrush::CBrush,
      XRF 119–120
    with bitmap-specified pattern,
      CBrush::CreatePatternBrush, XRF 123
    with DIB-specified pattern,
      CBrush::CreateDIBPatternBrush,
        XRF 121–122
    with hatch style, CBrush::CreateHatchBrush,
      XRF 122–124
    with specified structure,
      CBrush::CreateBrushIndirect, XRF 120–121
  call tree, PWB, ET 99–101
  CArchive objects, CArchive::CArchive, XRF 95
  CArchiveException objects,
      CArchiveException::CArchiveException,
        XRF 105
  carets
    grey, CWnd::CreateGrayCaret, XRF 671
    new shape, CWnd::CreateCaret, XRF 665
    solid, CWnd::CreateSolidCaret, XRF 672
  CBitmap objects, CBitmap::CBitmap, XRF 109
  CBrush objects, XRF 119–120
  CButton objects
    constructor, CButton::CButton, XRF 128
    control, CButton::Create, XRF 128–129, 131
  CClientDC objects, CClientDC::CClientCD,
      XRF 138
  CComboBox objects, CComboBox::CComboBox,
    XRF 142

Creating (*continued*)
  CDC objects, CDC::CDC, XRF 169
  CDumpContext objects,
      CDumpContext::CDumpContext, XRF 275
  CEdit objects
    CEdit::CEdit, XRF 285
    CEdit::Create, XRF 286, 289
    CFile::CFile, XRF 306
    opening file, CFile::Open, XRF 313
  CFileException objects,
      CFileException::CFileException, XRF 324
  CFrameWnd objects, CFrameWnd::Create,
      XRF 338–339
  CGdiObject objects, CGdiObject::CGdiObject,
      XRF 344
  charts and graphs, presentation graphics,
      PT 205–206
  child windows
    constructor, CMDIChildWnd::CMDIChildWnd,
      XRF 397
    QuickWin, PT 151, 157–158
  CListBox objects
    CListBox::CListBox, XRF 355
    specifying style, CListBox::Create, XRF 356–357,
      359
  CMDIChildWnd objects, creating and attaching,
      CMDIChildWnd::Create, XRF 397
  CMDIFrameWnd objects,
      CMDIFrameWnd::CMDIFrameWnd, XRF 403
  CMemoryException objects,
      CMemoryException::CMemoryException,
        XRF 413
  CMemoryState objects,
      CMemoryState::CMemoryState, XRF 54
  CMenu objects, constructor, CMenu::CMenu,
      XRF 420
  CMetaFileDC objects
    constructor, CMetaFileDC::CMetaFileDC,
      XRF 440
    device context, CMetaFileDC::Create, XRF 441

- Creating (*continued*)
  - CModalDialog objects
    - constructor, CModalDialog::CModalDialog, XRF 446
    - indirectly, CModalDialog::CreateIndirect, XRF 446–447
  - CNotSupportedException objects, XRF 449
  - CObject objects, CObject::CObject, XRF 467
  - CObject pointer arrays, CObArray::CObArray, XRF 454
  - CObject pointer lists, CObList::CObList, XRF 482–483
  - combo boxes, CComboBox::Create, XRF 146
  - CPaintDC objects, CPaintDC::CPaintDC, XRF 499
  - CPalette objects, CPalette::CreatePalette, XRF 504
  - CPen objects
    - constructor, CPen::CPen, XRF 509
    - initializing, CPen::CreatePen, XRF 510
  - CPoint objects, CPoint::CPoint, XRF 513
  - CRect objects, CRect::CRect, XRF 524
  - CResourceException objects, CResourceException::CResourceException, XRF 536
  - CRgn objects constructor, CRgn::CRgn, XRF 545
  - CSize objects, CSize::CSize, XRF 559
  - CStatic objects
    - attaching, CStatic::Create, XRF 563–564
    - constructor, CStatic::CStatic, XRF 563
  - CStdioFile objects, CStdioFile::CStdioFile, XRF 568
  - CString objects, CString::CString, XRF 579
  - CString-to-CObject map objects, CMapStringToOb::CMapStringToOb, XRF 379
  - CTime objects, CTime::CTime, XRF 608–609
  - CTimeSpan object, CTimeSpan::CTimeSpan, XRF 620–621
  - CWinApp objects, constructor, CWinApp::CWinApp, XRF 631
  - CWindowDC objects, CWindowDC::CWindowDC, XRF 642
  - CWnd objects called when, CWnd::OnCreate, XRF 724–726
  - databases, Foundation classes tutorial, XUG 139
  - device contexts, CDC::CreateDC, XRF 172–173
  - dialog box objects, CDialog::Create, XRF 264–265
  - dialog boxes
    - modeless from template, CDialog::CreateIndirect, XRF 266–267
    - modeless, CDialog::CDialog, XRF 264
  - directories, _mkdir function, LIB 516–517
- Creating (*continued*)
  - elliptical regions
    - CRgn::CreateEllipticRgn, XRF 541
    - CRgn::CreateEllipticRgnIndirect, XRF 541
  - environment variables, _putenv function, LIB 597–599
  - executable files, CL, ET 486–487
  - filebuf objects, filebuf::filebuf, XRF 833
  - file handles, _dup and _dup2 functions, LIB 236–238
  - filenames
    - temporary, _tempnam and tmpnam functions, LIB 809–811
    - unique, _mktemp function, LIB 518–520
  - files
    - _creat function, LIB 168–170
    - _dos_creat functions, LIB 189–190
    - temporary, tmpfile function, LIB 815–816
  - fonts
    - constructor, CFont::CFont, XRF 330
    - initializing with given structure, CFont::CreatFontIndirect, XRF 334
    - initializing with specified characteristics, CFont::CreateFont, XRF 330–333
  - frame windows for CMDIFrameWnd object, CMDIFrameWnd::Create, XRF 403
  - fstream objects, fstream::fstream, XRF 839, 841
  - graphics objects, Foundation classes cookbook, XUG 347
  - graphics output, LIB 26–27
  - identifiers, LR 5
  - ifstream objects, ifstream::ifstream, XRF 847–848
  - import libraries, IMPLIB, ET 745–746
  - inline files, NAME, ET 665
  - iostream objects, iostream::iostream, XRF 873
  - Iostream_init objects, Iostream_init::Iostream_init, XRF 874
  - istream objects, istream::istream, XRF 881
  - istream_withassign objects, istream_withassign::istream_withassign, XRF 888
  - istrstream objects, istrstream::istrstream, XRF 891
  - library files, ET 700, 703
  - long string literals, LR 19
  - macros
    - __asm blocks, PT 123–124
  - map files
    - CL, ET 505–507
    - LINK, ET 582–583
  - MDI client window, CMDIFrameWnd::CreateClient, XRF 404

Creating (*continued*)
  menus
    empty, CMenu::CreateMenu, XRF 421
    pop-up, CMenu::CreatePopupMenu, XRF 421
  module-definition files, ET 600–601
  new child process, _spawn functions, LIB 717–722
  ofstream objects, ofstream::ofstream, XRF 896–897
  ostream objects, ostream::ostream, XRF 903
  ostream_withassign objects, ostream_withassign::ostream_withassign, XRF 909
  ostrstream objects, ostrstream::ostrstream, XRF 912
  overlaid programs
    LINK, ET 598–601
    module-definition files, ET 619–620
    MOVE, ET 598–601
  packaged functions
    CL options, ET 524
  path names, _makepath function, LIB 476–478
  pens with specified structure, CPen::CreatePenIndirect, XRF 510
  precompiled headers, LR 261; PT 34–36
  preprocessor-output files, ET 540
  projects, PWB, ET 42
  pseudofiles, in PWB, ET 187–188, 245
  queue collections, Foundation classes cookbook, XUG 276
  QuickWin programs
    enhanced, PT 146–147, 157–165
    simple, PT 146
  rectangles, NULL, CRect::SetRectEmpty, XRF 529
  regions
    by combination, CRgn::CombineRgn, XRF 539–540
    polygonal, CRgn::CreatePolygonRgn, XRF 542
    rectangular, CRgn::CreateRectRgn, XRF 544
    rectangular, indirect, CRgn::CreateRectRgnIndirect, XRF 545
    series of polygonal, CRgn::CreatePolyPolygonRgn, XRF 543
  scroll bars
    constructor, CScrollBar::CScrollBar, XRF 553
    initializing, CScrollBar::Create, XRF 553–555
  segmented files, LINK, ET 564

Creating (*continued*)
  stack collections, Foundation classes cookbook, XUG 275
  stdiobuf objects, stdiobuf::stdiobuf, XRF 916
  stdiostream objects, stdiostream::stdiostream, XRF 918
  streambuf objects, streambuf::streambuf, XRF 936
  strstream objects, strstream::strstream, XRF 941
  strstreambuf objects, strstreambuf::strstreambuf, XRF 945
  text windows, _settextwindow function, LIB 687
  types, LR 102
  viewports, _setviewport function, LIB 699–700
  Window Edit control, CEdit class, XRF 282
  Windows child windows
    attaching to CWnd object, CWnd::Create, XRF 664–665
    constructor, CWnd::CWnd, XRF 673
  windows with extended style, CWnd::CreateEx, XRF 666–667, 670
Creation message handlers
  adding, Foundation classes tutorial, XUG 224–226
  Phone Book sample program, XUG 199
CRect class
  creating NULL rectangle, CRect::SetRectEmpty, XRF 529
  described, XRF 20, 521–522
  dimensions, setting, CRect::SetRect, XRF 528
  HELLO sample program, XUG 83
  member functions
    BottomRight, XRF 523
    CopyRect, XRF 523
    CRect, XRF 524
    EqualRect, XRF 525
    Height, XRF 525
    InflateRect, XRF 525–526
    IntersectRect, XRF 526
    IsRectEmpty, XRF 527
    IsRectNull, XRF 527
    OffsetRect, XRF 527–528
    PtInRect, XRF 528
    SetRect, XRF 528–529
    SetRectEmpty, XRF 529

- CRect class (*continued*)
  - member functions (*continued*)
    - Size, XRF 529
    - TopLeft, XRF 529
    - UnionRect, XRF 530
    - Width, XRF 530
  - operators, XRF 531–535
- CRect constructor, XRF 524
- CRect objects, creating, CRect::CRect, XRF 524
- CResourceException class
  - described, XRF 536
  - member functions, CResourceException, XRF 536
- CResourceException constructor, XRF 536
- CResourceException exception handler, Foundation classes cookbook, XUG 298
- CResourceException objects, creating, CResourceException::CResourceException, XRF 536
- CRgn class
  - described, XRF 537–538
  - member functions
    - CombineRgn, XRF 539–540
    - CopyRgn, XRF 540
    - CreateEllipticRgn, XRF 541
    - CreateEllipticRgnIndirect, XRF 541–542
    - CreatePolygonRgn, XRF 542
    - CreatePolyPolygonRgn, XRF 543
    - CreateRectRgn, XRF 544
    - CreateRectRgnIndirect, XRF 544
    - CreateRoundRectRgn, XRF 545
    - CRgn, XRF 545
    - EqualRgn, XRF 546
    - FromHandle, XRF 546
    - GetRgnBox, XRF 547
    - OffsetRgn, XRF 547–548
    - PtInRegion, XRF 548
    - RectInRegion, XRF 549
    - SetRectRgn, XRF 549–550
- CRgn constructor, XRF 545
- CRgn objects
  - checking
    - equivalent, CRgn::EqualRgn, XRF 546
    - if coordinates are within, CRgn::PtlnRegion, XRF 548
    - if specified rectangle is within, CRgn::RectlnRegion, XRF 549
  - copying region into, CRgn::CopyRgn, XRF 540
- CRgn objects (*continued*)
  - creating
    - by combination, CRgn::CombineRgn, XRF 539–540
    - constructor, CRgn::CRgn, XRF 545
  - handles, CRgn::FromHandle, XRF 546
  - moving stored region, CRgn::OffsetRgn, XRF 547
  - retrieving bounding rectangle coordinates, CRgn::GetRgnBox, XRF 547
- Cross-reference listing, LIB, ET 705–706
- CS command, CodeView, ET 422
- CS register, CodeView syntax, ET 419, 450
- CS:IP
  - defined, ET 805
  - saving, CodeView, ET 344
- C_Softcr switch, PWB, ET 312
- C_suffixes switch, PWB, ET 312–313
- _cscanf function, LIB 171–172
- CScrollBar class
  - described, XRF 551–552
  - Foundation classes cookbook, XUG 335
  - member functions
    - Create, XRF 553–555
    - CScrollBar, XRF 553
    - GetScrollPos, XRF 555
    - GetScrollRange, XRF 556
    - SetScrollPos, XRF 556–557
    - SetScrollRange, XRF 557
- CScrollBar constructor, XRF 553
- CScrollBar objects, creating
  - constructor, CScrollBar::CScrollBar, XRF 553
  - initializing, CScrollBar::Create, XRF 553–555
- CSize class
  - described, XRF 20, 558
  - member functions, CSize, XRF 559
  - operators, XRF 560–561
- CSize constructor, XRF 559
- CSize objects, creating, CSize::CSize, XRF 559
- CSTARTUP.BAT, LIB 42
- CStatic class
  - described, XRF 562
  - Foundation classes cookbook, XUG 335
  - member functions
    - Create, XRF 563–566
    - CStatic, XRF 563
- CStatic constructor, XRF 563

- CStatic objects, creating
  - attaching, CStatic::Create, XRF 563–564
  - constructor, CStatic::CStatic, XRF 563
- CStdioFile class
  - data members, m_pStream, XRF 571
  - described, XRF 567
  - member functions
    - CStdioFile, XRF 568–569
    - ~CStdioFile, XRF 569
    - ReadString, XRF 569–570
    - WriteString, XRF 570
- CStdioFile constructor, XRF 568–569
- CStdioFile destructor, XRF 569
- CStdioFile objects
  - creating, CStdioFile::CStdioFile, XRF 568
  - destroying, CStdioFile::~CStdioFile, XRF 569
  - reading text into buffer from associated file, CStdioFile::ReadString, XRF 569–570
  - writing data from buffer to associated file, CStdioFile::WriteString, XRF 570
- CString class
  - argument passing conventions, XRF 599
  - assignment operator, CString::operator=, XRF 592
  - basic operations, Foundation classes cookbook, XUG 257
  - casting operator, CString::operator const char* (), XRF 592
  - comparison operators, XRF 596
  - concatenation operator
    - CString::operator +, XRF 594
    - CString::operator +=, XRF 595
  - contents, modifying
    - Foundation classes cookbook, XUG 260
  - described, XRF 28, 572–575
  - diagnostic dumping and storing to archive, CString::operators <<, >>, XRF 593
  - DMTEST sample program, XUG 20, 28–31
  - exception cleanup, XRF 598
  - formal parameters, specifying, XUG 259
  - insertion operator, CString::operators <<, >>, XRF 593

- CString class (*continued*)
  - member functions
    - AnsiToOem, XRF 576
    - C run-time functions, comparison to, XUG 258
    - Collate, XRF 576–577
    - Compare, XRF 577
    - CompareNoCase, XRF 577–578
    - CString, XRF 578–579
    - ~CString, XRF 580
    - Empty, XRF 580
    - Find, XRF 580–581
    - FindOneOf, XRF 581
    - Foundation classes tutorial, XUG 164
    - GetAt, XRF 581–582
    - GetBuffer, XRF 582–583
    - GetBufferSetLength, XRF 583–584
    - GetLength, XRF 584
    - IsEmpty, XRF 584–585
    - Left, XRF 585
    - LoadString, XRF 585–586
    - MakeLower, XRF 586
    - MakeReverse, XRF 586
    - MakeUpper, XRF 587
    - Mid, XRF 587–588
    - OemToAnsi, XRF 588
    - ReleaseBuffer, XRF 588–589
    - ReverseFind, XRF 589
    - Right, XRF 589–590
    - SetAt, XRF 590
    - SpanExcluding, XRF 591
    - SpanIncluding, XRF 591
  - operators, XRF 592–600
  - serialization, Foundation classes tutorial, XUG 31
  - string manipulation, Foundation classes cookbook, XUG 256–261
  - strings as function inputs, XRF 599
  - subscript operator [], XRF 597
- CString constructor, XRF 578–579
- CString destructor, XRF 580

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

- CString objects
  - arrays, CStringArray class described, XRF 601
  - as actual strings, Foundation classes cookbook, XUG 258
  - converting characters from ANSI to OEM character set, CString::AnsiToOem, XRF 576
  - converting characters from OEM to ANSI character set, CString::OemToAnsi, XRF 588
  - converting to lowercase, CString::MakeLower, XRF 586
  - converting to uppercase, CString::MakeUpper, XRF 587
  - creating, CString::CString, XRF 579
  - destroying, CString::~CString, XRF 580
  - exceptions, Foundation classes cookbook, XUG 304
  - extracting
    - first characters from and returning copy, CString::Left, XRF 585
    - largest substring excluding specified characters, CString::SpanExcluding, XRF 591
    - last characters and returning copy, CString::Right, XRF 589–590
    - substring of specified length and returning copy, CString::Mid, XRF 587
  - filename argument, Foundation classes tutorial, XUG 42
  - keyed to CString objects, XRF 389
  - list, CStringList class described, XRF 603
  - making empty string, CString::Empty, XRF 580
  - maps to CObject pointers, XRF 377
  - operations, Foundation classes cookbook, XUG 259–261
  - overwriting specified character, CString::SetAt, XRF 590
  - reading Windows string resource into, CString::LoadString, XRF 586
  - reinitializing with new data, CString::operator=, XRF 592
  - returning
    - count of characters in, CString::GetLength, XRF 584
    - pointer to internal character buffer and matching length, CString::GetBufferSetLength, XRF 583–584
  - reversing character order in, CString::MakeReverse, XRF 586
  - searching for last substring match, CString::ReverseFind, XRF 589
  - terminating use of buffer, CString::ReleaseBuffer, XRF 588
- CString objects (*continued*)
  - testing for empty condition, CString::IsEmpty, XRF 584
  - with variable argument functions, using, XUG 261
- CStringArray class, XRF 601–602
- CStringList class, XRF 603–605
- CTime class
  - date and time management, Foundation classes cookbook, XUG 255–256
  - described, XRF 28, 606–607
  - DMTEST sample program, XUG 20, 28–31
  - member functions
    - CTime, XRF 608
    - Format, XRF 609
    - FormatGmt, XRF 610
    - GetCurrentTime, XRF 610
    - GetDay, XRF 611
    - GetDayofWeek, XRF 611
    - GetGmtTm, XRF 611
    - GetHour, XRF 612
    - GetLocalTm, XRF 613
    - GetMinute, XRF 613
    - GetMonth, XRF 613
    - GetSecond, XRF 614
    - GetTime, XRF 614
    - GetYear, XRF 614
  - operators, XRF 608–617
- ctime function, LIB 173–174
- CTime objects
  - adding and subtracting CTimeSpan object, CTime::operator+=, XRF 616
  - creating, CTime::CTime, XRF 608–609
  - getting struct tm, returning local time, CTime::GetLocalTm, XRF 612
  - getting struct tm, returning UCT, CTime::GetGmtTm, XRF 611
  - getting time_t value, CTime::GetTime, XRF 614
  - returning current time, CTime::GetCurrentTime, XRF 610
- CTimeSpan class
  - described, XRF 28, 618–619
  - member functions
    - CTimeSpan, XRF 620-621
    - Format, XRF 621-622
    - GetDays, XRF 622
    - GetHours, XRF 622
    - GetMinutes, XRF 623
    - GetSeconds, XRF 623
    - GetTotalHours, XRF 623

CTimeSpan class (*continued*)
  member functions (*continued*)
    GetTotalMinutes, XRF 624
    GetTotalSeconds, XRF 624
  operators, XRF 620–627
CTimeSpan objects, creating, CTimeSpan::CTimeSpan, XRF 620–621
CTRL+Z character (end-of-file indicator), LR 4
Curdate function, PWB, ET 151, 165
Curday function, PWB, ET 151, 165
Curfile predefined macro, PWB, ET 222, 224–225
Curfileext predefined macro, PWB, ET 222, 225–226
Curfilenam predefined macro, PWB, ET 222, 225–226
Current date, PWB, ET 165
Current Location command, CodeView, ET 423, 471
Current position
  carets
    displaying, CWnd::ShowCaret, XRF 816–817
    getting, CWnd::GetCaretPos, XRF 686
  scroll bars
    getting, CWnd::GetScrollPos, XRF 696–697
    setting, CWnd::SetScrollPos, XRF 811–812
CURRENT.STS
  CodeView
    overview, ET 344
    saving, ET 360
    toggling status of, ET 343
  PWB, ET 138
Cursor
  defined, ET 805
  PWB commands, ET 154–155
  shape of, in PWB, ET 273
Cursormode switch, PWB, ET 263, 273
Cursors
  called on press of mouse button, CWnd::OnMouseActivate, XRF 758–759
  called when moved within nonclient area, CWnd::OnNcMouseMove, XRF 769
  loading predefined, CWinApp::LoadOEMCursor, XRF 634; CWinApp::LoadStandardCursor, XRF 635

Cursors (*continued*)
  pixel, coordinate systems, PT 186
  setting attributes, _settextcursor function, LIB 681–682
  setting toggle for graphics, _displaycursor function, LIB 179–180
  specified, loading in Windows applications, CWinApp::LoadCursor, XRF 633
Curtime function, PWB, ET 151, 166
Customer assistance. *See* the Product Assistance Request Form in LIB
Customize Project Template command, PWB, ET 75
Customize Run Menu command, PWB, ET 74
Customizing
  About dialog box, QuickWin, PT 151, 157
  AfxRegisterWndClass function, Foundation classes tutorial, XUG 113
  command-line processing, LR 33–34
  icons, QuickWin, PT 164–165
  memory models
    code pointer sizing, PT 70–71
    code segments, specifying, PT 77–78
    data placement, PT 74–76
    data pointer sizing, PT 71
    declarations, defining and referencing, PT 74–76
    library support, PT 74
    module naming, PT 76–77
    option, CLs, PT 70
    segment naming, PT 76–77
    segment setup options, PT 71–74
  OnIdle member function, Foundation classes tutorial, XUG 113
  optimization, PT 9
  output stream manipulators, XUG 381–382
  presentation graphics, PT 219–220
  Windows applications, Foundation classes tutorial, XUG 112–113
  WinMain function, XUG 89
Cut command, PWB, ET 73
  predefined macros, ET 143
Cut member function, CComboBox class, XRF 146
  CEdit class, XRF 289

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

- Cutting, Windows Edit control selection, CEdit::Cut, XRF 289
- CV. *See* CodeView
- Cvdllpath entry, TOOLS.INI file, CodeView, ET 330–331
- CVPACK
  - command line, ET 744
  - error messages. *See* "Error Messages," Part 2 of this book
  - exit codes, ET 745
  - help, ET 745
  - options, ET 744
  - overview, ET 743–744
  - syntax, ET 744
- CVW
  - *See also* CodeView
  - commands, ET 382–385
  - compared to CV, ET 377
  - debugging techniques, ET 386–388
  - multiple applications, debugging, ET 379–382
  - running, ET 378–379
- CWinApp class
  - data members
    - m_hInstance, XRF 639
    - m_hPrevInstance, XRF 639
    - m_lpCmdLine, XRF 639
    - m_msgCur, XRF 639
    - m_nCmdShow, XRF 640, 819
    - m_pMainWnd, XRF 640
    - m_pszAppName, XRF 640
  - deriving from, Foundation classes cookbook, XUG 307
  - described, XRF 11, 628–630
    - Foundation classes cookbook, XUG 307
  - HELLO sample program, XUG 83, 86–87
  - member functions
    - CWinApp, XRF 631
    - ExitInstance, XRF 631
    - InitApplication, XRF 632; XUG 308
    - InitInstance, XRF 632
    - LoadCursor, XRF 633, 775
    - LoadIcon, XRF 176, 633–634, 775
    - LoadOEMCursor, XRF 634
    - LoadOEMIcon, XRF 176, 634–635
    - LoadStandardCursor, XRF 635–636
    - LoadStandardIcon, XRF 176, 636–637
    - OnIdle, XRF 637; XUG 309–310
    - PreTranslateMessage, XRF 638
    - Run, XRF 638
  - overridable member functions, XRF 11
  - overriding, XUG 307
- CWinApp constructor, XRF 631
- CWinApp objects
  - creating, CWinApp::CWinApp, XRF 631
  - returning pointer to, AfxGetApp, XRF 36
- CWindowDC class
  - described, XRF 641
  - member functions
- CWindowDC constructor, XRF 642
- CWindowDC destructor, XRF 642
- CWindowDC objects
  - creating, CWindowDC::CWindowDC, XRF 642
  - destroying, CWindowDC::~CWindowDC, XRF 642
- CWnd
  - clipboard, opening, CWnd::OpenClipboard, XRF 799
  - dialog boxes, searching for previous or next control, CWnd::GetNextDlgGroupItem, XRF 693–694
  - handles, getting safe, CWnd::GetSafeHwnd, XRF 696
  - mouse button, right, XRF 778–780
  - nonclient area
    - calculating size, CWnd::OnNcCalcSize, XRF 762
    - called when destroyed, CWnd::OnNcDestroy, XRF 763
- CWnd class
  - applications, XRF 737
    - called when destroyed, CWnd::OnRenderAllFormats, XRF 780–781
    - confirming choice to terminate, CWnd::OnClose, XRF 720
    - redrawing or preventing redrawing of changes, CWnd::SetRedraw, XRF 811
  - attaching HWND, CWnd::m_hWnd, XRF 822
  - buttons, XRF 732–735
  - buttons, boxes, called when control created, CWnd::OnMeasureItem, XRF 754, 756
  - called for displaying clipboard contents, CWnd::OnAskCbFormatName, XRF 716–717
  - called for mouse capture, CWnd::OnNcHitTest, XRF 763–764
  - called when device-mode settings changed, CWnd::OnDevModeChange, XRF 731
  - called when sessions end, CWnd::OnEndSession, XRF 736
  - caption titles
    - copying into specified buffer, CWnd::GetWindowText, XRF 704
    - returning length, CWnd::GetWindowTextLength, XRF 704

CWnd class (*continued*)
carets
displaying, CWnd::ShowCaret, XRF 816–817
moving to position specified by point, CWnd::SetCaretPos, XRF 806–807
client areas
called after size changed, CWnd::OnSize, XRF 784–785
invalidating, CWnd::Invalidate, XRF 706
retrieving pointer to display context, CWnd::GetDC, XRF 688
updating, CWnd::UpdateWindows, XRF 819
validating within rectangle, CWnd::ValidateRect, XRF 820
clipboard viewers, called with event in vertical scroll bar, CWnd::OnVScrollClipboard, XRF 797–798
combo boxes, comparing items in, CWnd::OnCompareItem, XRF 722, 724
Control menu, called when Maximize or Minimize button selected, CWnd::OnSysCommand, XRF 788, 790
control, specifying input type, CWnd::OnGetDlgCode, XRF 739–740
converting
client coordinates to screen coordinates, CWnd::ClientToScreen, XRF 663
screen coordinates of point or rect to client coordinates, CWnd::ScreenToClient, XRF 802
copying caption into specified buffer, CWnd::GetWindowText, XRF 704
cursor
called on press of mouse button, CWnd::OnMouseActivate, XRF 758–759
called when moved within nonclient area, CWnd::OnNcMouseMove, XRF 769
CWinApp message translator, CWnd::PreTranslateMessage, XRF 801
data members
m_hWnd, XRF 822
wndBottom, XRF 822
wndTop, XRF 822
described, XRF 644–658

CWnd class (*continued*)
determining maximization, CWnd::IsZoomed, XRF 711
device contexts, releasing, CWnd::ReleaseDC, XRF 801–802
dialog objects, Foundation classes tutorial, XUG 165
enabling or disabling mouse or keyboard input, CWnd::EnableWindow, XRF 681–682
fonts, setting, CWnd::SetFont, XRF 809
Foundation classes cookbook, XUG 345
input focus
called after gaining, CWnd::OnSetFocus, XRF 783
called when ALT and another key pressed, CWnd::OnSysKeyDown, XRF 791–792
called with release of key pressed with ALT, CWnd::OnSysKeyUp, XRF 792–794
claiming, CWnd::SetFocus, XRF 809
specifying character value of dead key, CWnd::OnSysDeadChar, XRF 790–791
specifying virtual-key code of Control menu key, CWnd::OnSysChar, XRF 786–787
invalidating client area, CWnd::Invalidate, XRF 707
key, returning active, CWnd::OnChar, XRF 718–719
keyboard, enabling or disabling input, CWnd::EnableWindow, XRF 681–682
list boxes
called with keyboard input, CWnd::OnVKeyToItem, XRF 795–796
retrieving current selection, CWnd::DlgDirSelect, XRF 679–680
returning application response, CWnd::OnCharToItem, XRF 719–720
member functions
ArrangeIconicWindows, XRF 659
Attach, XRF 659
BeginPaint, XRF 500, 660
BringWindowToTop, XRF 661
ChangeClipboardChain, XRF 661
CheckDlgButton, XRF 661–662
CheckRadioButton, XRF 662
ChildWindowFromPoint, XRF 662–663
ClientToScreen, XRF 663
CloseWindow, XRF 664

CWnd class (*continued*)
 member functions (*continued*)
  Create, XRF 664–665
  CreateCaret, XRF 665
  CreateEx, XRF 666–670
  CreateGrayCaret, XRF 671
  CreateSolidCaret, XRF 672
  CWnd, XRF 673
  ~CWnd, XRF 673
  Default, XRF 673
  DefWindowProc, XRF 674
  DeleteTempMap, XRF 674
  DestroyWindow, XRF 265, 267, 675
  Detach, XRF 676
  DlgDirList, XRF 676–677
  DlgDirListComboBox, XRF 678–679
  DlgDirSelect, XRF 679–680
  DlgDirSelectComboBox, XRF 680–681
  DrawMenuBar, XRF 418, 422, 430, 433–434, 681
  EnableWindow, XRF 681–682
  EndPaint, XRF 682
  FindWindow, XRF 683
  FlashWindow, XRF 683–684
  FromHandle, XRF 684
  GetActiveWindow, XRF 685
  GetCapture, XRF 685
  GetCaretPos, XRF 686
  GetCheckedRadioButton, XRF 686
  GetClientRect, XRF 686–687
  GetClipboardOwner, XRF 687
  GetClipboardViewer, XRF 687
  GetCurrentMessage, XRF 688
  GetDC, XRF 174, 688
  GetDesktopWindow, XRF 689
  GetDlgCtrlID, XRF 689
  GetDlgItem, XRF 689–690
  GetDlgItemInt, XRF 690–691
  GetDlgItemText, XRF 691
  GetFocus, XRF 692
  GetFont, XRF 692
  GetLastActivePopup, XRF 692–693
  GetMenu, XRF 693
  GetNextDlgGroupItem, XRF 693–694
  GetNextDlgTabItem, XRF 694
  GetNextWindow, XRF 695
  GetParent, XRF 696
  GetSafeHwnd, XRF 696
  GetScrollPos, XRF 696–697
  GetScrollRange, XRF 697
  GetStyle, XRF 698
  GetSuperWndProcAddr, XRF 698

CWnd class (*continued*)
 member functions (*continued*)
  GetSysModalWindow, XRF 698
  GetSystemMenu, XRF 699
  GetTopWindow, XRF 700
  GetUpdateRect, XRF 700–701
  GetUpdateRgn, XRF 701
  GetWindow, XRF 702
  GetWindowDC, XRF 703
  GetWindowRect, XRF 703–704
  GetWindowText, XRF 704
  GetWindowTextLength, XRF 704–705
  HideCaret, XRF 705
  HiliteMenuItem, XRF 705–706
  Invalidate, XRF 706–707
  InvalidateRect, XRF 302, 707
  InvalidateRgn, XRF 708
  IsChild, XRF 709
  IsDlgButtonChecked, XRF 709
  IsIconic, XRF 710
  IsWindowEnabled, XRF 710
  IsWindowVisible, XRF 710
  IsZoomed, XRF 711
  KillTimer, XRF 711
  MessageBox, XRF 711–714
  MoveWindow, XRF 714–715
  OnActivate, XRF 715
  OnActivateApp, XRF 716
  OnAskCbFormatName, XRF 716–717
  OnCancelMode, XRF 717
  OnChangeCbChain, XRF 717–718
  OnChar, XRF 718–719
  OnCharToItem, XRF 719–720
  OnChildActivate, XRF 720
  OnClose, XRF 720
  OnCommand, XRF 721
  OnCompacting, XRF 722
  OnCompareItem, XRF 722–724
  OnCreate, XRF 724–726
  OnCtlColor, XRF 726–727
  OnDeadChar, XRF 727–729
  OnDeleteItem, XRF 729–730
  OnDestroy, XRF 730
  OnDestroyClipboard, XRF 731
  OnDevModeChange, XRF 731
  OnDrawClipboard, XRF 731–732
  OnDrawItem, XRF 732–735
  OnEnable, XRF 735–736
  OnEndSession, XRF 736
  OnEnterIdle, XRF 737
  OnEraseBkgnd, XRF 737–738

CWnd class (*continued*)
member functions (*continued*)
OnFontChange, XRF 738–739
OnGetDlgCode, XRF 739–740
OnGetMinMaxInfo, XRF 740–741
OnHScroll, XRF 741–742
OnHScrollClipboard, XRF 742–743
OnIconEraseBkgnd, XRF 743
OnInitMenu, XRF 744
OnInitMenuPopup, XRF 744–745
OnKeyDown, XRF 745–746
OnKeyUp, XRF 746–747
OnKillFocus, XRF 747–748
OnLButtonDblClk, XRF 748–749
OnLButtonDown, XRF 749
OnLButtonUp, XRF 750
OnMButtonDblClk, XRF 751
OnMButtonDown, XRF 752
OnMButtonUp, XRF 752–753
OnMDIActivate, XRF 753–754
OnMeasureItem, XRF 754–756
OnMenuChar, XRF 756–757
OnMenuSelect, XRF 757–758
OnMouseActivate, XRF 758–759
OnMouseMove, XRF 760
OnMove, XRF 760–761
OnNcActivate, XRF 761
OnNcCalcSize, XRF 762
OnNcCreate, XRF 762–763
OnNcDestroy, XRF 763
OnNcHitTest, XRF 763–764
OnNcLButtonDblClk, XRF 765
OnNcLButtonDown, XRF 765–766
OnNcLButtonUp, XRF 766
OnNcMButtonDblClk, XRF 767
OnNcMButtonDown, XRF 767–768
OnNcMButtonUp, XRF 768
OnNcMouseMove, XRF 769
OnNcPaint, XRF 769
OnNcRButtonDblClk, XRF 770
OnNcRButtonDown, XRF 770–771
OnNcRButtonUp, XRF 771
OnPaint, XRF 772

CWnd class (*continued*)
member functions (*continued*)
OnPaintClipboard, XRF 772–773
OnPaintIcon, XRF 773
OnPaletteChanged, XRF 773–774
OnParentNotify, XRF 774–775
OnQueryDragIcon, XRF 775–776
OnQueryEndSession, XRF 776
OnQueryNewPalette, XRF 777
OnQueryOpen, XRF 777
OnRButtonDblClk, XRF 778
OnRButtonDown, XRF 779
OnRButtonUp, XRF 780
OnRenderAllFormats, XRF 780–781
OnRenderFormat, XRF 781
OnSetCursor, XRF 782
OnSetFocus, XRF 783
OnShowWindow, XRF 783–784
OnSize, XRF 784–785
OnSizeClipboard, XRF 785
OnSpoolerStatus, XRF 786
OnSysChar, XRF 786–787
OnSysColorChange, XRF 788
OnSysCommand, XRF 788–790
OnSysDeadChar, XRF 790–791
OnSysKeyDown, XRF 791–792
OnSysKeyUp, XRF 792–794
OnTimeChange, XRF 794
OnTimer, XRF 795
OnVKeyToItem, XRF 795–796
OnVScroll, XRF 796–797
OnVScrollClipboard, XRF 797–798
OnWinIniChange, XRF 798–799
OpenClipboard, XRF 799
OpenIcon, XRF 800
PostMessage, XRF 800–801
PreTranslateMessage, XRF 801
ReleaseDC, XRF 174, 801–802
ScreenToClient, XRF 802
ScrollWindow, XRF 803–804
SendDlgItemMessage, XRF 804
SendMessage, XRF 805
SetActiveWindow, XRF 805–806

CWnd class (*continued*)
member functions (*continued*)
SetCapture, XRF 806
SetCaretPos, XRF 806–807
SetClipboardViewer, XRF 807
SetDlgItemInt, XRF 808
SetDlgItemText, XRF 808
SetFocus, XRF 809
SetFont, XRF 809
SetMenu, XRF 810
SetParent, XRF 810
SetRedraw, XRF 811
SetScrollPos, XRF 811–812
SetScrollRange, XRF 812–813
SetSysModalWindow, XRF 813
SetTimer, XRF 813–814
SetWindowPos, XRF 814–816
SetWindowText, XRF 297, 816
ShowCaret, XRF 816–817
ShowOwnedPopups, XRF 817
ShowScrollBar, XRF 817–818
ShowWindow, XRF 640, 818–819
UpdateWindow, XRF 819
ValidateRect, XRF 820
ValidateRgn, XRF 820
WindowFromPoint, XRF 821
WindowProc, XRF 821
menu item, called when control created, CWnd::OnMeasureItem, XRF 754, 756
menu mnemonic character, called when user presses, CWnd::OnMenuChar, XRF 756–757
menus, setting current to specified, CWnd::SetMenu, XRF 810
message handlers
Foundation classes tutorial, XUG 203
message handling upon selection of item, CWnd::OnCommand, XRF 721
messages, sending to specified control, CWnd::SendDlgItemMessage, XRF 804
minimizing, CWnd::CloseWindow, XRF 664
mouse button, left
called when double-clicked, CWnd::OnLButtonDblClk, XRF 748-749
called when double-clicked in nonclient area, CWnd::OnNcLButtonDblClk, XRF 765
called when pressed, CWnd::OnLButtonDown, XRF 749
called when pressed in nonclient area, CWnd::OnNcLButtonDown, XRF 765–766
called when released, CWnd::OnLButtonUp, XRF 750

CWnd class (*continued*)
mouse button, middle
called when double-clicked, CWnd::OnMButtonDblClk, XRF 751
called when double-clicked in nonclient area, CWnd::OnNcMButtonDblClk, XRF 767
called when pressed, CWnd::OnMButtonDown, XRF 752
called when pressed in nonclient area, CWnd::OnNcMButtonDown, XRF 767–768
called when released, CWnd::OnMButtonUp, XRF 752–753
called when released in nonclient area, CWnd::OnNcMButtonUp, XRF 766–768
mouse button, right
called when double-clicked in nonclient area, CWnd::OnNcRButtonDblClk, XRF 770
called when pressed in nonclient area, CWnd::OnNcRButtonDown, XRF 770–771
called when released in nonclient area, CWnd::OnNcRButtonDown, XRF 771
mouse capture, retrieving, CWnd::GetCapture, XRF 685
mouse cursor
called when input isn't captured, CWnd::OnSetCursor, XRF 782
called when moved, CWnd::OnMouseMove, XRF 760
mouse enumerated values (list), XRF 88
mouse input, causing all subsequent to be sent to current object, CWnd::SetCapture, XRF 806
mouse, enabling or disabling input, CWnd::EnableWindow, XRF 681–682
nonclient area, called when needing painting, CWnd::OnNcPaint, XRF 769
nonsystem key
called on input, CWnd::OnKeyDown, XRF 745–746
called on release, CWnd::OnKeyUp, XRF 746–747
painting
called when repainting, CWnd::OnPaint, XRF 772
preparing for, CWnd::BeginPaint, XRF 660
Print Manager, called when job added or deleted from queue, CWnd::OnSpoolerStatus, XRF 786
removing windows from clipboard viewer chain, CWnd::OnChangeCbChain, XRF 717–718
returning specified class, XRF 683
scroll bars
hiding, CWnd::ShowScrollBar, XRF 817–818
setting range of position values, CWnd::SetScrollRange, XRF 812–813

CWnd class (*continued*)
  scroll bars (*continued*)
    vertical, called when clicked, CWnd::OnVScroll, XRF 796–797
  scroll boxes, setting to specified position, CWnd::SetScrollPos, XRF 811–812
  setting
    control text owned by CWnd, CWnd::SetDlgItemText, XRF 808
    control text to specified integer value, CWnd::SetDlgItemInt, XRF 808
  system time, called after change, CWnd::OnTimeChange, XRF 794
  window-manager's list, searching for windows, CWnd::GetWindow, XRF 702
  windows
    containing given point, identifying, CWnd::WindowFromPoint, XRF 821
    making active, CWnd::SetActiveWindow, XRF 805–806
    specifying memory compaction time, CWnd::OnCompacting, XRF 722
  Windows initialization file, called after change made, CWnd::OnWinIniChange, XRF 798–799
  Windows windows
    attaching to CWnd object, CWnd::Attach, XRF 659
    returning maximized position or dimensions, or tracking size range, CWnd::OnGetMinMaxInfo, XRF 740–741
  windows, child
    called on activation or deactivation, CWnd::OnMDIActivate, XRF 753–754
    changing parent, CWnd::SetParent, XRF 810
    constructor, CWnd::Cwnd, XRF 673
    creating and attaching to object, CWnd::Create, XRF 664–665
    creating with extended style, CWnd::CreateEx, XRF 666–670
  windows, pop-up, showing or hiding, CWnd::ShowOwnedPopups, XRF 817
  WS_TABSTOP style control, retrieving pointer to, CWnd::GetNextDlgTabItem, XRF 694
CWnd constructor, XRF 673
CWnd destructor, XRF 673
CWnd objects
  button control, determining if check-marked, CWnd::IsDigButtonChecked, XRF 709
  called after CWnd moved, CWnd::OnMove, XRF 760–761
  called when about to be shown or hidden, CWnd::OnShowWindow, XRF 783–784
  called when activating for different task, CWnd::OnActivateApp, XRF 716
  called when activating or deactivating, CWnd::OnActivate, XRF 715
  called when application creates, CWnd::OnCreate, XRF 724–726
  called when background needs erasing, CWnd::OnEraseBkgnd, XRF 737–738
  called when enabled state is changed, CWnd::OnEnable, XRF 735–736
  changing position and dimensions, CWnd::MoveWindow, XRF 714–715
  copying dimensions of bounding rectangle, CWnd::GetWindowRect, XRF 703–704
  creating, called prior to WM_CREATE message, CWnd::OnNcCreate, XRF 762–763
  deleting temporary, CWnd::DeleteTempMap, XRF 674
  destroying
    called to inform, CWnd::OnDestroy, XRF 730
    destructor, CWnd::~CWnd, XRF 673
  detaching Windows handle, CWnd::Detach, XRF 676
  displaying CWnd, CWnd::ShowWindow, XRF 818–819
  enabling for mouse and keyboard input, CWnd::IsWindowEnabled, XRF 710
  iconized, called when user requests open window, CWnd::OnQueryOpen, XRF 777
  ID, returning, CWnd::GetDlgCtrlID, XRF 689
  input focus, called before losing, CWnd::OnKillFocus, XRF 747–748
  making into system-modal window, CWnd::SetSysModalWindow, XRF 813

- CWnd objects (*continued*)
  - menu items, called when user selects, CWnd::OnMenuSelect, XRF 757–758
  - messages, placing in queue, CWnd::PostMessage, XRF 800
  - minimized, called when background must be filled before painting, CWnd::OnIconEraseBkgnd, XRF 743
  - minimizing, CWnd::IsIconic, XRF 710
  - moving to end of window list, CWnd::WndBottom, XRF 822
  - moving to top of window list, CWnd::WndTop, XRF 822
  - providing Windows procedure for, CWnd::WindowProc, XRF 821
  - returning pointer to when given handle to window, CWnd::FromHandle, XRF 684
  - scrolling, CWnd::ScrollWindow, XRF 803–804
  - with focus, displaying dialog or message boxes, CWnd::OnCancelMode, XRF 717
- CWordArray class, described, XRF 823–824
- CX register, CodeView syntax, ET 419, 450
- CXX command macro, NMAKE, ET 676
- .CXX files, defined, ET 806
- CXXFLAGS options macro, NMAKE, ET 676
- Cyan, color value, ET 272
- _CYAN constant, PT 180

D

- /D option
  - CL, ET 429–432, 492–493; LR 195, 206, 267
  - HELPMAKE, ET 714
  - NMAKE, ET 648
  - PWB, ET 141
- d. context prefix, HELPMAKE, ET 729
- /D options, CL, consistency rules, precompiled headers, PT 39
- /DA option, PWB, ET 141
- Dark Gray, color value, ET 273
- Dash (-)
  - character classes, PWB, ET 92
  - character ranges, regular expression syntax, ET 779
  - CL syntax, ET 488
  - command line, NMAKE, ET 647
  - command modifier, NMAKE, ET 661
  - Delete command, LIB, ET 700, 704
  - HELPMAKE options, ET 711
- .DAT files
  - defined, ET 806
- Data
  - analyzing series of, _pg_analyzechart functions, LIB 549–551
  - archive
    - determining if loading, CArchive::IsLoading, XRF 98
    - determining if storing, CArchive::IsStoring, XRF 98
  - dumping, CodeView, ET 438–439
  - entering, CodeView, ET 440–441
  - exporting module-definition files, ET 623
  - extracting from streams, istream::get, XRF 878, 880
  - importing module-definition files, ET 624–625
  - moving blocks, CodeView, ET 442–443
  - reading from files, _read function, LIB 611–612
- _DATA (default data segment)
  - defined by __near keyword, LR 56
- Data abstraction, TUT 163–164
- Data allocation
  - *See also* Memory models
  - based, PT 86–88
  - CL options, ET 523–524
  - far, PT 65–66
- _DATA constant, C++, LR+ 415
- Data directives, inline assembly limitations, PT 113
- Data files, portability limitations, PT 292
- Data hiding. *See* Encapsulation
- Data interface, simplifying, Phone Book sample program, XUG 122–134
- Data members
  - accessing inline assembly, PT 117–118
  - static, described, LR+ 247–248
- Data menu, CodeView, ET 364–368
- Data model
  - C++ objects, Foundation classes tutorial, XUG 19
  - creating, DMTEST sample program, XUG 17
  - defined, Foundation classes tutorial, XUG 18
  - DMTEST sample program, XUG 19
  - implementation file, DMTEST sample program, XUG 22
  - interface file, DMTEST sample program, XUG 22
  - reusability, DMTEST sample program, XUG 19
  - testing, DMTEST sample program, XUG 49, 51–64
  - user interface, independence from Foundation classes tutorial, XUG 19
- Data pointers
  - *See also* Pointers
  - memory model codes, CL options, ET 489–490
- Data segment register, LINK, ET 579

Data segments
  defining attributes, module-definition files, ET 618–620
  loading data, LINK, ET 579
  memory models, CL options, ET 488–490
  naming, CL option, ET 528–530
  naming, custom memory models, PT 76–77
  overlaid DOS programs, ET 598, 603–605
  packing, LINK, ET 588–589; PT 27
  stack segments, equality with, PT 71–74
Data series descriped, PT 202
DATA statement, module-definition files, ET 609, 618–620
Data storage
  class-member, LR+ 239
  portability guidelines, PT 274–276
Data symbol defined, ET 806
Data threshold, PT 75
  setting, CL option, ET 522
Data types
  p-code instructions, PT 48–49
  portability guidelines, PT 271–274
Data windows, presentation graphics, PT 204
data_seg pragma
  new in version 7.0, LR 258
  precompiled header compilation, effect on, PT 41
data_seg pragma, C++, LR+ 388
Database
  browser. *See* Browser database
  help
    context prefixes, ET 729
    creating, ET 711–712
    decoding, ET 713–714
    overview, ET 710–711
Databases
  creating
    Foundation classes tutorial, XUG 58, 139
  destroying
    Foundation classes tutorial, XUG 58
  member functions
    Phone Book sample program, XUG 135
  opening
    Foundation classes tutorial, XUG 139
Databases (*continued*)
  serialization
    Foundation classes tutorial, XUG 131–133
Data-conversion routines, LIB 20
Date
  copying to buffers, _strdate function, LIB 751–752
  current in PWB, ET 165
  getting date file written, _dos_getftime function, LIB 204–206
  setting for files, _dos_setftime function, LIB 224–226
  system
    getting, _dos_getdate function, LIB 196–197
    setting, _dos_setdate function, LIB 218–219
__DATE__ macro, C++, LR+ 374
Date management
  described
    Foundation classes cookbook, XUG 255–256
__DATE__ predefined macro
  default, LR 247
  described, LR 199
Day
  current in PWB, ET 165
daylight variable, LIB 62
Days
  of month, CTime::GetDay, XRF 611
  of week, CTime::GetDayOfWeek, XRF 611
  hours in current, getting, CTimeSpan::GetHours, XRF 622
  span, getting, CTimeSpan::GetDays, XRF 622
.DBG files defined, ET 806
Dblclick switch, PWB, ET 263, 274
dbp member function, streambuf class, XRF 923
DBL_DIG constant, C++
  floating limits, LR+ 63–64
DBL_EPSILON constant, C++
  floating limits, LR+ 63–64
DBL_MANT_DIG constant, C++
  floating limits, LR+ 63–64
DBL_MAX constant, C++
  floating limits, LR+ 63–64
DBL_MAX_10_EXP constant, C++
  floating limits, LR+ 63–64

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

DBL_MAX_EXP constant, C++
  floating limits, LR+ 63–64
DBL_MIN constant, C++
  floating limits, LR+ 63–64
DBL_MIN_10_EXP constant, C++
  floating limits, LR+ 63–64
DBL_MIN_EXP constant, C++
  floating limits, LR+ 63–64
DBL_RADIX constant, C++
  floating limits, LR+ 63–64
Dead keys
  defined, XRF 728
  returning character value, CWnd::OnDeadChar, XRF 727–729
  specifying character value, CWnd::OnSysDeadChar, XRF 790–791
Dead-store elimination, PT 8
Deallocating
  heap space, Foundation classes cookbook, XUG 304
  memory blocks, free functions, LIB 306–308
  memory, C++, delete operator, LR+ 323–325
  virtual memory blocks, _vfree function, LIB 844
Debug command, PWB, ET 74
Debug environment, assertions, XRF 29
_DEBUG flag, Foundation classes tutorial, XUG 12, 25, 50, 203
Debug mode
  diagnostic reporting, Foundation classes tutorial, XUG 125
  makefile defaults, XUG 9
  NMAKE, XUG 12
  PWB, XUG 12
  release mode, switching from, XUG 12
DEBUG_NEW macro, XRF 38
  Foundation classes cookbook, XUG 296
  writing ASCII information on stdout, streambuf::dbp, XRF 923
Debug options, finding symbols, PWB, ET 101
Debugger
  defined, ET 806
  symbolic, optimizations, removing, PT 9
Debugging
  See also CodeView
  assembly language, ET 412–415
  ASSERT macro, Foundation classes cookbook, XUG 288–289
  AssertValid member function, Foundation classes tutorial, XUG 138
  CL options, ET 553
Debugging (*continued*)
  CodeView
    options, ET 338–343
    setting, ET 344
    using, Foundation classes tutorial, XUG 11, 14
    with inline assembly code, PT 115
  controlling memory allocator, afxMemDF variable, XRF 46
  CVW
    commands, ET 382–385
    compared to CV, ET 377–382
    multiple applications, ET 380–382
    multiple instances, ET 378–380
    techniques, ET 386–388
  DEBUG_NEW macro, XRF 38
    Foundation classes cookbook, XUG 296
  diagnostic memory tracking, AfxEnableMemoryTracking, XRF 49
  diagnostic services described, XRF 28
  diagnostics, Foundation classes cookbook, XUG 285–296
  features, XUG 285
  heap-related problems
    _heapchk functions, LIB 410–412
    _heapset functions, LIB 415–417
    _heapwalk functions, LIB 418–421
  identifying bugs, ET 326
  information, overriding CodeView, PT 38–39
  inline-assembly code, with CodeView, PT 115
  locating bugs, ET 326
  macros, LR 196
  makefiles, NMAKE, ET 648–649
  messages. *See* “Error Messages,” Part 2 of this book
  p-code, ET 389–393; PT 45
  precompiled header object files, PT 38–39
  precompiled headers, LR 261
  programs
    preparing, ET 321–325
    PWB, ET 28–34
  remote
    bit rate, ET 396
    options, ET 396
    overview, ET 393
    requirements, ET 393–395
    starting a session, ET 397–398
    syntax, ET 396
  specifying libraries, LINK, ET 566
  syntax, TOOLS.INI file entries, ET 330–336
  TRACE macro, Foundation classes cookbook, XUG 288

- Debugging (*continued*)
  - tuning allocation diagnostics, afxMemDF variable, XRF 46
  - watch expressions, setting, ET 326–327
  - Windows programs, Foundation classes tutorial, XUG 11, 14
- Debugging information. *See* Symbolic Debugging Information
- Debugging Information Compactor. *See* CVPACK
- Decimal constants. *See* Integer constants
- Declaration statements, C++
  - automatic object declaration, LR+ 149–151
  - described, LR+ 134, 149–154
  - static object declaration, LR+ 152–154
- Declarations
  - arrays described, LR 74–76
  - clarified from definitions, LR 43
  - custom memory models, defining and referencing in, PT 74–76
  - defining, LR 47
  - function prototypes, LR 181–183
  - in declaration-list, LR 168
  - keywords, LR 55–61
  - overview, LR 41–43
  - placement in source file, LR 153, 165
  - pointers, LR 76–79
  - structures, LR 65–71
  - union, LR 71–74
  - warnings, LR 262, 266
- Declarations, C++
  - *See also* Declarators, C++; Definitions, C++
  - arrays, unsized, in member lists, LR+ 239
  - class members, LR+ 237–238
  - class types, LR+ 228
  - classes
    - friends, defining in, LR+ 295
    - type names, using in, LR+ 238
  - constructors, LR+ 302–304
  - conversion functions, LR+ 317–318
  - defined, LR+ 25–26
  - derived classes, LR+ 259
  - described, LR+ 27, 155–156
  - destructors, LR+ 306–307
- Declarations, C++ (*continued*)
  - empty classes, LR+ 232
  - enumeration
    - conversion by integral promotion, LR+ 177–178
    - definition, LR+ 177
    - described, LR+ 173–176
    - names, LR+ 176
  - friends, LR+ 294–295
  - grammar summary, LR+ 427–429
  - linkage specifications
    - calling conventions, effect on, LR+ 178–179
    - described, LR+ 178–181
  - matching, overloaded functions, LR+ 342–343
  - multiple declarations
    - described, LR+ 28
    - limitations, LR+ 28
  - placement, TUT 13–14
  - point of declaration, LR+ 29–30
  - prototypes, LR+ 155
  - specifiers
    - described, LR+ 156
    - friend, LR+ 167
    - function, LR+ 159–163
    - storage-class, LR+ 157–158
    - type, LR+ 168–173
    - typedef, LR+ 163–167
  - uses (list), LR+ 155
- Declarators
  - defined, LR 43, 54
  - initializing, LR 91–98
  - names of exported functions, LR 172
  - restrictions, LR 55, 244
- Declarators, C++
  - abstract
    - arrays, LR+ 199–202
    - default arguments, LR+ 210–212
    - described, LR+ 187
    - function, LR+ 203–210
    - pointers, LR+ 188–190
    - pointers to members, LR+ 196–198
    - references, LR+ 190–196
  - defined, LR+ 183
  - described, LR+ 183–185

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

Declarators, C++ (*continued*)
  function definitions, LR+ 213–216
  grammar summary, LR+ 429–431
  initializers
    described, LR+ 217–218
  modification. *See* Modifiers, C++
  type name use, LR+ 185–187
DECLARE_DYNAMIC macro, XRF 38–39, 468
  Foundation classes cookbook, XUG 264–266
DECLARE_MESSAGE_MAP macro
    Foundation classes cookbook, XUG 313–314, 343, 352
  Foundation classes tutorial, XUG 95
DECLARE_SERIAL macro, XRF 39–40, 468
  CObject class, Foundation classes tutorial, XUG 24
  CPerson class, Foundation classes tutorial, XUG 33
  Foundation classes cookbook, XUG 281
  Foundation classes tutorial, XUG 38, 44
Declaring
  addresses, keywords for, PT 64–66
  arrays, mixed-language programming, PT 263, 265
  bit fields, LR 69
  classes, TUT 42
  functions, __near and __far, PT 66–68
  objects, TUT 43, 48
  pointers, LR 77
  struct or union tags, LR 263
  variables
    floating-point types, PT 127–129
    multiple, LR 42
    near, far, huge and based, PT 65–66
    volatile, LR 53
Decoding, HELPMAKE options, ET 713–714
Decomposition
  object-oriented, TUT 172
  procedural, TUT 172
Decompressing
  help database, ET 714
  help files, specifying buffer size, ET 332
Decorated names
  debugging considerations, ET 324
  overview, ET 789–790
  specifying, ET 790–791
Decreasing program size, PT 9–10
Decrement operator, C++
  overloading, LR+ 355–358
  postfix expressions, LR+ 90–91
  unary-operator expressions, LR+ 94–95
.DEF files. *See* Module-definition files
Default arguments, C++
  declarators, LR+ 210–212
  overloaded functions, TUT 22
  scope, LR+ 212, 283
  tutorial information, TUT 11–12
Default constructors, TUT 54, 59, 94, 116
Default keys
  PWB, ET 146–150
    menu commands, ET 142–149
default labels in switch statements, LR 161–164
Default libraries
  defined, ET 806
  LINK, ET 568, 583–584
Default member function, CWnd class, XRF 673
default statements, C++
  labels, use restrictions, LR+ 135–136, 139–142
Default values, LINK, ET 573
Default window procedures, calling from a message-handler, Foundation classes cookbook, XUG 319–320
_DEFAULTMODE constant, PT 168–169
Defaults
  auto storage class, LR 47
  bit fields, LR 71
  characters per line, LR 212
  CWnd class message handler functions, XUG 203
  floating-point math packages, PT 131
  makefiles, Foundation classes tutorial, XUG 9
  memory models, PT 56, 62–63
  message handler values, Foundation classes tutorial, XUG 100
  Microsoft C extensions enabled, LR 108
  optimization, PT 8–9
  pointer sizes, PT 58, 283
  pointers, LR 60
  signals, LR 249
  storage class, LR 176
  structure packing size, LR 70
  symbol characteristics, LR 172
  translation date, LR 199
  translation time, LR 199
  type specifier int, LR 43
deffile field, LINK, ET 570–572
Define Mark command, PWB, ET 73
#define preprocessor directive
  described, LR 190–197
  enumerations an alternative, LR 62
#define preprocessor directive, C++, LR+ 368–370
defined directive, C++ header file, TUT 61

DEFINED operator, NMAKE, ET 690–691
defined operator
  described, LR 195
  used with #if and #elif, LR 204–206
defined preprocessor operator, C++, LR+ 381
Defining
  class types, LR+ 228–230
  classes, LR+ 230
  constants, CL, ET 492–493
  declarations described, LR 45–47
  friends in class declarations, LR+ 295
  identifiers, LR 5
  locales, setlocale function, LIB 668–669
  macros, LR 192
    __asm blocks, PT 123–124
    CL, ET 492–493
  manifest constants, LR 192
  type names, C++, LR+ 59–60
Definitions
  clarified from declarations, LR 43
  defined, LR 41
  serving as declarations, LR 165
Definitions, C++
  See also Declarations, C++
  defined, LR+ 25–26
  described, LR+ 28
  function, described, LR+ 213–216
Definitions of terms, C++, LR+ 25–26
Deflang switch, PWB, ET 263, 274
DefWindowProc class, default message handling, Foundation classes tutorial, XUG 100
DefWindowProc member function, CWnd class, XRF 674
Defwinstyle switch, PWB, ET 263, 275
Delay command, CodeView, ET 423, 475
delbuf member function, ios class, XRF 856–857
Delete command
  LIB, ET 700, 704
  PWB
    described, ET 73
    predefined macros, ET 143
Delete function, PWB, ET 151, 166
delete operator, PT 105–106
  array deallocation, TUT 66, 96, 99
  class scope, TUT 103–104, 106
  Foundation classes tutorial, XUG 31, 41, 46–47
  memory deallocation, LR+ 323–325
  overloading, TUT 101–102
  overview, TUT 65
  unary-operator expressions, LR+ 101–102
Delete Watch command, CodeView, ET 364–365, 423
Delete Watch dialog box, CodeView, ET 365
Delete Watch Expressions command, CodeView, ET 461
DELETED directory, backup utilities, ET 747–748
DeleteDC member function, CDC class, XRF 174–175
DELETEITEMSTRUCT structure, XRF 80
  CWnd::OnDeleteItem, XRF 729
DeleteMenu member function
  CMenu class, XRF 422
DeleteObject member function
  CGdiObject class, XRF 346–347
    CDialog::OnSetFont, XRF 272
DeleteString member function
  CComboBox class, XRF 147
    CWnd::OnDeleteItem, XRF 729
  CListBox class, XRF 359
    CWnd::OnDeleteItem, XRF 729
DeleteTempMap member function
  CGdiObject class, XRF 347
  CWnd class, XRF 674
Deleting
  array elements, Foundation classescookbook, XUG 274
  breakpoints, CodeView, ET 367
  CGdiObject object, CGdiObject::DeleteObject, XRF 346
  CGdiObject, temporary, CGdiObject::DeleteTempMap, XRF 347
  characters, PWB, ET 161, 167–168, 209–210

Deleting (*continued*)
  combo boxes
    edit control selection, CComboBox::Clear, XRF 143
    edit control selection, CComboBox::Cut, XRF 146
  databases, Foundation classes tutorial, XUG 47
  device contexts attached to CDC object, CDC::DeleteDC, XRF 174
  files
    during debugging session, ET 359
    EXP, ET 750
    PWB, ET 48
    RM, ET 748–749
    specified by filename, remove function, LIB 624
    specified by path, CFile::Remove, XRF 315
    specified by path, _unlink function, LIB 831–832
  graphics objects, Foundation classes cookbook, XUG 347
  items from list or combo boxes, called when, CWnd::OnDeleteItem, XRF 729–730
  lines, PWB, ET 176–177
  list objects, Foundation classes cookbook, XUG 273
  map elements,Foundation classes cookbook, XUG 274
  marks, PWB, ET 178–179
  menu items, CMenu::DeleteMenu, XRF 422
  object modules, ET 704
  objects in a CObject collection, Foundation classes cookbook, XUG 273
  open files, LR 251
  strings
    from list boxes, CListBox::DeleteString, XRF 359
    list box in combo box, CComboBox::DeleteString, XRF 147
  temporary CWnd objects, CWnd::DeleteTempMap, XRF 674
  text, PWB, ET 166, 232
  watch expressions, CodeView, ET 365, 461
  Windows DC attached to CDC object, CDC::DeleteDC, XRF 174
Delimiters
  help (>>), ET 726, 728
  regular expressions, PWB, ET 91
Denormalized numbers, floating-point math packages, PT 128–129
Dependencies
  executing, NMAKE, ET 648
  PWB programs, ET 45, 48
Dependency lines
  defined, ET 655
  dependents, NMAKE, ET 659–660
Dependency tree
  building, NMAKE, ET 655
  updating dependents, NMAKE, ET 659
Dependent files
  defined, ET 646
  dependency lines, NMAKE, ET 655
Dependents
  dependency lines, NMAKE, ET 659–660
  inferred, NMAKE, ET 659, 685–686
Dereference Global Handle command, CodeView, ET 463–464
Dereference Local Handle command, CodeView, ET 466–467
Dereferencing
  C++, defined, LR+ 25–26
  memory handles, CodeView, ET 386
Derivation of classes, XRF 10
Derived classes
  abstract
    described, LR+ 265–266, 280–281
  access to base classes, TUT 112, 130–131
  conversion to base class, TUT 116
  described, LR+ 259
  Foundation classes tutorial, XUG 166
  multiple inheritance
    access control, LR+ 297
    base classes, LR+ 267–271
    described, LR+ 264–265
    name ambiguities, LR+ 271–274
  overriding, Foundation classes cookbook, XUG 307
  overview, TUT 111
  pointers to members
    conversion from pointers to base classes, LR+ 76
  scope, LR+ 282–284
  single inheritance, LR+ 259–264
  virtual functions, LR+ 265, 275–279
Derived types, C++
  composed, LR+ 58–59
  described, LR+ 52
  directly derived, LR+ 52–57
Description blocks
  dependency lines, ET 655
  NMAKE, ET 656–658
    commands, ET 660
    dependents, ET 659–660
DESCRIPTION statement, module-definition files, ET 609, 613–614

Deserialization
- failure of, Foundation classes tutorial, XUG 65
- of objects, Foundation classes cookbook, XUG 285
- OnOpen member function, using, Foundation classes tutorial, XUG 207
- person objects, recreation, Foundation classes tutorial, XUG 52
- procedure, Foundation classes tutorial, XUG 42–44, 51–52

Design
- object-oriented principles, TUT 161, 172–173
- philosophy, Foundation class library, XRF 7–8

Desktop window, returning, CWnd::GetDesktopWindow, XRF 689

Destroying
- CArchive objects, CArchive::~CArchive, XRF 96
- CClientDC objects, CClientDC::~CClientDC, XRF 138
- CDC objects, CDC::~CDC, XRF 170
- CFrameWnd objects, CFrameWnd::~CFrameWnd, XRF 338
- CGdiObject objects, CGdiObject::~CGdiObject, XRF 345
- CMapStringToOb objects, CMapStringToOb::~CMapStringToOb, XRF 379
- CMenu objects, CMenu::~CMenu, XRF 421
- CObject objects, CObject::~CObject, XRF 467
- CObject pointer arrays, CObArray::~CObArray, XRF 454
- CObject pointer lists, CObList::~CObList, XRF 483
- CPaintDC objects, CPaintDC::~CPaintDC, XRF 499
- CStdioFile objects, CStdioFile::~CStdioFile, XRF 569
- CString objects, CString::~CString, XRF 580
- CWindowDC objects, CWindowDC::~CWindowDC, XRF 642
- CWnd objects
  - called to inform, CWnd::OnDestroy, XRF 730
  - destructor, CWnd::~CWnd, XRF 673
- fstream objects, fstream::~fstream, XRF 841

Destroying (*continued*)
- ifstream objects, ifstream::~ifstream, XRF 849
- iostream objects, iostream::~iostream, XRF 873
- Iostream_init objects, Iostream_init::~Iostream_init, XRF 874
- istream objects, istream::~istream, XRF 882
- istream_withassign objects, istream_withassign::~istream_withassign, XRF 888
- istrstream objects, istrstream::~istrstream, XRF 891
- list or combo box, called to inform owner, CWnd::OnDeleteItem, XRF 729–730
- MDI child windows, CMDIChildWnd::MDIDestroy, XRF 399
- menus, CMenu::DestroyMenu, XRF 423
- ofstream objects, ofstream::~ofstream, XRF 897
- ostream objects, ostream::~ostream, XRF 903
- ostream_withassign objects, ostream_withassign::~ostream_withassign, XRF 909
- ostrstream objects, ostrstream::~ostrstream, XRF 913
- stdiobuf objects, stdiobuf::~stdiobuf, XRF 916
- stdiostream objects, stdiostream::~stdiostream, XRF 918
- streambuf objects, streambuf::~streambuf, XRF 936
- strstream objects, strstream::~strstream, XRF 942
- strstreambuf objects, strstreambuf::~strstreambuf, XRF 946
- Windows windows, attached to CWnd, CWnd::DestroyWindow, XRF 675

DestroyMenu member function
- CMenu class, XRF 423
  - CWnd::SetMenu, XRF 810

DestroyWindow member function
- CWnd class, XRF 675
  - CDialog::Create, XRF 265
  - CDialog::CreateIndirect, XRF 267

Destruction, C++
- automatic objects, LR+ 150–151
- order, LR+ 308–310
- static objects, LR+ 154

Destructors
- array deallocation, TUT 99
- base classes, TUT 129
- called by delete operator, TUT 65, 96, 99, 105
- calling, LR+ 310–311
- CArchive, XRF 96
- CClientDC, XRF 138
- CDC, XRF 170
- CFile, XRF 308
- CFrameWnd, XRF 338
- CGdiObject, XRF 345
- classes requiring, TUT 69
- CMapStringToOb, XRF 379
- CMemFile, XRF 412
- CMenu, XRF 421
- CObArray, XRF 454
- CObject, XRF 467
- CObList, XRF 483
- CPaintDC, XRF 499
- CStdioFile, XRF 569
- CString, XRF 580
- CWnd, XRF 673
- declaring, LR+ 306–307
- described, LR+ 305–306
- filebuf, XRF 834
- fstream, XRF 841
- global objects, TUT 50
- ifstream, XRF 849
- ios, XRF 860
- iostream, XRF 873–874
- istream, XRF 882, 888–889
- istrstream, XRF 891
- ofstream, XRF 897
- ostream, XRF 903, 909
- ostrstream, XRF 913
- overview, TUT 43, 49, 69, 71
- static objects, TUT 50
- stdiobuf, XRF 916
- stdiostream, XRF 918
- streambuf, XRF 936
- strstream, XRF 942
- strstreambuf, XRF 946
- using, LR+ 307–308
- using C++ expressions, ET 410–411
- virtual, TUT 129

Detach member function
- CGdiObject class, XRF 347
- CMenu class, XRF 423
- CWnd class, XRF 676

Detaching
- Windows GDI object, CGdiObject::Detach, XRF 347
- Windows menu from CMenu object, CMenu::Detach, XRF 423

Device contexts
- bit pattern, creating, CDC::PatBlt, XRF 218–219
- brushes, retrieving origin of current, CDC::GetBrushOrg, XRF 193
- checking BitBlt support, XRF 169
- classes (list), XRF 6
- classes generally, XRF 18
- client areas
  - CClientDC class described, XRF 137
  - retrieving pointer to, CWnd::GetDC, XRF 688
- clipping region, specifying whether point is within, CDC::PtVisible, XRF 223
- copying bitmap to current, CDC::BitBlt, XRF 167, 169
- CPaintDC class described, XRF 498
- creating
  - CDC objects, CDC::CreateCompatibleDC, XRF 171
  - CDC::CreateDC, XRF 172–173
- CWindowDC class described, XRF 641
- CWnd, getting from, XUG 345
- graphic objects, Foundation classes cookbook, XUG 348
- informing of new print job, CDC::StartDoc, XRF 254
- metafile, closing and creating handle to play, CMetaFileDC::Close, XRF 440
- objects, CDC class described, XRF 156
- operations described, XRF 156
- palettes, selecting logical, CDC::SelectPalette, XRF 233
- pens, brushes, enumerating available, CDC::EnumObjects, XRF 182–184
- releasing, CWnd::ReleaseDC, XRF 801–802
- retrieving x- and y-coordinates
  - of associated window, CDC::GetWindowExt, XRF 209
  - viewport origin, CDC::GetViewOrg, XRF 209
  - windows origin, CDC::GetWindowOrg, XRF 210
- saving current state, CDC::SaveDC, XRF 227
- selecting GDI object into, XRF 18
- selecting object into, CDC::SelectObject, XRF 232–233
- setting viewport origins, CDC::SetViewportOrg, XRF 251

Device contexts (*continued*)
setting window origin, CDC::SetWindowOrg, XRF 253
text-alignment flags, retrieving status, CDC::GetTextAlign, XRF 205
translation origin, obtaining, CDC::GetDCOrg, XRF 196
viewports
retrieving x- and y-extents, CDC::GetViewportExt, XRF 209
setting x- and y-extents, CDC::SetViewportExt, XRF 250
Windows, restoring to previous state, CDC::RestoreDC, XRF 226
Devices
applications, allowing access to, CDC::Escape, XRF 184–185
contexts, creating, CDC::CreateDC, XRF 172–173
creating
information context for, CDC::CreateIC, XRF 173
memory device context, CDC::CreateCompatibleDC, XRF 171
DGROUP defined, ET 806
DH register, CodeView syntax, ET 419, 450
DI register
CodeView syntax, ET 419, 450
variables for version 7.0, LR 48
Diagnostic, assert function, LR 247
Diagnostic dump context described, XRF 24
Diagnostic dumping
and storing to archive, CTime::operators <<, >>, XRF 617
time spans, CTimeSpan::operators<<,>>, XRF 627
Diagnostic messages, Foundation classes tutorial, XUG 127, 138
Diagnostic output
stream-oriented, human readable, CDumpContext class described, XRF 273
Diagnostic reporting
defined
Foundation classes tutorial, XUG 21
Phone Book sample program, XUG 125
Diagnostic services
ASSERT macro, XRF 53
assertions, XRF 29
checking for corrupt guard bytes, AfxCheckMemory, XRF 47
described, XRF 28–29
executing specified iteration function for CObject-derived classes, AfxDoForAllClasses, XRF 48
executing specified iteration function for CObject-derived objects, AfxDoForAllObjects, XRF 49
forcing program halt on specified sequence numbers, AfxSetAllocStop, XRF 52
macros and global functions described, XRF 43, 46–49, 51–57
output, XRF 29
printing memory statistics report, CMemoryState::DumpStatistics, XRF 56
setting hook for memory allocation, AfxSetAllocHook, XRF 51
testing
ensuring memory block is contained in program's memory space, AfxIsValidAddress, XRF 51
memory address, AfxIsMemoryBlock, XRF 50
tracking memory, AfxEnableMemoryTracking, XRF 49
tuning allocation diagnostics, afxMemDF variable, XRF 46
Diagnostics
debugging, features of, Foundation classes cookbook, XUG 285–296
memory, XRF 29; XUG 291
Diagrams. *See* Charts; Graphs
Dialog box objects, constructing, CDialog::Create, XRF 264–265
Dialog boxes
About
command, QuickWin, customizing, PT 151, 157
Foundation classes tutorial, XUG 105–107, 110

Dialog boxes (*continued*)

- adding
  - HELLO sample program, XUG 105–107
  - Phone Book sample program, XUG 153–161
    - called to inform main window when entering idle state, CWnd::OnEnterIdle, XRF 737
- CEditDialog class, Foundation classes tutorial, XUG 164
- CFindDialog class, Foundation classes tutorial, XUG 163–164
- CModalDialog class, Foundation classes tutorial, XUG 106–107
- CodeView
  - getting help, ET 761
- COMMDLG.DLL file, XUG 163, 167
- construction and use, XRF 17
- control, returning pointer to specified, CWnd::GetDlgItem, XRF 689–690
- converting units of rectangle to screen units, CDialog::MapDialogRect, XRF 270
- creating
  - building resource, XRF 262
  - from resource template, XRF 261
- CWnd, called to cancel other modes, CWnd::OnCancelMode, XRF 717
- defined, ET 806; XRF 17
- defining in resource files, XRF 17
- derived controls, using in, Foundation classes cookbook, XUG 340
- deriving from CDialog class, XUG 332
- described, XRF 261
- dialog resources, XUG 166
- DoModal member function, Foundation classes cookbook, XUG 329
- focus control
  - CDialog::GotoDlgCtrl, XRF 268
  - CDialog::NextDlgCtrl, XRF 270
  - previous, CDialog::PrevDlgCtrl, XRF 272
  - Foundation classes cookbook, XUG 334–335
- fonts
  - for drawing text, CDialog::OnSetFont, XRF 271
  - setting on the fly, XRF 262
- HELPMAKE context prefix, ET 729
- invoking and returning result, CModalDialog::DoModal, XRF 447
- main window, using as
- message-checking, CDialog::IsDialogMessage, XRF 268–269
- message handlers, XUG 164

Dialog boxes (*continued*)

- modal
  - Cancel button action overriding, CModalDialog::OnCancel, XRF 448
  - CModalDialog class described, XRF 443–444
  - constructor, CModalDialog::CModalDialog, XRF 446
  - creating, XUG 329
  - creating indirect, CModalDialog::CreateIndirect, XRF 446–447
  - customizing, XUG 331
  - explained, XRF 17
  - Foundation classes tutorial, XUG 162, 165
  - initializing, XUG 330
  - invoking and returning result, CModalDialog::DoModal, XRF 447
  - OK button, overriding, CModalDialog::OnOK, XRF 448
  - terminating, CDialog::EndDialog, XRF 267
- modeless
  - constructor, CDialog::CDialog, XRF 264
  - creating, XRF 261
  - creating, CDialog::CreateIndirect, XRF 266–267
  - described, XRF 261
  - explained, XRF 17
  - Foundation classes tutorial, XUG 162, 165
- open, Foundation classes tutorial, XUG 153, 167
- overriding member functions, XRF 261
- Phone Book sample program
  - Foundation classes tutorial, XUG 151, 157, 162
  - HELLO, using as a template, XUG 152–153
- print, Foundation classes tutorial, XUG 153, 167
- push button control, getting ID, CDialog::GetDefID, XRF 268
- PWB
  - default key assignments, ET 150
  - displaying, ET 284–285, 288, 294
  - function, ET 80–82
  - getting help, ET 761
  - help, ET 758, 764
- retrieving associated caption or text, CWnd::GetDlgItemText, XRF 691
- save, Foundation classes tutorial, XUG 153, 167
- searching for previous or next control, CWnd::GetNextDlgGroupItem, XRF 693–694
- standard procedure, CDialog::OnInitDialog, XRF 271
- type-safe member functions, Foundation classes cookbook, XUG 333

Dialog boxes (*continued*)
  Windows programs, standard for, XUG 167
  translating text of specified control into integer value, CWnd::GetDlgItemInt, XRF 690–691
Dialog classes
  deriving from CDialog class Foundation classes cookbook, XUG 332
  message handler functions, Foundation classes tutorial, XUG 167
  message maps, Foundation classes tutorial, XUG 166
Dialog editors, dialog boxes, adding, XUG 156
Dialog objects, described, Foundation classes tutorial, XUG 165
Dialog resource files, Foundation classes tutorial, XUG 156
Dialog resource template, Foundation classes tutorial, XUG 224
Dialog resources, XUG 166
Dialog, windows, classes (list), XRF 6
Dictionaries, extended, suppressing, in LIB, ET 701
_dieeetomsbin function, LIB 175
Difference member function, CMemoryState class, memory leaks, detecting, XUG 292
difftime function, LIB 176–177
Digits, predefined expression syntax, ET 778, 780, 785
DIR command, PWB, replacing text, ET 94–95
Dir member function
  CComboBox class, XRF 147–148
  CListBox class, XRF 360
Directives
  dot, NMAKE, ET 687–688
  #endif, XUG 24
  error, C++. *See* Error directives, C++
  extern "C", Phone Book sample program, XUG 153, 167
  #ifdef, DMTEST sample program, XUG 49
  #include, DMTEST sample program, XUG 49
  inline assembly
    limitations, PT 113
    using in, PT 115–116
  pragma, C++. *See* Pragma directives, C++
  preprocessing, NMAKE, ET 688, 690–692
Directives (*continued*)
  preprocessor, C++. *See* Preprocessor directives, C++
Directories
  adding to list boxes, CListBox::Dir, XRF 360
  build, Foundation classes tutorial, XUG 9
  creating, _mkdir function, LIB 516–517
  current
    changing, _chdir function, LIB 140–141
    getting attributes, _dos_getfileattr function, LIB 202–203
    getting path names, _getdcwd function, LIB 356–358
    getting, _getcwd function, LIB 354–355
  ignoring, CL option, ET 545
  listing C files, PWB, ET 94
  putting
    in combo boxes, CWnd::DlgDirListComboBox, XRF 678–679
    in list boxes, CWnd::DlgDirList, XRF 676–677
  removing, _rmdir function, LIB 629–630
  renaming, rename function, LIB 625–626
  searching, CL options, ET 525–526
  setting attributes, _dos_setfileattr function, LIB 222–223
  subdirectory conventions, LIB 9
Directory-control routines, LIB 20
_disable function, LIB 178
Disable Mouse in CodeView option, ET 342
Disabling
  breakpoints, CodeView, ET 367, 427–428
  memory diagnostics, Foundation classes cookbook, XUG 291
  mouse, CodeView option, ET 342
  optimizations
    all, PT 9
    loop, unsafe, PT 20
  p-code quoting, PT 46
  stack checking, PT 21
Disabling interrupts, _disable function, LIB 178
Disassembling defined, ET 806, 818
DISCARDABLE keyword, module-definition files, ET 620

Disk
  drives, getting current
    _dos_getdrive function, LIB 200–201
    _getdrive function, LIB 359
  services, calling BIOS, _bios_disk function, LIB 110–113
Disk file input, ifstream class described, XRF 845
Disks, getting information, _dos_getdiskfree function, LIB 198–199
Display
  CodeView
    arranging, ET 327
    black-and-white display, ET 339
    line-display mode, ET 339
    memory format, ET 356–357
    overview, ET 345–347
    redrawing, ET 479
    screen exchange, ET 341, 343, 371, 445–447, 479
    specifying, ET 338–339
    suppressing snow, ET 341
  contexts, retrieving for entire window, CWnd::GetWindowDC, XRF 703
  devices, returning information about, CDC::GetDeviceCaps, XRF 196, 201
  PWB
    height, ET 283
    specifying color, ET 271–273
    width, ET 306
  screen, PWB, ET 67–68
Display Expression command, CodeView, ET 424, 477–478
Display modules, listing, CodeView, ET 383
_displaycursor function, LIB 179–180
Displaying
  charts
    pie, _pg_chartpie function, LIB 558–559
    scatter, _pg_chartscatter functions, LIB 560–561
    single or multiseries, _pg_chart functions, LIB 555–557
  fonts, PT 197
  nongraphic control characters, LR 19
  windows
    HELLO sample program, XUG 93
Distribution disks, Foundation classes tutorial, XUG 7
div function, LIB 181–182
Dividing integers, div function, LIB 181–182
Division assignment operator (/=), table listing, LR 138
Division operator, C++
  binary-operator expressions, LR+ 103–104
  overloading, LR+ 359
DL register
  CodeView syntax, ET 419, 450
  fastcall functions, LR 170–171
DlgDirList member function, CWnd class, XRF 676–677
DlgDirListComboBox member function, CWnd class, XRF 678–679
DlgDirSelect member function, CWnd class, XRF 679–680
DlgDirSelectComboBox member function, CWnd class, XRF 680–681
.DLL files
  See also Dynamic-link library
  defined, ET 806
_DLL macro, C++, LR+ 375
Dlllibs switch, PWB, ET 310–311
_dmsbintoieee function, LIB 175
DMTEST sample program
  building, XUG 65
  CDump Context class, Foundation classes tutorial, XUG 36
  code listings, XUG 66–79
  CPerson class, XUG 22–28
  CPersonList object, designing, XUG 37
  CString class, described, XUG 31
  CTime class, described, XUG 31
  data object, designing, Foundation classes tutorial, XUG 22
  deserialization, XUG 44
  developing, XUG 18
  exception handling, XUG 61–65
  Foundation classes tutorial, XUG 6
  serialization, XUG 42, 50
  summary, XUG 65
  testing, XUG 49–64
  writing, overview, Foundation classes tutorial, XUG 21
do statements, C++
  described, LR+ 144
  iteration statements, LR+ 142–143
doallocate member function, streambuf class, XRF 923
.DOC files defined, ET 806
Documenting code, writing comments, LR 2

- Dollar sign ($)
  - end of line, regular expression syntax, ET 778, 780–781, 786
  - environment variables, NMAKE, ET 679
  - filename macros, NMAKE, ET 672–673
  - inline assembly, PT 122
  - literal characters, NMAKE, ET 653
  - match line end, regular expression syntax, ET 779
  - reference to tagged expressions, regular expression syntax, ET 780, 787
  - regular expressions, PWB, ET 93
  - user-defined macros, NMAKE, ET 669
- Domain errors, LR 248
- Dominance, C++, LR+ 273–274
- DoModal member function
  - CModalDialog class, XRF 447
    - Foundation classes cookbook, XUG 329
    - Foundation classes tutorial, XUG 163
- DoModal(), member of CModalDialog, XRF 447
- DOS
  - command shell, Foundation classes tutorial, XUG 9
  - compatibility, LIB xi
  - defined, LIB xv
  - device names, appending, ET 497–498
  - executable files, EXEHDR output, ET 632–633
  - help, getting, ET 768–769
  - interface routines
    - (list), LIB 56–57
    - described, LIB 58
  - managing memory, browser database, ET 733–734
  - NMAKE, using, XUG 13
  - overlays, LINK, ET 566
  - PWB, using, XUG 13
  - sample programs, running, XUG 13
  - system calls
    - _bdos function, LIB 101–102
    - _intdos function, LIB 433–434
    - _intdosx function, LIB 435–436
- DOS applications
  - defined, ET 806
  - module-definition files, ET 607–608
  - optimizing precautions, PT 28
- DOS applications (*continued*)
  - overlaid
    - LINK, ET 597–601
    - MOVE, ET 597–603
  - Windows applications, converting to. *See* QuickWin
- DOS Extender
  - defined, ET 806
  - described, LIB xi
- DOS Protected-Mode Interface server, memory management, CodeView, ET 336
- DOS redirection symbol (>), HELPMAKE syntax, ET 713
- DOS session defined, ET 806
- DOS Shell command
  - CodeView, ET 358–359
  - PWB, ET 72
    - predefined macros, ET 142–143
- DOS shell, creating, PWB, ET 214–215, 257–258
- _dos_allocmem function, LIB 183–184
- _dos_close function, LIB 185–186
- _dos_commit function, LIB 187–188
  - bypassing buffers, LR 260
- _dos_creat function, LIB 189–190
- _dos_creatnew function, LIB 189–190
- _doserrno variable, LIB 63–64
- DOS-extended defined, ET 806
- _dosexterr function, LIB 234–235
- _dos_find function, LIB 191–193
- _dos_findfirst function, LIB 191–193
- _dos_findnext function, LIB 191–193
- _dos_freemem function, LIB 194–195
- _dos_getdate function, LIB 196–197
- _dos_getdiskfree function, LIB 198–199
- _dos_getdrive function, LIB 200–201
- _dos_getfileattr function, LIB 202–203
- _dos_getftime function, LIB 204–206
- _dos_gettime function, LIB 207–208
- _dos_getvect function, LIB 209
- _dos_keep function, LIB 210–211
- _dos_open function, LIB 212–213
- _dos_read function, LIB 214–215
- _dos_setblock function, LIB 216–217

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

_dos_setdate function, LIB 218–219
_dos_setdrive function, LIB 220–221
_dos_setfileattr function, LIB 222–223
_dos_setftime function, LIB 224–226
_dos_settime function, LIB 227–228
_dos_setvect function, LIB 229–231
_dos_write function, LIB 232–233
Doslibs switch, PWB, ET 310–311
/DOSS option, LINK, ET 561, 578
/DOSSEG option, LINK, ET 561, 578
Dot commands
  HELPMAKE, ET 722–724
    translating, ET 713–714
Dot directives, NMAKE, ET 687–688
Double precision defined, ET 806
double type
  described, LR 99
  floating-point constants, LR 11
  portability guidelines, PT 273
  variables, declaring as, PT 127–129
double type, C++
  described, LR+ 50–51
  size, LR+ 51–52
do-while statements
  described, LR 154–155
  iterations, LR 155
  terminating, LR 152
Down function, PWB, ET 151, 167
DPMI defined, ET 806
DPMI server. *See* DOS Protected-Mode Interface server
DPtoLP member function, CDC class, XRF 175
Dragging
  defined, ET 807
  minimized CWnd, CWnd::OnQueryDragIcon, XRF 775
DrawFocusRect member function, CDC class, XRF 175–176
DrawIcon member function, CDC class, XRF 176
Drawing
  borders
    around rectangles, CDC::FrameRect, XRF 191
    around regions, CDC::FrameRgn, XRF 192
  chords, CDC::Chord, XRF 170–171
  ellipses
    CDC::Ellipse, XRF 179–180
    _ellipse functions, LIB 241–242
  elliptical arcs
    _arc functions, LIB 86–87
    CDC::Arc, XRF 165–166
Drawing (*continued*)
  formatted text in rectangle, CDC::DrawText, XRF 177–179
  functions (list), PT 189–191
  icons on CDC device, CDC::DrawIcon, XRF 176
  line segments, CDC::Polyline, XRF 222
  lines
    getting mode, _getwritemode function, LIB 392–393
    to points, _lineto functions, LIB 453–454
  lines, CDC::LineTo, XRF 214
  pie-shaped wedges, CDC::Pie, XRF 219–220
  polygons
    CDC::Polygon, XRF 221
    CDC::PolyPolygon, XRF 222
    _polygon functions, LIB 580–582
  preventing in invalid window area, CDC::ExcludeUpdateRgn, XRF 186
  rectangles
    CDC::Rectangle, XRF 224
    _rectangle functions, LIB 616–617
    style indicating focus, CDC::DrawFocusRect, XRF 175
    with rounded corners, CDC::RoundRect, XRF 226–227
  retrieving current mode, CDC::GetROP2, XRF 203
  setting mode, CDC::SetROP2, XRF 243–244
  text
    dimmed, CDC::GrayString, XRF 210–212
    setting font in dialog boxes, CDialog::OnSetFont, XRF 271
  wedge-shaped figures, _pie functions, LIB 577–579
DRAWITEMSTRUCT structure, XRF 81–83
  CWnd::OnDrawItem, XRF 732–733
DrawMenuBar member function
  CWnd class, XRF 681
    CMenu::AppendMenu, XRF 418
    CMenu::DeleteMenu, XRF 422
    CMenu::InsertMenu, XRF 430
    CMenu::ModifyMenu, XRF 433
    CMenu::RemoveMenu, XRF 434
DrawText member function, CDC class, XRF 177–179
Drives
  adding to list boxes, CListBox::Dir, XRF 360
  changing current, _chdir function, LIB 142–143
  default, setting, _dos_setdrive function, LIB 220–221
  getting current
    _dos_getdrive function, LIB 200–201
    _getdrive function, LIB 359

/DS option
  HELPMAKE, ET 714, 773
  LINK, ET 579
  PWB, ET 141
DS register
  based pointers, LR 81
  CodeView syntax, ET 419, 450
  LINK, ET 579
DS register, equal to SS, PT 71–74
/DSALLOC option, LINK, ET 579
/DSALLOCATE option, LINK, ET 579
/DT option, PWB, ET 141
/Du option, HELPMAKE, ET 714
Dump context class, afxDump object, XUG 36
Dump member function
  CObject class, XRF 469–470
    CDumpContext::operator<<, XRF 278–279
  CObject class
    CObject::Dump, XRF 469
    Foundation classes cookbook, XUG 286–287
    Foundation classes tutorial, XUG 35
  overriding, XUG 286
  serialize function differences, XUG 35
Dumping
  array of hexadecimal-formatted bytes, CDumpContext::HexDump, XRF 276
  defined, ET 807
  depth
    getting, CDumpContext::GetDepth, XRF 276
    setting, CDumpContext::SetDepth, XRF 277
  flushing data to file attached to dump context, CDumpContext::CDumpContext, XRF 275
  matching reported corrupted memory with contents, CMemoryState::DumpAllObjectsSince, XRF 55
  math registers, CodeView, ET 473–474
  memory
    CodeView, ET 438–439
  memory statistics, Foundation classes cookbook, XUG 293
  object contents, Foundation classes cookbook, XUG 286
  objects, XUG 294–295
Dumping (*continued*)
  objects to CObject objects, CObject::Dump, XRF 469–470
_dup function, LIB 236–238
_dup2 function, LIB 236–238
Duplicate member function, CFile class, XRF 309
Duplicating
  CFile object, CFile::Duplicate, XRF 309
  strings, _strdup functions, LIB 753–754
DW operator, CodeView, ET 405, 414–415
DX register
  changed, LR 58
  CodeView syntax, ET 419, 450
  fastcall functions, LR 170–171
DX:AX register
  fastcall functions, LR 170–171
  for 4-byte return values, LR 171
/DY option, LINK, ET 561, 579
Dynamic allocation, based data, PT 81–83
Dynamic allocation, C++
  failed, testing for, LR+ 321–323
  freeing memory, delete operator, LR+ 323–325
  new operator, LR+ 318–320
Dynamic address, viewing memory, CodeView, ET 357
Dynamic binding, TUT 124
Dynamic character strings, CString class described, XRF 28
Dynamic Data Exchange, debugging, ET 379–382
Dynamic links defined, ET 807
Dynamic memory allocation
  See also malloc function
  not part of language, LR 35
/DYNAMIC option, LINK, ET 561, 579, 601–602
Dynamic overlays, MOVE, ET 604–605
Dynamic-link libraries
  debugging p-code, ET 389–390
  default names, PWB switches, ET 310–311
  defined, ET 807
  EXEHDR output, ET 635
  initialization routine, debugging, ET 381–382
  LINK object files, ET 563
  listing modules, CodeView, ET 383, 463

Dynamic-link libraries (*continued*)
  loading symbolic information, CodeView, ET 342, 363–364
  module-definition files, LINK, ET 570, 613–614
  optimizing entry/exit codes, CL options, ET 515
  protected mode, specifying, ET 617
  real mode, specifying, ET 617
  searching module statements, ET 622
  specifying module-definition files, ET 612–613
  values, CodeView, ET 328
Dynamic-link library (DLL)
  exporting functions to Windows, LR 171
  exporting symbols to Windows, LR 58

E

:e command, HELPMAKE, ET 722
E command, CodeView, ET 422, 433, 453
/E option
  CL, ET 493
  CL, consistency rules, precompiled headers, PT 39
  HELPMAKE, ET 711–712
  LINK, ET 580; PT 27
  NMAKE, ET 648, 678, 680
  PWB, ET 142
e. context prefix, HELPMAKE, ET 729
e option, optimize pragma, PT 22
eatwhite member function, istream class, XRF 877
EAX register, CodeView syntax, ET 419, 450
eback member function, streambuf class, XRF 924
EBP register, CodeView syntax, ET 419, 450
ebuf member function, streambuf class, XRF 924
EBX register
  CodeView syntax, ET 419, 450
  variables for 32-bit compiler, LR 48
_ecvt function, LIB 239–240
ECX register
  argument passing, LR 171
  changed, LR 58
  CodeView syntax, ET 419, 450
__edata, LINK, ET 578
EDI register
  CodeView syntax, ET 419, 450
  variables for 32-bit compiler, LR 48
Edit Breakpoints
  command, CodeView, ET 364, 367
  dialog box, CodeView, ET 367
Edit control
  See also Multiple-line edit control
  characters, selecting range, CEdit::SetSel, XRF 301
  combo boxes, getting position of current selection, CComboBox::GetEditSel, XRF 149
  current selection
    getting starting, ending character positions, CEdit::GetSel, XRF 293
    replacing with text, CEdit::ReplaceSel, XRF 297
  determining if contents modified, CEdit::GetModify, XRF 292
  getting formatting rectangle, CEdit::GetRect, XRF 293
  lines' length, retrieving, CEdit::LineLength, XRF 295
  maximum text length, specifying, CEdit::LimitText, XRF 294
  modification flag setting, clearing, CEdit::SetModify, XRF 298
  multiple-line. *See* Multiple-line edit control
  password character, setting, removing, CEdit::SetPasswordChar, XRF 299
  pasting data to, CEdit::Paste, XRF 297
  undoing last operation, CEdit::Undo, XRF 302
Edit menu
  CodeView, ET 360
  PWB
    described, ET 73
    functions, ET 143
    predefined macros, ET 143
  QuickWin, PT 148–149
Edit operations, undoing, CEdit::CanUndo, XRF 285
Edit Project command, PWB, ET 74
Editing
  breakpoints, CodeView, ET 367
  CONFIG.SYS, PWB, ET 66
  files, Editreadonly switch, PWB, ET 275–276
  macros, PWB, ET 111
  Noedit function, PWB, ET 190
  projects, PWB, ET 47–49, 55
  repeat function, PWB, ET 205
  text, menu commands, PWB, ET 73
  tools, Phone Book sample program, XUG 156
Editor
  PIF, starting PWB, ET 66
  restricting global search, ET 767
Editor Settings command, PWB, ET 75

Editors, dialog. *See* Dialog editors
Editreadonly switch, PWB, ET 263, 275
EDX register
  argument passing, LR 171
  changed, LR 58
  CodeView syntax, ET 419, 450
Efficiency, program, increasing, PT 24–25
EGA defined, ET 807
egptr member function, streambuf class, XRF 924
/Ei option, BSCMAKE, ET 736
/El option, BSCMAKE, ET 736
ElementAt member function, CObArray class, XRF 454–455
#elif preprocessor directive, LR 190, 202–204
#elif preprocessor directive, C++, LR+ 379–383
Eliminating
  common subexpressions, PT 8, 23
  dead store, PT 8
  stack probes, PT 21
_ellipse function, LIB 241–242; PT 189
Ellipse member function, CDC class, XRF 179–180
Ellipses
  creating region
    CRgn::CreateEllipticRgn, XRF 541
    CRgn::CreateEllipticRgnIndirect, XRF 541
  drawing
    CDC::Ellipse, XRF 179–180
    _ellipse functions, LIB 241–242
_ellipse_w function, LIB 241–242; PT 189
_ellipse_wxy function, LIB 241–242; PT 189
Ellipsis (...)
  call tree, PWB, ET 101
  menu command, PWB, ET 80, 82, 126
Ellipsis notation (...)
  compiling errors, LR 262
  indicating variable number of arguments, LR 180
  terminating partial parameter list, LR 187
Elliptical arcs, drawing
  _arc functions, LIB 86–87
  CDC::Arc, XRF 165–166
else clauses, nesting, LR 159
!ELSE preprocessing directive, NMAKE, ET 689
#else preprocessor directive, LR 190, 202–204
#else preprocessor directive, C++, LR+ 379–383
else statements, C++
  selection statements, LR+ 138–139
!ELSEIF preprocessing directive
  NMAKE, ET 690
!ELSEIFDEF preprocessing directive, NMAKE, ET 690
!ELSEIFNDEF preprocessing directive, NMAKE, ET 690
/Em option, BSCMAKE, ET 736
Emacscdel function, PWB, ET 151, 167–168
Emacsnewl function, PWB, ET 151, 168
Embedded structures, LR 67
Embedding text strings, CL option, ET 544
_emit pseudoinstruction, PT 115
EMM386.EXE
  See also Getting Started
  CodeView, ET 336, 338
  defined, ET 807
EMM.386.SYS, CodeView, ET 336, 338
EMOEM.ASM, floating-point math libraries, modifying with, PT 139
Empty classes, declaring, LR+ 232
Empty member function, CString class, XRF 580
EmptyUndoBuffer member function
  CEdit class, XRF 290
EMS. *See Getting Started*; Expanded memory;
Memory Emulator floating-point math package
    options; CL, PT 134–135
  described, PT 131
  environment variable, NO87, PT 138
Emulator library, floating-point math, CL options, ET 509–511, 513–514
Emulators defined, ET 807
_enable function, LIB 243
Enablealtgr switch, PWB, ET 263, 276
EnableMenuItem member function, CMenu class, XRF 423–424
EnableWindow member function, CWnd class, XRF 681–682

- Enabling
  - breakpoints, CodeView, ET 367, 428–429
  - interrupts, _enable function, LIB 243
  - memory diagnostics, Foundation classes cookbook, XUG 291
  - menu items, CMenu::EnableMenuItem, XRF 423–424
- Encapsulation
  - access to base classes, TUT 112
  - defined, TUT 165
  - design principle, TUT 165–169
  - Foundation classes tutorial, XUG 32
  - header files, TUT 61–62
  - member functions, TUT 54
  - Phone Book sample program, XUG 120, 134, 140
- Encoding, HELPMAKE options, ET 712–713, 727
- __end, LINK, ET 578
- .end command, HELPMAKE, ET 722
- END_CATCH macro, XRF 66
  - Foundation classes tutorial, XUG 51, 63
- EndDialog member function
  - CDialog class, XRF 267
    - Foundation classes tutorial, XUG 164
- EndDoc member function, CDC class, XRF 180–181
- Endfile function, PWB, ET 151, 168
- #endif directive, DMTEST sample program, XUG 24
- !ENDIF preprocessing directive, NMAKE, ET 690
- #endif preprocessor directive
  - described, LR 190, 202–204
  - testing code, LR 3
- #endif preprocessor directive, C++, LR+ 379–383
- #endif statement, Foundation classes tutorial, XUG 34
- END_MESSAGE_MAP macro
  - Foundation classes cookbook, XUG 313–315
  - Foundation classes tutorial, XUG 95
- Ending session, called when, CWnd::OnQueryEndSession, XRF 776
- Endline function, PWB, ET 151, 169
- END_MESSAGE_MAP macro
  - Foundation classes cookbook, XUG 313–315
  - Foundation classes tutorial, XUG 95
- End-of-file indicator, CTRL+Z, LR 4
- EndPage member function, CDC class, XRF 181–182
- EndPaint member function, CWnd class, XRF 682
- English word, predefined expression syntax, ET 778, 780, 785
- Enhanced graphics adapter, defined, ET 807
- Entab switch, PWB, ET 263, 276–277
  - white space, ET 127–128
- Enterinsmode switch, PWB, ET 263, 277–278
- Enterlogmode switch, PWB, ET 263, 278–279
- Enterselmode switch, PWB, ET 263, 278
- Entry codes
  - optimizing, ET 515
  - Windows functions
    - customizing, ET 515
    - generating, ET 522–523
- Entry points
  - p-code functions
    - described, PT 47
    - removing, PT 51
  - QuickBASIC, B_OnExit function, PT 242
- Entry tables
  - p-code, specifying maximum, PT 51–52
  - specifying, CL option, ET 521
- enum keyword, C++, TUT 19–20
  - declarations, LR+ 173–176
- enum type names, C++
  - introduction by declaration statements, LR+ 149
- Enumeration declarations
  - conversion, LR 147
  - described, LR 62–64
  - overview, LR 39
  - variables defined, LR 54
- Enumerations, C++, TUT 19–20
- Enumerators, C++
  - conversion by integral promotion, LR+ 177–178
  - definition, LR+ 177
  - described, LR+ 173–176
  - linkage, LR+ 35–36
  - names, LR+ 176
- EnumObjects member function, CDC class, XRF 182–184
- Envcursave switch, PWB, ET 137, 263, 279
- environ variable, LIB 66
- Environment
  - control functions, LIB 49–52
  - creating variables, _putenv function, LIB 597–599
  - function, PWB, ET 151, 169–170
  - strings defined, ET 807
  - table, getting value from, getenv function, LIB 360–361
  - tables
    - defined, ET 807
    - saving, in PWB, ET 279–280
  - time, setting, _tzset function, LIB 820–822

Environment variables
  See also Getting Started
  defined, ET 807
  HELPFILES
    defined, ET 809
    help file location, ET 771
    opening help files, ET 769
    restricting global search, ET 767
  INCLUDE
    BSCMAKE, ET 736
    Foundation classes tutorial, XUG 8
  INIT
    defined, ET 810
    remote debugging, ET 396
  LIB
    defined, ET 810
    Foundation classes tutorial, XUG 8
  LINK
    clearing, ET 594
    defined, ET 810
    setting, ET 593–594
  macros, NMAKE, ET 678
  menu commands, PWB, ET 75
  NO87, ET 513
  NO87, software emulator, PT 138
  PATH, installing CodeView, ET 327
  presentation graphics
    _axistype structures, PT 221–223
    _chartenv structures, PT 219–220, 226–227
    described, PT 219–220
    _legendtype structures, PT 225
    resetting, PT 219
    _titletype structures, PT 220–221
    _windowtype structures, PT 223–224
  PWB
    function, ET 169–170
    starting, ET 67
    TOOLS.INI file, ET 137
  specifying options, CL, ET 557–559
  SYSTEM defined, ET 817
  TEMP defined, ET 817
  TMP defined, ET 817

Environment Variables command, PWB, ET 75
Environment-processing, C++
  _setenv function, LR+ 41–42
Environments
  I/O portability guidelines, PT 291
  optimization precautions, PT 28
envp argument, C++
  main function syntax, LR+ 38
envp parameters, passing information to main, LR 31
Envprojsave switch, PWB, ET 137, 263, 280
_eof function, LIB 244–245
eof member function
  ios class, XRF 857
  ofstream class, iostream classes tutorial, XUG 377
/EP option, CL, ET 494
/Ep option, CL, consistency rules, precompiled headers, PT 39
epptr member function, streambuf class, XRF 925
Equal sign (=)
  module statement syntax, ET 610
  Redirect Input and Output command, CodeView, ET 424, 477
  substituting for number sign, ET 492
Equality operators, LR 132–133
Equality operators, C++
  binary-operator expressions, LR+ 107–110
  overloading, LR+ 358
EqualRect member function, CRect class, XRF 525
EqualRgn member function, CRgn class, XRF 546
EQUOTE p-code instruction, PT 46
/Er option, BSCMAKE, ET 736
ERESBOX.C sample graphics program, PT 168–169
_ERESCOLOR constant, PT 168–170
_ERESNOCOLOR constant, PT 170
.ERR files
  defined, ET 807
errno
  values and meanings (list), LIB 64
  variable, LIB 63–64
ErrnoToException data member, CFileException class, XRF 324–325

- Error bits
  - LINK, clearing, with EXEHDR, ET 631, 634–636
  - setting or clearing, ios::clear, XRF 856
  - testing if clear, ios::good, XRF 859
- Error checking, turning off, NMAKE, ET 661
- Error codes
  - CVPACK, ET 745
  - defined, ET 807
  - LIB, ET 708
  - LINK, ET 596
  - NMAKE, ET 696
    - from commands, ET 662–663
    - ignoring, ET 649, 687
  - SBRPACK, ET 741
  - Windows applications, optimizing, ET 515
  - Windows functions
    - customizing, ET 515
    - generating, ET 522–523
- Error directives, LR 208–209
- Error directives, C++, LR+ 385
- Error handler, C++, PT 106–107
- Error handling
  - critical conditions, _hard functions, LIB 402–405
  - math
    - _matherr and _matherrl functions, LIB 483–485
    - routines, LIB 13
  - stream I/O, LIB 13
  - transferring control to handler, _set_new_handler functions, LIB 672–675
  - using, LIB 12–13
- Error messages
  - *See also* "Error Messages," Part 2 of this book
  - getting, printing, strerror and _strerror functions, LIB 755–756
  - help, getting, ET 764
  - printing, perror function, LIB 547–548
- Error numbers, HELPMAKE context prefix, ET 729
- Error output, NMAKE, ET 650
- !ERROR preprocessing directive, NMAKE, ET 689
- #error preprocessor directive
  - described, LR 190
  - error messages, LR 208–209
- #error preprocessor directive, C++, LR+ 385
- Error stream, C++, TUT 7
- Error testing, I/O, ios::fail, XRF 857
- Errors
  - *See also* Error messages arrays, LR 94
  - block-scope function declarations, LR 266
  - building a PWB program, ET 46, 50
  - caused by interpretation of tokens, LR 4

- Errors (*continued*)
  - compiling, LR 213
    - __based used incorrectly, LR 258–259
    - old style function declarations with /Za, LR 262
  - converting run-time library values to CFileException values, CFileException::ErrnoToException, XRF 324
  - defined, ET 815
  - domain, LR 248
  - exceptions, Foundation classes tutorial, XUG 61
  - extraction, iostream classes tutorial, XUG 384
  - file position, LR 252
  - function parameters with auto attribute, LR 266
  - getting information, _dosexterr function, LIB 234–235
  - help, getting, ET 764
  - hex escape sequence, LR 266
  - I/O, testing for serious, ios::bad, XRF 855
  - illegal expression, LR 114
  - integer constants, LR 16
  - macro redefinition, LR 194
  - main function causes, LR 266
  - menu commands
    - PWB, ET 74
  - messages, LR 233–254
    - *See also* Error messages
  - operating system, CFileException::m_lOsError, XRF 328
  - processing, ofstream class member functions, XUG 377
  - recovering from, Foundation classes tutorial, XUG 62
  - resolving references, LINK, ET 569
  - simulating in CodeView, ET 385, 464–465
  - symbol redefinition, LR 37
  - taking address of register array, LR 266
  - testing on streams, ferror function, LIB 269–270
  - type mismatch, LR 263
- /Es option, BSCMAKE, ET 736
- ES register, CodeView syntax, ET 419, 450
- Escape member function, CDC class, XRF 184–185
- Escape sequences
  - CodeView expressions, ET 408
  - defined, ET 807
  - described, LR 18–19
  - hexadecimal, LR 266
  - in string literals, LR 21
  - string literals, LR 196
  - table listing, LR 18
  - unrecognized, LR 265
  - unsupported by preprocessor directives, LR 190

- Escapes, regular expression syntax, ET 778, 780–781, 786–787
- ESI register
  - CodeView syntax, ET 419, 450
  - variables, LR 48
- ESP register, CodeView syntax, ET 419, 450
- eSpan operator, CTimeSpan class, XRF 626
- Eval entry, TOOLS.INI file
  - CodeView, ET 328, 330–331, 403–404
  - remote debugging, ET 393–395
- Evaluating
  - arguments
    - ASSERT macro, XRF 53
    - VERIFY macro, XRF 57
  - expressions, LR 4, 108, 155, 185
  - subscript expressions, LR 117–118
  - tokens, LR 4
- Evaluation order
  - effect of parentheses, LR 107
  - expressions, C++, LR+ 127–129
  - operators, C++, LR+ 11–14
  - portability guidelines, PT 289–290
- EVEN directive, inline assembler support, PT 113
- Event classes
  - designing, TUT 183, 185
  - hierarchy, TUT 193–195
  - relationships, TUT 186
- Event passing, TUT 193
- Event-handler functions, exporting, LR 172
- Examine Symbols command, CodeView, ET 423, 467–468
- Example programs. *See* Sample programs
- Exception classes. *See* Reference, exception classes
- Exception handlers
  - floating-point libraries, PT 139
  - predefined, Foundation classes cookbook, XUG 298
- Exception handling
  - described, XRF 30
  - DMTEST sample program, XUG 61–65
  - exception classes and macros described, XRF 30
  - when to use, XRF 30
- Exception object
  - CATCH macro, Foundation classes tutorial, XUG 63
  - passing as parameter, Foundation classes tutorial, XUG 64
- Exception processing, XRF 59, 67
  - AfxAbort, XRF 61
  - AfxSetTerminate, XRF 61
  - AfxTerminate, XRF 62
  - AfxThrowArchiveException, XRF 63
  - AfxThrowMemoryException, XRF 64
  - AfxThrowNotSupportedException, XRF 64
  - AfxThrowResourceException, XRF 64
  - AND_CATCH macro, XRF 65
  - CNotSupportedException class described, XRF 449
  - defining block of code for catching
    - additional exception types, AND_CATCH macro, XRF 65
    - first exception type, CATCH macro, XRF 65
  - END_CATCH macro, XRF 66
  - marking end of last CATCH block, END_CATCH macro, XRF 66
  - termination
    - default function, AfxAbort, XRF 61
    - fatal error, AfxTerminate, XRF 62
    - linking to specified function, AfxSetTerminate, XRF 61
  - THROW macro, XRF 66
  - THROW_LAST macro, XRF 67
  - throwing
    - CArchiveException, AfxThrowArchiveException, XRF 63
    - CFileException, AfxThrowFileException, XRF 63
    - CMemoryException, AfxThrowMemoryException, XRF 64
    - CNotSupportedException, AfxThrowNotSupportedException, XRF 64
    - CResourceException, AfxThrowResourceException, XRF 64
    - identifying code that might, TRY macro, XRF 67
    - rethrowing back to next outer CATCH block, THROW_LAST macro, XRF 67
    - specified exception, THROW macro, XRF 66
  - TRY macro, XRF 67

Exception-mask bits, 8087 command, CodeView, ET 473–474
Exceptions
  AND_CATCH macro, XUG 51, 63
  CATCH macro, XUG 51, 63, 298–302
  catching
    defined, XUG 62
    Foundation classes cookbook, XUG 298–299
  CException class described, XRF 303
  CFileException
    class described, XRF 322
    objects, creating, CFileException::CFileException, XRF 324
  CNotSupportedException class described, XRF 449
  constructors, Foundation classes cookbook, XUG 303
  contents, examining, XUG 300
  CString objects
    deallocating heap space, XUG 304
    described, XUG 304
  defined, sample programs, XUG 20
  DMTEST sample program, XUG 61–65
  END_CATCH macro, XUG 51, 63
  frame variables, Foundation classes cookbook, XUG 304
  frames, XUG 62–63
  global functions described, XRF 59–67
  macros described, XRF 59–67
  math errors. *See* "Error Messages," Part 2 of this book
  memory, CMemoryException objects, XRF 413
  memory leaks, avoiding, Foundation classes cookbook, XUG 304
  objects, freeing
    described, XUG 300
    handling locally, XUG 301
    throwing after destroying, XUG 301
  resources, CRecesourceException class described, XRF 536
  throwing
    CArchiveException, XRF 63
    CFileException::ThrowOsError, XRF 326
    CFileException, AfxThrowFileException, XRF 63
    CMemoryException, XRF 64
    CNotSupportedException, XRF 64
    CResourceException, XRF 64
    defined, XUG 62
    described, XUG 297
    from your own functions, XUG 302
    THROW macro, XUG 64
    THROW_LAST macro, XUG 64
Exceptions (*continued*)
  TRY macro, XUG 51, 63, 298–302
Exclamation point (!)
  command modifiers, NMAKE, ET 662
  HELPMAKE command, ET 719
  preprocessing directives, NMAKE, ET 688
  replacing text, PWB, ET 94
  Shell Escape command, CodeView, ET 423, 468–469
ExcludeClipRect member function, CDC class, XRF 185–186
ExcludeUpdateRgn member function, CDC class, XRF 186
Exclusive OR operator, bitwise, C++. *See* Bitwise exclusive OR operator, C++
EXE File Header Utility. *See* EXEHDR
.EXE files
  See also Executable (.EXE) files
  defined, ET 807
  overlaid DOS programs, ET 597
_exec function, LR 34; LIB 246–250
_execl function, LIB 246–250
_execle function, LIB 246–250
_execlp function, LIB 246–250
_execlpe function, LIB 246–250
Executable (.EXE) files
  adding to module-definition files, ET 614
  creating, CL, ET 486–487
  defined, ET 807
  EXEHDR output, ET 632–636
  format, ET 631
  LINK, ET 563
  packing, PT 27
  renaming, CL, ET 499
Executable program, formed by linking translation units, LR 190
.execute command, HELPMAKE, ET 722
Execute command, PWB, ET 74
Execute Commands option, CodeView, ET 340
Execute function, PWB, executing, ET 108, 151, 170
EXECUTEONLY keyword, module-definition files, ET 621
EXECUTEREAD keyword, module-definition files, ET 621
Executing
  8086 interrupts, LIB 429–432
  commands
    PWB, ET 78–82, 142, 219
    system function, LIB 803–804

Executing (*continued*)
DOS system calls
_intdos function, LIB 433–434
_intdosx function, LIB 435–436
functions, PWB, ET 106–108, 170
macros, PWB, ET 106–108
new child process, _spawn functions, LIB 717–722
Execution character set, LR 18
Execution, CodeView, controlling, ET 386
Execution model, CodeView, ET 333
Execution speed, improving, LR 192, 257
Execution time, optimizing, CL option, ET 539
_execv function, LIB 246–250
exefile field, LINK, ET 566
EXEHDR
application type, setting, ET 630
command line, ET 629–631
DLL output, ET 635
error bits, clearing, ET 631, 634–636
executable-file format, ET 631
exports tables, ET 636, 638
heap allocation, ET 630
Help, ET 630
memory allocation, ET 630
output
DOS executable files, ET 632–633
segmented executable files, ET 634–636
verbose output, ET 637–639
overview, ET 629
relocations, ET 639
segment tables, ET 635–636, 638
syntax, ET 629–631
/EXEPACK option
LINK, PT 27
debugging considerations, ET 325
described, ET 580
EXETYPE statement
module-definition files, ET 609, 615–616
segmented files, LINK, ET 564
EXIST operator, NMAKE, ET 690–691
Exit
atexit and _fatexit functions, LIB 96–97
QuickWin applications, _wsetexit function, LIB 889–891
registering routine to be called at, _fonexit and_onexit functions, LIB 531–532
Exit codes
CVPACK, ET 745
defined, ET 807
LIB, ET 708
LINK, ET 596
NMAKE, ET 696
from commands, ET 662–663
ignoring, ET 649, 687
SBRPACK, ET 741
Windows, ET 515
Exit command
CodeView, ET 358, 360
Phone Book sample program, Foundation classes tutorial, XUG 209–210, 216
PWB, ET 72, 142–143
QuickWin, PT 148
exit function, LIB 251–252; LR 30
exit function, C++
described, LR+ 42
initialization considerations, LR+ 44–45
Exit function, PWB, ET 151, 171
Exit processing, C++
atexit function, LR+ 45
Exit sequence, optimizing, CL option, ET 538
Exiting
See also Terminating
CodeView, ET 360
PWB, ET 47, 171, 251
QuickWin applications, LIB 871–872; PT 148, 162–163
ExitInstance member function, XRF 11, 631
EXP
command line, ET 750
options, ET 750
overview, ET 743, 747–748
syntax, ET 750

- exp function, LIB 253–254; PT 11
- _expand function, LIB 255–257
- Expanded memory
  - *See also Getting Started*; Memory
  - defined, ET 807
  - overlaid DOS programs, ET 598
- Expanded memory emulator, ET 808
- Expanded memory manager, ET 808
- Expanding
  - elements in CodeView, ET 367–368, 478–479
  - macro arguments, LR 31, 192–193, 196
- Expansion, inline, ET 532
- _expl function, LIB 253–254; PT 11
- Explicit allocation, ET 626, 808
- Explicit links, HELPMAKE, ET 718
- Explicit type conversions, C++
  - described, LR+ 121–124
  - expressions with
    - described, LR+ 119
  - operator, LR+ 119–121
- Explicitly allocated functions, ET 599
- Exponential functions, LIB 253–254, 583–584
- Exponents
  - C++ floating-point constants, LR+ 20
  - floating-point variables, PT 128
- Export functions
  - described, LR 58, 171–172
  - event-handler, LR 172
- __export keyword
  - described, LR 58
  - dynamic-link library, LR 171–172
  - entry/exit code
    - generating, ET 522–523
    - optimizing, ET 515
  - expanded functionality in version 7.0, LR 266
  - invalid with fastcall, LR 171
- __export keyword, C++
  - described, LR+ 419–420
- Export ordinals, searching for, ET 622
- EXPORTS statement, decorated names, ET 609, 623, 790–791
- Exports tables, EXEHDR output, ET 636, 638
- Expression evaluators, CodeView
  - choosing, ET 403–404, 454–455
  - defined, ET 399
  - listing, ET 370
  - numbers, ET 407–408
  - operators, ET 404–407
  - specifying, ET 331
  - string literals, ET 408
  - symbol format, ET 409
- Expression statements, C++, LR+ 136–137
- expression-list
  - converting, LR 185
  - evaluating, LR 185
- Expressions
  - address ranges, ET 402–403, 420–421
  - addresses, ET 401–402, 420
  - binding, LR 114
  - C++, in CodeView, ET 409–411
  - constant, defined, ET 805
  - defined, ET 808; LR 105
  - displaying, CodeView, ET 477–478
  - editing, CodeView, ET 351
  - evaluating, LR 4, 111
  - evaluation order, PT 289–290
  - floating-point, LR 10
  - function calls, LR 183–185
  - line number, ET 400, 418
  - live, creating, ET 357
  - l-value expressions, LR 108
  - MASM, use in inline assembly, PT 113
  - overview, CodeView, ET 399
  - parentheses, used in, LR 107
  - predefined. *See* Predefined expressions
  - preprocessing directives, NMAKE, ET 690–692
  - registers, ET 401, 419
  - regular. *See* Regular expressions
  - setting breakpoints, CodeView, ET 365
  - statements, LR 152–156
  - tagged. *See* Tagged expressions
  - watch. *See* Watch expressions
- Expressions, C++
  - binary-operator
    - additive operators, LR+ 104–106
    - assignment operators, LR+ 112–116
    - bitwise, LR+ 110–111
    - bitwise shift operators, LR+ 106–107
    - comma, LR+ 116–117
    - described, LR+ 102–103
    - equality operators, LR+ 107–110
    - logical operators, LR+ 111–112
    - multiplicative operators, LR+ 103–104
    - relational operators, LR+ 107–110
  - categories (list), LR+ 77–78
  - conditional-operator
    - described, LR+ 117–118
  - constant
    - described, LR+ 118–119
    - integral, conversion to null pointer, LR+ 75
  - defined, LR+ 77
  - evaluation order, LR+ 127–129

Expressions, C++ (*continued*)
explicit-type-conversion
described, LR+ 119–124
grammar summary, LR+ 424–427
gray expressions, LR+ 130
notation, LR+ 130–131
pointer conversions, LR+ 73–74
pointer-to-member-operator
described, LR+ 124–126
postfix
described, LR+ 81
primary, LR+ 78–80
sequence points, LR+ 129–130
unary-operator
address-of operator, LR+ 92–93
decrement operator, LR+ 94–95
delete operator, LR+ 101–102
described, LR+ 91
increment operator, LR+ 94–95
indirection operator, LR+ 92
logical NOT operator, LR+ 94
new operator, LR+ 97–101
one's complement operator, LR+ 94
sizeof operator, LR+ 95–96
unary negation operator, LR+ 93–94
unary plus operator, LR+ 93
Expunging files. *See* EXP
Extended ASCII, ET 808
Extended dictionaries
defined, ET 808
resolving references, LINK, ET 584
suppressing, in LIB, ET 701
Extended instructions, p-code (table), PT 297–299
Extended libraries, resolving references, LINK, ET 569
Extended memory
See also Getting Started; Memory; XMS
browser database, ET 733–737
defined, ET 808
Keepmem switch, PWB, ET 285
overlaid DOS programs, ET 598
Extended memory manager
CodeView, ET 336
defined, ET 808
Extending operations line, ET 702
Extension switches, PWB, ET 265
Extensions
autoloading, PWB, ET 131, 268
Curfileext predefined macro, PWB, ET 225
default
CL, ET 495
PWB, ET 274
defined, ET 808
IMPLIB, ET 746
language, CL options, ET 550–552
LINK, ET 565–567
loading, PWB, ET 286
Microsoft-specific, LR 108
processing CL, ET 486
specifying, CL, ET 496
extern "C", LR+ 37, 40–41, 179–181
directive, Phone Book sample program, XUG 153, 167
linkage specification, PT 257; TUT 23–24
extern "C++", LR+ 37, 179
extern keyword, C++
declaration statements, LR+ 152–154
declarations, use in, LR+ 158
described, LR+ 47
linkage specification, LR+ 181
extern storage-class specifier, LR 37, 43
External data, mixed-language programming, PT 265–266
External declarations, LR 37, 44–50
External identifiers, naming restrictions, LR 6
External linkage
C++
defined, LR+ 25–26
described, LR+ 33
identifiers, LR 37
External names, restricting length, CL option, ET 525

External references, ET 808
External variables, C++, LR+ 47
ExtFloodFill member function, CDC class, XRF 187–188
Extracting white space from streams, istream& ws, XRF 886
Extraction operators
  CArchive class, XRF 102
  CString class, XRF 593
  CTime class, XRF 617
  CTimeSpan class, XRF 627
  input streams, iostream classes tutorial, XUG 384
  istream class, XRF 885
  overloading, input streams, iostream classes tutorial, XUG 391
  testing for, iostream classes tutorial, XUG 384
ExtTextOut member function, CDC class, XRF 188–189

F

/f option, CL, ET 494–495; LR 260
\f (escape sequence), form feed, LR 18
/F option
  CL, ET 494, 527
  CodeView, ET 338, 341
  LINK, ET 580–581; PT 25–26
  NMAKE, ET 648; XUG 11
  RM, ET 748
F1 key
  HELLO sample program, XUG 93
  Help, getting, ET 757
/Fa option, CL, ET 495–497, 501–502
fabs function, LIB 258–259
_fabsl function, LIB 258–259
__fac floating-point accumulator, PT 130
Factor switch, PWB, ET 263, 280–281
fail member function
  ios class, XRF 857
  ofstream class, iostream classes tutorial, XUG 377
Far address defined, ET 808
Far calls
  LINK, ET 584, 587–588
  overlaid DOS programs, ET 603–604
Far function calls, translating to near calls, PT 25–26
far functions, PT 64, 66–68
  optimizing entry/exit codes, CL options, ET 515
  use, LIB 18
__far keyword, PT 64–68, 239–240
  /A options, CL, ET 489
  accepting with /Zf option, LR 261
__far keyword (*continued*)
  CL, accepting, ET 555
  conversions, LR 150
  data allocation, CL options, ET 523–524
  described, LR 56
  enabling, CL options, ET 550
  modifying items, LR 56, 61
  overriding addressing modes, LR 169
  referencing far objects, LR 169
  related to addressing, LR 55
  restrictions, LR 60, 169
  usage with __based keyword, LR 173
__far keyword, C++
  described, LR+ 7, 403
  this pointer modification, LR+ 246
Far objects, accessing, PT 57
Far pointers, PT 57
Far pointers, C++, LR+ 74
Far variables, declaring, PT 65–66
/FARCALL option, LINK, ET 580–581; PT 25–26
/FARCALLTRANSLATION option, LINK, ET 580–581; PT 25–26
Fast functions, PWB switches, ET 280
_FAST macro, C++, LR+ 375
Fastcall functions, registers used, LR 170–171
__fastcall keyword, PT 30–32
  calling conventions, LR 55; LR+ 417–418
    described, PT 30–32
    inline assembly limitations, PT 120–121
  CL
    calling conventions, ET 520
    enabling, ET 550
    naming conventions, ET 518
    symbol format, CodeView, ET 409
  described, LR 58
  specifying, LR 170
Fastfunc switch, PWB, ET 263, 281–282
FAT, defined, ET 808
Fatal errors
  See also "Error Messages," Part 2 of this book
  defined, ET 808
  simulating, CodeView, ET 385, 464–465
_fatexit function, LIB 96–97, LR 259
/FBr option, BSCMAKE, ET 732
/FBx option, BSCMAKE, ET 732
/Fc option, CL
  option interactions, ET 496–497
  output files, ET 495
  translating source code, ET 503–505

_fcalloc function, LIB 131–132
fclose function, LIB 260–261; PT 162
_fcloseall function, LIB 260–261; PT 162
_fcvt function, LIB 262–263
.FD files, ET 808
fd member function
  filebuf class, XRF 833
  fstream class, XRF 839
  ifstream class, XRF 847
  ofstream class, XRF 895
_fdopen function, LIB 264–266
/Fe option, CL, ET 494, 496, 499
feof function, LIB 267–268
ferror function, LIB 269–270
_fexpand function, LIB 255–257
FFLAGS options macro, NMAKE, ET 676
fflush function, LIB 271–272
_ffree function, LIB 306–308
_fgetchar function, LIB 273–274
fgetpos function, LIB 275–276
fgets function, LIB 273–274, 277–278
_fheapchk function, LIB 410–412
_fheapmin function, LIB 413–414
_fheapset function, LIB 415–417
_fheapwalk function, LIB 418–421
.FI files, ET 808
\fi formatting code, HELPMAKE, ET 726
_fieeetomsbin function, LIB 279
Fields
  BSCMAKE, ET 735
  LIB, specifying, ET 699–705
  LINK
    deffile, ET 570
    exefile, ET 566
    libraries, ET 567–570
    mapfile, ET 567
    objfiles, ET 565
    overview, ET 564–572
  SBRPACK, syntax, ET 740
File allocation table, ET 808
File buffering, LR 250
File classes described, XRF 26
File descriptors
  associated with streams, XRF 839, 847
  returning for filebuf object, filebuf::fd, XRF 833
  streams, returning, ofstream::fd, XRF 895
File Expunge Utility. *See* EXP
File handle defined, ET 808
File handles
  closing QuickWin window's, _wclose function, LIB 865–866
  creating, reassigning, _dup and _dup2 functions, LIB 236–238
  getting active QuickWin window, _wgetfocus function, LIB 873–874
  getting, _fileno function, LIB 282
  increasing maximum number, LIB 40–41
  low-level I/O (list), LIB 40
  predefined, LIB 40
File handling, member functions, Phone Book sample program, XUG 136
File Header Utility. *See* EXEHDR
File history
  maximum files, setting, ET 300
  PWB, ET 72
__FILE__ macro, C++, LR+ 375, 384
File menu
  CodeView, ET 358–360
  message handlers
    adding, XUG 205–212
    described, XUG 213–215
  Phone Book sample program, Foundation classes tutorial, XUG 206–216
  PWB
    described, ET 72
    predefined macros, ET 142
  QuickWin, PT 148
File operations, DMTEST sample program, XUG 60
File pointers
  defined, LIB 37
  moving, LIB 318–320, 471–473
  position, getting
    CFile::GetPosition, XRF 310
    ftell function, LIB 329–330
    _tell function, LIB 807–808

File pointers (*continued*)
  reassigning, freopen function, LIB 311–313
  repositioning, LIB 627–628; XRF 316–317
  setting value, XRF 317
File Removal Utility. *See* RM
File scope, C++
  described, LR+ 29
  linkage rules, LR+ 34
File sharing, opening stream with, _fsopen function, LIB 323–325
File streams, opening for QuickWin window, _fwopen function, LIB 335–337
File Undelete Utility. *See* UNDEL
__FILE__ macro, LR 199, 207
File scope, LR 35, 176
File-access permission, _access function, LIB 80–81
filebuf class
  consume defined, XRF 927
  described, XRF 831
  member functions
    attach, XRF 832
    close, XRF 832, 838, 846, 894
    fd, XRF 833
    filebuf, XRF 833
    ~filebuf, XRF 834
    is_open, XRF 834
    open, XRF 834–835
    setmode, XRF 835
filebuf constructor, XRF 833
filebuf destructor, XRF 834
filebuf objects
  attaching specified reserve area to stream's, ifstream::setbuf, XRF 850
  buffer associated with stream, returning pointer to, ifstream::rdbuf, XRF 850
  calling and closing associated file, fstream::close, XRF 838
  closing and disconnecting, ifstream::close, XRF 846
  closing connected file, filebuf::~filebuf, XRF 834
  connecting to specified open file, filebuf::attach, XRF 832
  constructor, ifstream::ifstream, XRF 847–848
  creating, filebuf::filebuf, XRF 833
  destroying, ifstream::~ifstream, XRF 849
  disconnecting file from and flushing, filebuf::close, XRF 832
  fstream constructor, fstream::fstream, XRF 839, 841
  opening disk file for stream, ifstream::open, XRF 849
  opening file and connecting, filebuf::open, XRF 834–835

filebuf objects (*continued*)
  reserve area, attaching, fstream::setbuf, XRF 843
  returning associated file descriptor, filebuf::fd, XRF 833
  setting binary/text mode, filebuf::setmode, XRF 835
  setting binary/text mode, fstream::setmode, XRF 844
  streams, XRF 894, 898–899
  testing for connection to open disk file, filebuf::is_open, XRF 834
FileDlg member function, adding, Foundation classes tutorial, XUG 238–239
File-handling routines, LIB 21
_filelength function, LIB 280–281
filename! command, HELPMAKE, ET 718
Filename extensions
  autoloading, PWB, ET 268
  Curfileext predefined macro, PWB, ET 225
  default
    CL, ET 495
    PWB, ET 274
  defined, ET 808
  IMPLIB, ET 746
  LINK, ET 565–567
  loading, PWB, ET 286
  processing, CL, ET 486
Filename macros, NMAKE, ET 672–673
Filename-extension tags, TOOLS.INI file, PWB, ET 132–133
Filename-Parts Syntax, PWB switches, ET 265–266
Filenames
  adding to list box of combo box, CComboBox::Dir, XRF 147
  alternate, setting, CL, ET 495
  base names
    Curfilenam predefined macro, PWB, ET 225
    defined, ET 804
    Shortnames switch, PWB, ET 296
  creating, LIB 518–520, 809–811
  defined, ET 808
  fully qualified, LR 200
  macros, NMAKE, ET 672–673
  operating system conventions, LIB 8–9
  path specification, CL, ET 496
  precompiled headers, ET 546
  predefined expression syntax, ET 778, 780, 785
  rules, LR 250
  specifying
    HELPMAKE, ET 712–714
    LINK, ET 566–567
    NMAKE, ET 648

- Filenames (*continued*)
  - syntax
    - LINK, ET 565
    - NMAKE, ET 663–664
- _fileno function, LIB 282
- File-permission settings, changing, _chmod, function, LIB 144–145
- File-position indicators, getting from streams, fgetpos function, LIB 275–276
- Files
  - accessing, LIB 80–81; LR 251
  - adding, PWB, ET 44, 48, 50
  - AFXWIN.H, Foundation classes tutorial, XUG 86, 203
  - assembly. *See* Assembly files
  - attributes, current, _dos_getfileattr function, LIB 202–203
  - backup, ET 95, 303, 747–750
  - batch, executing CL, ET 490
  - beginning, setting file pointers to, CFile::SeekToBegin, XRF 317
  - browser, generating from CL, ET 507–508
  - buffers, flushing, CFile::Flush, XRF 309
  - C++, translation order, LR+ 1–2
  - CFile class described, XRF 304
  - changing size, _chsize function, LIB 146–147
  - closing, ET 163
    - associated with CFile object, CFile::Close, XRF 308
    - _close function, LIB 156–157
    - _dos_close function, LIB 185–186
    - filebuf objects, filebuf::~filebuf, XRF 834
    - for I/O, LIB 40
    - Foundation classes cookbook, XUG 278
    - operating system, CFile::CFile, XRF 308
    - PWB, ET 72, 233
  - CodeView requirements, ET 328–329
  - command
    - defined, ET 805
    - NMAKE, ET 651
  - COMMDLG.DLL
    - dialog classes, XUG 163, 167
    - PrintDlg function, XUG 212

- Files (*continued*)
  - COMMDLG.H, Phone Book sample program, XUG 153, 167
  - compacting for CodeView, CVPACK, ET 743–744
  - compiling, PWB, ET 234
  - creating
    - CFile::CFile, XRF 306
    - _creat function, LIB 168–170
    - _dos_creat functions, LIB 189–190
    - PWB, ET 245
  - date and time written, _dos_getftime function, LIB 204–206
  - deleting
    - during debugging session, ET 359
    - PWB, ET 48
    - RM, ET 748–749
    - specified by filename, remove function, LIB 624
    - specified by path, CFile::Remove, XRF 315
    - specified by path, _unlink function, LIB 831–832
  - dialog resource, Foundation classes tutorial, XUG 156
  - directories
    - putting in combo boxes, CWnd::DlgDirListComboBox, XRF 678–679
    - putting in list boxes, CWnd::DlgDirList, XRF 678–677
  - disconnecting from filebuf object, filebuf::close, XRF 832
  - duplicating CFile object, CFile::Duplicate, XRF 309
  - editing, Editreadonly switch, PWB, ET 275
  - end of
    - setting file pointers to, CFile::SeekToEnd, XRF 317
    - testing, ios::eof, XRF 857
  - end-of-file testing, LIB 13
  - estimating size, PWB, ET 103–104
  - executable. *See* Executable (.EXE) files
  - expunging, ET 750
  - _fdopen function 264–266
  - finding
    - _dos_find functions, LIB 191–193
    - PWB, ET 72

- Files (*continued*)
  - flushing to disk
    - _commit function, LIB 158–159
    - COMMODE.OBJ, LIB 33, 37
    - _dos_commit function, LIB 187–188
  - font. *See* Font (.FON) files
  - _fopen function, LIB 290–292
  - handling routines, LIB 21
  - header. *See* Header (.H) files
  - Help. *See* Help files
  - icon. *See* Icon (.ICO) files
  - icon resource
    - Foundation classes tutorial, XUG 156
  - implementation
    - Foundation classes cookbook, XUG 282
    - Foundation classes tutorial, XUG 22
  - include. *See* Include files
  - #include, Foundation classes tutorial, XUG 22
  - increasing system limit, LIB 42
  - information about open, _fstat function, LIB 326–328
  - inline, NMAKE, ET 664–667
  - inserting, module statements, ET 627
  - interface, XUG 22
  - length
    - changing, CFile::SetLength, XRF 318
    - _filelength function, LIB 280–281
    - obtaining in bytes, CFile::GetLength, XRF 309
  - library. *See* Library files
  - linker output (.PXE). *See* LINK
  - listing, PWB, ET 94
  - loading, PWB, ET 142
  - locking bytes in, _locking function, LIB 460–462
  - locking range of bytes, CFile::LockRange, XRF 312
  - low-level I/O, reading and writing data, LIB 39
  - machine-code, creating listing, CL, ET 502–505
  - makefiles. *See* Makefiles
  - map
    - creating, CL, ET 505–507
    - LINK, ET 582–583
    - memory, XRF 412
  - memory
    - closing, CMemFile::~CMemFile, XRF 412
    - opening, CMemFile::CMemFile, XRF 412
  - module-definition. *See* Module-definition files
  - moving
    - PWB, ET 49
    - RM, ET 748–749
  - naming
    - Foundation classes tutorial, XUG 9
    - SBRPACK, ET 740

- Files (*continued*)
  - object. *See* Object files
  - open
    - testing for attachment to stream, XRF 842, 849
    - testing streams, ofstream::is_open, XRF 895
  - opening
    - described, LIB 39
    - _dos_open function, LIB 212–213
    - fopen function, LIB 290–292
    - for attachment to stream's filebuf object, fstream::open, XRF 842
    - for CFile objects, CFile::Open, XRF 313
    - for connection to filebuf objects, filebuf::open, XRF 834–835
    - for file sharing, _sopen function, LIB 714–716
    - Foundation classes cookbook, XUG 277
    - _open function, LIB 533–536
    - PWB, ET 72, 141, 191, 289–290
  - operating system, closing, CFile::~CFile, XRF 308
  - operating system handle, CFile::m_hFile, XRF 321
  - optimizing size, CL option, ET 538
  - output
    - alternate, CL, ET 495
    - LINK, ET 566
  - packing, SBRPACK, ET 739–740
  - pointers
    - getting current position, CFile::GetPosition, XRF 310
    - repositioning, CFile::Seek, XRF 316–317
    - setting value to beginning of file, CFile::SeekToBegin, XRF 317
    - setting value to logical end of file, CFile::SeekToEnd, XRF 317
  - precompiled header (.PCH). *See* Precompiled headers
  - preprocessing output, ET 540
  - printing
    - CodeView, ET 359
    - PWB, ET 194–195
  - project file list, PWB, ET 43–44
  - reading data from
    - _dos_open function, LIB 214–215
    - _read function, LIB 611–612
  - reading data into buffers, CFile::Read, XRF 314
  - reading from, Foundation classes cookbook, XUG 278
  - relocatable, LINK, ET 563
  - remote debugging, ET 393–395
  - removing libary name, CL option, ET 553–554
  - renaming, LIB 625–626; XRF 315
  - resource. *See* Resource files

Files (*continued*)
- resource include. *See* Resource include files
- resource script. *See* Resource script files
- response
  - BSCMAKE, ET 738
  - LINK, ET 573–575
- restoring, UNDEL, ET 749
- run-time stream, CStdioFile class described, XRF 567
- saving
  - Autosave switch, PWB, ET 269
  - PWB, ET 72, 209, 255–256, 300
- searching, PWB, ET 86–90
- searching for files using environment paths, _searchenv function, LIB 643–644
- segmented. *See* Segmented executable files
- setting
  - attributes, _dos_setfileattr function, LIB 222–223
  - modification time, _utime function, LIB 834–835
  - permission masks, _umask function, LIB 825–826
  - time, date, _dos_setftime function, LIB 224–226
  - translation mode, _setmode function, LIB 670–671
- source. *See* Source files
- specifying type, HELPMAKE, ET 713
- startup, modified, LIB 42
  - creating, tmpfile function, LIB 815–816
  - removing, _rmtmp function, LIB 631–632
- startup, PWB, ET 137
- state. *See* State file
- status
  - CFile object, CFile::GetStatus, XRF 310–311
  - getting, Foundation classes cookbook, XUG 279
  - PWB, ET 138–139
  - setting, CFile::SetStatus, XRF 318
- status information about, _stat function, LIB 734–735
- supporting, XUG 107–109
- temporary, LINK, ET 595
- testing for connection to open, filebuf::is_open, XRF 834
- testing for end-of-file, _eof function, LIB 244–245
- truncated, BSCMAKE, ET 733–735

Files (*continued*)
- unlocking range of bytes, CCFile::UnLockRange, XRF 319
- writing
- associated with CFile object, CFile::Write, XRF 320
  - buffers to, _dos_write function, LIB 232–233
  - data to, _write function, LIB 887–888
  - to, Foundation classes cookbook, XUG 278

Filetab switch, PWB, ET 127–128, 263, 282

Fill masks
- getting current, _getfillmask function, LIB 362–363
- setting, _setfillmask function, LIB 660–661

fill member function, ios class, XRF 857–858

Fill patterns
- graphics, functions (list), PT 191–192
- presentation graphics, PT 217–218

Filling display area with color, _floodfill and _floodfill_w functions, LIB 283–284

FillRect member function, CDC class, XRF 189–190

FillRgn member function, CDC class, XRF 190

Find command
- CodeView, ET 361
- PWB, ET 72–73, 87–90

Find Dialog box
- CodeView, ET 361

Find member function
- CObList class, XRF 483–484
- CString class, XRF 580–581

FindIndex member function
- CObList class, XRF 484

Finding
- characters
  - in buffers, memchr and _fmemchr functions, LIB 498–499
  - in strings, strchr and _fstrchr functions, LIB 740–742
- files, PWB, ET 72
- files with specified attributes, _dos_find functions, LIB 191–193
- first substring, strspn and _fstrspn functions, LIB 784–785
- fonts, _setfont function, LIB 662–664

Finding (*continued*)
  largest memory block size, _memmax function, LIB 508–509
  next token in string, strtok and _fstrtok functions, LIB 794–796
  relationships between classes, TUT 176
  strings in list boxes, CListBox::FindString, XRF 361
  substrings
    strcspn and _fstrcspn functions, LIB 749–750
    strstr and _fstrstr functions, LIB 786–787
  symbol definitions, PWB, ET 98–103
  text, in PWB, ET 91–93
    Mreplace function, ET 185
    Mreplaceall function, ET 185–186
    Qreplace function, ET 202
    Replace function, ET 205–207
FindOneOf member function, CString class, XRF 581
FindString member function
  CComboBox class, XRF 148
  CListBox class, XRF 361
FindWindow member function, CWnd class, XRF 683
FIXED keyword
  module-definition files, ET 621
Fixup, ET 808
/Fl option, CL, ET 496–496, 502–503
Flags
  8087 command, CodeView, ET 473–474
  buffer-deletion, assigning value for stream, ios::delbuf, XRF 856
  changing values, CodeView, ET 450–452
  _DEBUG, Foundation classes tutorial, XUG 25, 50, 203
  displaying value, CodeView, ET 354–355
  edit control undo, resetting, CEdit::EmptyUndoBuffer, XRF 290
  error-state, setting or clearing, ios::clear, XRF 856
  format
    clearing, ios::unsetf, XRF 864
    flag bits, defining, ios::bitalloc, XRF 855
  output file stream
    iostream classes tutorial, XUG 374–375
  setting specified format bits, ios::setf, XRF 863
  stream's internal variable, setting, ios::flags, XRF 858–859
  text-alignment
    retrieving status for device context, CDC::GetTextAlign, XRF 205
    specifying, CDC::SetTextAlign, XRF 246–247
  undo. *See* Undo Flags
flags member function, ios class, XRF 858–859
Flags register, ET 809
Flashing
  carets, CWnd::ShowCaret, XRF 816–817
  window, CWnd::FlashWindow, XRF 683–684
FlashWindow member function, CWnd class, XRF 683–684
Flat memory model
  defined, ET 809
Flipping
  screen exchange
    CodeView, ET 341, 445–447
    defined, ET 809
Float consistency
  CL options, ET 537–538
FLOAT.H header file
  restrictions, LR 11
float type
  portability guidelines, PT 273
  variables, declaring as, PT 127–129
float type, C++
  described, LR+ 50–51
  size, LR+ 51–52
floatfield data member
  ios class, XRF 867
Floating point
  arguments, calculating absolute value, fabs and _fabsl functions, LIB 258–259
  control word, getting and setting, _control87 function, LIB 160–162
  format flag bits, obtaining, ios::floatfield, XRF 867
  numbers
    converting between IEEE and Microsoft binary formats, _fieeetomsbin and _fmsbintoieee functions, LIB 279
    converting to strings, _fcvt function, LIB 262–263
    getting mantissa and exponent, frexp and _frexpl functions, LIB 314–315
  packages, resetting, _fpreset function, LIB 295–298
  precision variable, setting for stream, ios::precision, XRF 861, setprecision, 870
  remainders, calculating, fmod and _fmodl functions, LIB 288–289
  status word
    getting and clearing, _clear87 function, LIB 148–149
    getting, _status87 function, LIB 736–737
  support, LIB 14–15
  values
    converting to strings, _gcvt function, LIB 340–341
    splitting into mantissa and exponent, modf and _modfl functions, LIB 523–524

Floating types, C++
  conversion
    from integral, LR+ 69
    to integral, LR+ 69
    to other floating, LR+ 68–69
  described, LR+ 50–51
  limits, LR+ 63–64
Floating-point accumulator (__fac), PT 129
Floating-point constants, C++, LR+ 19–20
Floating-point math
  compatibility between, ET 512
  errors. *See* "Error Messages," Part 2 of this book
  intrinsic functions, generating, ET 534–535
  library selection, CL options, ET 509–514
  specifying, CL options, ET 508
Floating-point math functions
  intrinsic forms, PT 11
  long double type support, PT 130
Floating-point math libraries
  exception handler, PT 139
  linking, PT 137
  selecting, PT 132
  SETUP program, PT 130–131
Floating-point math packages
  alternate, PT 132, 136
  denormalized numbers, storing, PT 128–129
  emulator, PT 131, 134–135
  inline instructions, PT 137
  (list), PT 130
  math coprocessor, PT 131, 135–136
  optimization, effect, PT 133
  options, PT 132–136, 138
Floating-point numbers
  denormalized numbers, storing as, PT 128–129
  precision, increasing, PT 23–24
Floating-point types
  constants, LR 11–12
  conversions, LR 145–146
  described, LR 99
  functions that return, PT 130
  listed, LR 51
  promoting, PT 129
Floating-point types (*continued*)
  supported types (list), PT 127
  variables, declaring as, PT 127–129
Floating-point values
  converting, LR 145–146
  IEEE format, LR 99
  limits, LR 11–12
  preventing change of, LR 10
  representation, LR 240
  rounding, LR 149
  truncation, LR 241
Floating-point variables
  described, PT 128
  promoting, PT 129
_floodfill function, LIB 283–284; PT 191
FloodFill member function
  CDC class, XRF 190–191
_floodfill_w function, LIB 283–284; PT 191
floor function, LIB 285–286
  intrinsic form, PT 11
_floorl function, LIB 285–286
  intrinsic form, PT 11
Flow control
  statements, ET 112–114
FLT_DIG constant, C++
  floating limits, LR+ 63–64
FLT_EPSILON constant, C++
  floating limits, LR+ 63–64
FLT_MANT_DIG constant, C++
  floating limits, LR+ 63–64
FLT_MAX constant, C++
  floating limits, LR+ 63–64
FLT_MAX_10_EXP constant, C++
  floating limits, LR+ 63–64
FLT_MAX_EXP constant, C++
  floating limits, LR+ 63–64
FLT_MIN constant, C++
  floating limits, LR+ 63–64
FLT_MIN_10_EXP constant, C++
  floating limits, LR+ 63–64
FLT_MIN_EXP constant, C++
  floating limits, LR+ 63–64

FLT_RADIX constant, C++
  floating limits, LR+ 63–64
FLT_ROUNDS constant, C++
  floating limits, LR+ 63–64
Flush member function
  CArchive class, XRF 97
  CDumpContext class, XRF 275
  CFile class, XRF 309
  CFile::Flush, XRF 309
flush member function
  ostream class, XRF 902
_flushall function, LIB 287
Flushing
  buffers to dump context,
    CDumpContext::CDumpContext, XRF 275
  file buffers, CFile::Flush, XRF 309
  files to disks
    bypassing buffers, LR 260
    _commit function, LIB 158–159
    COMMODE.OBJ, LIB 33, 37
    _dos_commit function, LIB 187–188
    _fdopen function, LIB 264–292
    fopen function, LIB 290–292
  output buffers, ostream& flush, XRF 907
  stream buffers, ostream::flush, XRF 902
  streams
    fflush function, LIB 271–272
    _flushall function, LIB 287
/Fm option, CL
  mapfile, ET 505–507, 527
  option interactions, ET 496
  output file, ET 495
_fmalloc function, LIB 479–482
_fmblen function, LIB 487–488
_fmbstowcs function, LIB 489–490
_fmbtowc function, LIB 491–493
_fmemccpy function, LIB 496–497
_fmemchr function, LIB 498–499
_fmemcmp function, LIB 500–502
_fmemcpy function, LIB 503–505
_fmemicmp function, LIB 506–507
_fmemmove function, LIB 510–512
_fmemset function, LIB 513–514
fmod function, LIB 288–289; LR 248; PT 11
_fmode variable, LIB 64
_fmodl function, LIB 288–289; PT 11
_fmsbintoieee function, LIB 279
FmtLines member function
  CEdit class, XRF 290
/Fo option
  CL, ET 495, 498
Focus control of dialog boxes
  moving to specified control,
    CDialog::GotoDlgCtrl, XRF 268
  next, CDialog::NextDlgCtrl, XRF 270
  previous, CDialog::PrevDlgCtrl, XRF 272
.FON files. *See* Font (.FON) files
_fonexit function, LIB 531–532
  model-independent processing, LR 259
Font (.FON) files, PT 195
Fonts
  aspect-ratio filter, CDC::GetAspectRatioFilter,
    XRF 192
  called upon change, CWnd::OnFontChange,
    XRF 738–739
  CFont class described, XRF 329
  copying typeface name into buffer,
    CDC::GetTextFace, XRF 208
  creating, CFont::CFont, XRF 330
  current, retrieving, CWnd::GetFont, XRF 692
  dialog boxes, XRF 262, 271
  displaying, LIB 28–29
  document conventions, PT xviii–xix
  finding single, _setfont function, LIB 662–664
  freeing memory used by, _unregisterfonts function,
    LIB 833
  getting characteristics, _getfontinfo function,
    LIB 364
  getting width in pixels, _getgtextextent function,
    LIB 365
  graphics
    described, PT 193–195
    displaying, PT 197
    library, using, PT 195
    registering, PT 195–196
    sample program, PT 198–199
    setting, PT 196–197
    using effectively, PT 199–200
  initializing
    fonts graphics system, _registerfonts function,
      LIB 618
    LOGFONT-specified characteristics,
      CFont::CreatFontIndirect, XRF 334
    specified characteristics, CFont::CreateFont,
      XRF 330–333
  library, LIB xii
  mapper, altering, CDC::SetMapperFlags, XRF 241
  predefined, retrieving handle to,
    CGdiObject::CreateStockObject, XRF 345–346
  selecting, CMetaFileDC::SelectStockObject,
    XRF 442

Fonts (*continued*)
retrieving character widths, CDC::GetCharWidth, XRF 194
retrieving metrics for current, CDC::GetTextMetrics, XRF 208
returning pointer to CFont object, CFont::FromHandle, XRF 335
setting CWnd, CWnd::SetFont, XRF 809
fopen function, LIB 290–292
FOR command macro
NMAKE, ET 676
.FOR files
defined, ET 809
for statements
described, LR 156–157
iterations, LR 154
terminating, LR 152
for statements, C++
described, LR+ 145–146
iteration statements, LR+ 142–143
Foreign makefiles
in PWB, ET 61–63
Formal arguments, C++
defined, LR+ 21
scope, LR+ 33
Formal parameters. *See* Parameters
Format
commands
CodeView, ET 352–353, 417
conversion base, setting to 10, ios& dec, XRF 868
conversion base, setting to 16, ios& hex, XRF 868
conversion base, setting to 8, ios& oct, XRF 869
decorated names, ET 789
executable files, ET 631
flag bits, defining, ios::bitalloc, XRF 855
HELPMAKE
described, ET 716
QuickHelp, ET 716–724
rich text format, ET 725–727
memory
changing, ET 356–357
Format bits, setting, ios::setf, XRF 863
Format control
iostream classes tutorial, XUG 368–373
Format flags
clearing, ios::unsetf, XRF 864
streams, XRF 869–870
Formats
clipboard (list), XRF 86–87
clipboard, called for delayed rendering, XRF 781
Formatting
text, HELPMAKE topics, ET 721
Formatting attributes
HELPMAKE, ET 718
QuickHelp format, ET 721
Formatting codes
rich text format
HELPMAKE, ET 726
Formatting rectangles, edit control
getting, CEdit::GetRect, XRF 293
setting, CEdit::SetRect, XRF 299
Formfeed
escape sequence, LR 18
FORTRAN compiler
calling conventions, LR 55
enabling, ET 550
__fortran keyword, PT 29, 239–240, 244–245
CL, calling conventions, ET 516–518
mixed-language programming, PT 238–240,
modifying function names, LR 57, 243–246
NMAKE command macro, ET 676
NMAKE options macro, ET 676
restrictions, LR 57, 170–171, 266
specifying, LR 170
__fortran keyword, C++
calling convention, LR+ 418
described, LR+ 7
FORTRAN/Pascal calling convention, PT 29
Foundation class library
application design
Foundation classes cookbook, XUG 305–310
debug version
features, XUG 285

- Foundation class library (*continued*)
  - design philosophy, XRF 7–8
  - diagnostics
    - Foundation classes cookbook, XUG 285–296
  - dialogs and control windows
    - Foundation classes cookbook, XUG 329–340
  - exception handlers, predefined
    - Foundation classes cookbook, XUG 298
  - exceptions
    - Foundation classes cookbook, XUG 297–304
  - files and serialization
    - Foundation classes cookbook, XUG 277–285
  - general-purpose classes
    - Foundation classes cookbook, XUG 251–260
  - graphics
    - Foundation classes cookbook, XUG 343–348
  - introduction
    - general-purpose classes, XUG v
    - windows classes, XUG iv
  - memory leaks, detecting, XUG 290–295
  - release version
    - AssertValid member function, XUG 289
  - user input
    - Foundation classes cookbook, XUG 351–357
  - window management
    - Foundation classes cookbook, XUG 311–327
- Foundation classes
  - CArchive
    - Foundation classes cookbook, XUG 279
    - Foundation classes tutorial, XUG 20
  - CDialog
    - dialog boxes, modeless, XUG 162, 165
  - CFile
    - Foundation classes cookbook, XUG 277
    - Foundation classes tutorial, XUG 20
  - CFrameWnd
    - Foundation classes cookbook, XUG 311, 323, 325
    - Foundation classes tutorial, XUG 83
  - CMDIChildWnd
    - Foundation classes cookbook, XUG 311, 323, 325
  - CMDIFrameWnd
    - Foundation classes cookbook, XUG 311, 323, 325
  - CMenu
    - Foundation classes tutorial, XUG 119
  - CModalDialog
    - dialog boxes, modal, XUG 162, 165
    - Foundation classes tutorial, XUG 83
  - CObject
    - Foundation classes cookbook, XUG 263–267, 279
    - Foundation classes tutorial, XUG 19
- Foundation classes (*continued*)
  - CObList
    - Foundation classes tutorial, XUG 20, 127
  - collection classes described, XRF 26–27
  - collections
    - *See also* Collections
    - described, XUG 40
    - Foundation classes cookbook, XUG 269–276
    - predefined, XUG 270
  - control
    - deriving from, XUG 337–339
    - in dialog boxes, XUG 340
    - message handler functions, using, XUG 338
    - objects, creating, XUG 336
    - overriding, XUG 338
    - using, XUG 335
    - values, setting, XUG 339–340
  - CPaint DC
    - Foundation classes tutorial, XUG 83
  - CPersonList
    - Foundation classes tutorial, XUG 40
  - CRect
    - Foundation classes tutorial, XUG 83
  - CString
    - Foundation classes cookbook, XUG 256–261
    - Foundation classes tutorial, XUG 20, 31
  - CTime
    - Foundation classes cookbook, XUG 255–256
    - Foundation classes tutorial, XUG 20, 31
  - CWinApp
    - Foundation classes cookbook, XUG 307
    - Foundation classes tutorial, XUG 83, 86–87
  - CWnd
    - Phone Book sample program, XUG 165, 203
  - debugging
    - DEBUG_NEW macro, XUG 296
  - declaring
    - Foundation classes tutorial, XUG 22
  - DefWindowProc
    - Foundation classes tutorial, XUG 100
  - derived
    - overriding, XUG 307
  - deserialization
    - Foundation classes cookbook, XUG 285
  - device contexts, XUG 348
  - dialog classes
    - message maps, XUG 166
  - error messages. *See* "Error Messages," Part 2 of this book
  - exception handling, XUG 297

Foundation classes (*continued*)
files
closing, XUG 278
opening, XUG 277
reading from, XUG 278
status, getting, XUG 279
writing to, XUG 278
macros
ASSERT, XUG 288–289
CATCH, XUG 298–302
DEBUG_NEW, XUG 296
DECLARE_SERIAL, XUG 281
IMPLEMENT_SERIAL, XUG 282
TRACE, XUG 288
TRY, XUG 298, 300–302
VERIFY, XUG 289
message-maps, using, XUG 313
messages, handling, XUG 313–319
mouse
windows classes, creating, XUG 319
serialization
Foundation classes cookbook, XUG 279–285
tutorial
DMTEST sample program, XUG 17–65
HELLO sample program, XUG 81–114
PHBOOK sample program, database, XUG 117–140
PHBOOK sample program, dialog boxes, XUG 151–167
PHBOOK sample program, message handlers, XUG 197–242
using, XUG 5–14
windows
creating, XUG 311
dialog boxes, XUG 329–335
Foundation classes tutorial, XUG 92
keyboard events, XUG 356–357
messages, overriding, XUG 320
mouse clicks, XUG 351–352
mouse, tracking in, XUG 353–356
preregistered, XUG 92–93
registration, XUG 112
scrolling, XUG 322

Foundation classes (*continued*)
Windows applications
idle loop processing, XUG 309–310
initializing, XUG 307–308
resource file, XUG 310
writing, XUG 305–306
Windows classes
base classes, XUG 311
constructors for, XUG 312
icons, changing, XUG 326
registration, XUG 325
Windows graphics
Foundation classes cookbook, XUG 343–344
Windows tools equivalents
derivation, XRF 10
Foundation classes cookbook, XUG 346
general purpose, described, XRF 21–30
message processing, vs. native Windows, XRF 9
miscellaneous support classes described, XRF 28
polymorphism, XRF 10
special WinMain version, XRF 12
vs. native Windows program initialization, XRF 12
Foundation control classes
(list), Foundation classes cookbook, XUG 335
Foundation control objects
creating
Foundation classes cookbook, XUG 336
Foundation graphics
objects
Foundation classes cookbook, XUG 347–348
/Fp option
CL, ET 495, 546; LR 260; PT 35
/FP options; CL, PT 132–136
_FP_OFF function, LIB 293–294
/FPa option
CL, ET 508–509, 527
/FPc option
CL, ET 508–509, 527
/FPc87 option
CL, ET 508–510, 527
/FPi option
CL, ET 508–513, 527

- /FPi87 option
  - CL, ET 508–513, 527
- _fpreset function, LIB 295–298
- fprintf function, LIB 299–300
- _FP_SEG function, LIB 293–294
- fputc function, LIB 301–302
- _fputchar function, LIB 301–302
- fputs function, LIB 303
- /FR option
  - BSCMAKE, ET 732
  - CL, ET 495–496, 507–508
- /Fr option
  - BSCMAKE, ET 732
  - CL, ET 495–496, 507–508; PT 40
- Frame allocation
  - Foundation classes cookbook, XUG 251–254
- Frame sorting
  - CL option, ET 539
  - controlling p-code, PT 52
- Frame variables
  - exceptions
    - Foundation classes cookbook, XUG 304
- Frame windows
  - base classes
    - Foundation classes cookbook, XUG 311
  - CFrameWnd class described, XRF 336
  - changing
    - Foundation classes cookbook, XUG 324
  - child, getting, CFrameWnd::GetChildFrame, XRF 339
  - classes (list), XRF 6
  - creating
    - attaching, CFrameWnd::Create, XRF 338–339
    - constructor, CFrameWnd::CFrameWnd, XRF 338
    - MDI client window, CMDIFrameWnd::CreateClient, XRF 404
    - MDI constructor, CMDIFrameWnd::CMDIFrameWnd, XRF 403
  - destroying, CFrameWnd::~CFrameWnd, XRF 338
  - loading accelerator table, CFrameWnd::LoadAccelTable, XRF 340
  - MDI child windows, matching
    - Foundation classes cookbook, XUG 324
  - replacing menu of MDI, CMDIFrameWnd::MDISetMenu, XRF 408–409
  - returning active MDI child, CMDIFrameWnd::GetChildFrame, XRF 405
- FrameRect member function
  - CDC class, XRF 191
- FrameRgn member function
  - CDC class, XRF 192
- Frames defined, ET 809
- fread function, LIB 304–305
- _frealloc function, LIB 613–615
- free functions, LIB 306–308
- Free store
  - delete operator, PT 105–106
  - described, PT 103
  - error handler, PT 106–107
  - new operator, PT 103–105
- Free store exhaustion. *See* _set_new_handler
- _freect function, LIB 309–310
- FreeExtra member function
  - CObArray class, XRF 455
- .freeze command
  - HELPMAKE, ET 722
- freeze member function
  - strstreambuf class, XRF 944
- freopen function, LIB 311–313
- frexp function, LIB 314–315
- _frexpl function, LIB 314–315
- Friction switch, PWB, ET 264, 282–283
- Friend
  - classes
    - design issues, TUT 93, 177
    - overview, TUT 89–91, 93
  - functions, TUT 94
    - nested classes, LR+ 256–257
  - overview, TUT 89
- friend keyword, C++, LR+ 290–293
- friend specifier, C++, LR+ 167
- Friends, C++
  - access rules, LR+ 290–293
  - declaring, LR+ 293–295
  - defining in class declarations, LR+ 295
- FromHandle data member
  - CFont class, XRF 335
- FromHandle member function
  - CBitmap class, XRF 113
  - CBrush class, XRF 125
  - CGdiObject class, XRF 348
  - CPalette class, XRF 504
  - CPen class, XRF 511
  - CRgn class, XRF 546
  - CWnd class, XRF 684
- /Fs option, CL
  - described, ET 495, 501
  - option interactions, ET 496–497
- fscanf function, LIB 316–317
- fseek function, LIB 318–320
- fsetpos function, LIB 321–322
- _fsopen function, LIB 323–325

- _fstat function, LIB 326–328
- _fstrcat function, LIB 738–739
- _fstrchr function, LIB 740–742
- _fstrcmp function, LIB 743–745
- _fstrcpy function, LIB 747–748
- _fstrcspn function, LIB 749–750
- _fstrdup function, LIB 753–754
- fstream class
  - described, XRF 836–837
  - iostream classes tutorial, XUG 392–393
  - member functions
    - attach, XRF 838
    - close, XRF 838
    - fd, XRF 839
    - fstream, XRF 839–841
    - ~fstream, XRF 841
    - is_open, XRF 842
    - open, XRF 842
    - rdbuf, XRF 843
    - setbuf, XRF 843
    - setmode, XRF 844
- fstream constructor, XRF 839–841
- fstream destructor, XRF 841
- fstream objects, creating, fstream::fstream, XRF 839–841
- _fstricmp function, LIB 759–760
- _fstrlen function, LIB 761–762
- _fstrlwr function, LIB 763–764
- _fstrncat function, LIB 765–766
- _fstrncmp function, LIB 767–769
- _fstrncpy function, LIB 770–771
- _fstrnicmp function, LIB 772–773
- _fstrnset function, LIB 774–775
- _fstrpbrk function, LIB 776–777
- _fstrrchr function, LIB 778–779
- _fstrset function, LIB 782–783
- _fstrspn function, LIB 784–785
- _fstrstr function, LIB 786–787
- _fstrtok function, LIB 794–796
- _fstrupr function, LIB 797–798
- ftell function, LIB 329–330
- _ftime function, LIB 331–332
- Full build
  - building a database, BSCMAKE, ET 733, 735
- _fullpath function, LIB 333–334
- Full-screen application defined, ET 809
- Function arguments, C++
  - names, LR+ 283
- Function body
  - compound-statement, LR 168
  - described, LR 181
  - syntax, LR 168
- Function calls
  - conventions
    - C calling convention, PT 29
    - __fastcall calling convention, PT 30–32
    - FORTRAN/Pascal calling convention, PT 29
    - overview, PT 29
    - register calling convention, PT 30
  - conversions, LR 149–150
  - defined, ET 809
  - described, LR 119, 166, 183, 188
  - inline assembly
    - C, PT 122
    - C++, PT 123
  - mixed-language programming
    - described, PT 229–231
  - optimizing, PT 25–26
  - p-code, PT 47
  - recursive, LR 188, 211
  - replacing, CL option, ET 534–535
  - variable number of arguments, LR 187
- Function declarations
  - *See also* Declarations
  - definition, LR 165
  - described, LR 84–87
  - levels of scope, LR 28
  - obsolete forms, LR 166
  - overview, LR 28
  - placement, LR 44
  - type specifiers, LR 51–52
  - with storage-class specifiers, LR 50
- Function declarations in header files, LIB 7–8
- Function definitions, LR 166–176
- Function definitions, C++, LR+ 213–216

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

- Function Hierarchy command, PWB
  - described, ET 76
  - function, ET 145
- Function identifiers
  - as addresses, LR 106
- Function names, C++
  - introduction by declaration statements, LR+ 149
- Function parameters, C++
  - linkage, LR+ 35–36
- Function pointers
  - values, LR 119
- function pragma
  - described, LR 211
  - format, PT 11
  - precompiled header compilation, effect on, PT 41
  - using, PT 11
- function pragma, C++, LR+ 388
- Function prototypes, TUT 9–10
  - affects type of conversion, LR 149–150
  - described, LR 181–183
  - scope
    - rules, LR 36
- Function return values, LR 160–161
- Function scope
  - C++, LR+ 29
  - rules, LR 36
- Function specifiers, C++
  - inline, LR+ 159
  - virtual, LR+ 163
- Functionality
  - basic levels of
    - Foundation classes cookbook, XUG 264
- Function-call operator, C++
  - overloading, LR+ 361
  - postfix expressions, LR+ 83–88
- Function-level linking, enabling, PT 21
- Functions
  - *See also* Routines
  - addressing, LR 169
  - aliasing between functions, PT 13–18
  - allocated
    - overlaid DOS programs, ET 599
  - argument lists, variable, portability limitations, PT 289
  - arguments with side effects, portability guidelines, PT 290–291
  - attributes, LR 168–176
  - based addressing, PT 88–90
  - Bessel, LIB 103–105
  - BIOS interface (list), LIB 55
  - buffer-manipulation (list), LIB 18

- Functions (*continued*)
  - calling
    - CodeView expressions, ET 405
  - character classification and conversion (list), LIB 19
  - console and port I/O (list), LIB 43
  - data-conversion (list), LIB 20
  - declaring
    - __near and __far, PT 66–68
  - defined, ET 809; LIB 9
  - difference from macros, LIB 9–11
  - directory control (list), LIB 20
  - DOS interface (list), LIB 56–57
  - exporting, module-definition files, ET 623
  - far
    - optimizing entry/exit codes, ET 515
  - file-handling, LIB 21
  - floating-point math
    - long double type support, PT 130
  - floating-point types, returning
    - declaring, PT 130
  - graphics
    - analyzing presentation (list), LIB 30
    - configuring mode and environment (list), LIB 22–23
    - creating output (list), LIB 26–27
    - creating text output (list), LIB 27
    - displaying fonts (list), LIB 28–29
    - displaying presentation (list), LIB 29–30
    - low-level palette (list), LIB 25
    - low-level, character-font (list), LIB 22
    - presentation (list), LIB 29
    - presentation, manipulating structures (list), LIB 30–31
    - setting attributes (list), LIB 25
    - setting coordinates (list), LIB 23–24
    - transferring images (list), LIB 28
  - importing, module-definition files, ET 624–625
  - inline assembly
    - calling, C, PT 122
    - calling, C++, PT 123
    - versions, PT 118–119
  - inlining, PT 13
  - intrinsic
    - CL options, ET 534–535
    - generating, PT 10–12
  - I/O
    - (list), LIB 33, 35
    - predefined stream pointers (list), LIB 36
  - internationalization (list), LIB 44
  - listing
    - CodeView, ET 435–436

Functions (*continued*)
- low-level I/O (list), LIB 38–39
- math, LIB 44–46
- memory allocation (list), LIB 46–47
- ordering, module-definition files, ET 626–627
- QuickWin (list), LIB 53
- packaged
  - CL options, ET 524
  - overlaid DOS programs, ET 599
- p-code, native entry points
  - described, PT 47
  - removing, PT 51
- pointers. *See* Pointers
- process and environment (list), LIB 50–51
- prototypes
  - listing, Cl option, ET 552–553
- PWB
  - Arg, ET 94, 106–108
  - Assign, ET 108, 121–122, 124
  - Backtab, ET 127–128
  - call tree, ET 99–101
  - closing, ET 220
  - cursor-movement commands, ET 154–156
  - executing, ET 106–108
  - functions, ET 150–154, 170–221
  - Linsert, ET 108
  - listing references, ET 102
  - mark, ET 86
  - menu commands, ET 142, 144–146
  - Meta, ET 107–108
  - Mgrep, ET 87
  - modifying, ET 181
  - Msearch, ET 87
  - Paste, ET 94, 108
  - Prompt, ET 116–117
  - Psearch, ET 87, 107
  - tabs, ET 127–129
  - Tell, ET 108
- requiring floating-point support (list), LIB 14
- _spawn and _exec forms (list), LIB 52
- stack checking (list), LIB 12
- string manipulation (list), LIB 54–55

Functions (*continued*)
- tracing
  - CodeView, ET 452–453
- time
  - current (list), LIB 58–59
  - variables (list), LIB 62
- unreferenced, removing, PT 28
- using huge arrays with, LIB 16
- variable-length arguments list (list), LIB 59
- exporting, LR 171–172
- intrinsic, new, LR 258
- lifetime, LR 34–35, 43
- nesting, LR 44
- new in version 7.0, LR 259–260
- obsolete, LR 166
- overview, LR 165–166
- passed as arguments, LR 185
- return type, LR 51
- rules, LR 34
- specifying calling conventions, LR 169–170
- visibility of identifiers, LR 46
- WinMain
  - Foundation classes tutorial, XUG 89, 112
- writing
  - inline assembly, PT 118–119

Functions, C++
- *See also* Member functions
- based, LR+ 411–412
- conversion, LR+ 315–317
- declarators, LR+ 203–210
- friend, TUT 94
- inline
  - described, LR+ 159–163
  - macros, TUT 16
  - overview, TUT 15, 17
- message handler
  - dialog classes, XUG 167
  - Foundation classes tutorial, XUG 204
- overloaded, TUT 20–23
- overloading. *See* Overloading
- prototypes, LR+ 155
- types, LR+ 52

- Functions, C++ (*continued*)
  - virtual
    - abstract classes, LR+ 265–266
    - accessing, LR+ 296–297
    - described, LR+ 265, 275–279
- FUNCTIONS statement
  - module-definition files
    - decorated names, ET 790–791
  - module-definition files, ET 609, 626–627
  - overlaid DOS programs, ET 600–601
  - overlay number, LINK, ET 564, 598
- Fundamental types, C++
  - conversions. *See* Conversions, C++
  - described, LR+ 50–52
- _fwcstombs function, LIB 867–868
- _fwctomb function, LIB 869–870
- _fwopen function, LIB 335–337; PT 151, 157–158
- fwrite function, LIB 338–339

G

- :g command, HELPMAKE, ET 723
- G command, CodeView, ET 422, 433–434
- /G option
  - CodeView, ET 338, 341
- g option, optimize pragma, PT 23
- /G0 option
  - CL, ET 514–515; PT 24
- /G1 option
  - CL, ET 514–515; PT 24
- /G2 option
  - CL, ET 514–515; PT 24
- /G3 option
  - CL, ET 514–515; PT 24
- /G4 option
  - CL, ET 514–515; PT 24
- /GA option
  - CL, ET 515; LR 260
- _GBORDER constant, PT 168
- gbump member function
  - streambuf class, XRF 925
- /Gc option
  - CL, ET 516–518; LR 57–58, 170, 175, 235; PT 29
- gcount member function
  - istream class, XRF 877
- _gcvt function, LIB 340–341
- /Gd option
  - CL, ET 516–518; LR 170; PT 29
- /GD option
  - CL, ET 515; LR 260
- GDI
  - classes
    - described, XRF 18
    - list, XRF 6–7
  - device contexts
    - classes (list), XRF 6
    - specifying origin for next brush assignment, CDC::SetBrushOrg, XRF 239
  - drawing objects classes (list), XRF 6
  - object classes described, XRF 18
  - raster-operation codes (list), XRF 167–168
- GDI bitmaps
  - CBitmap class described, XRF 107–108
- GDI objects
  - attaching, CGdiObject::Attach, XRF 344
  - CGdiObject class described, XRF 342
- /GE option
  - CL, ET 515–516; LR 260
- /Ge option
  - CL, ET 518–520
- /GEa option
  - CL, ET 515
- /GEd option
  - CL, ET 515
- /GEe option
  - CL, ET 515
- /GEf option
  - CL, ET 515
- /GEm option
  - CL, ET 515
- /GEr option
  - CL, ET 515
- /GEs option
  - CL, ET 515
- Generating
  - code, selecting processor, PT 24
  - faster code, LR 173
  - form feeds, LR 213
  - in-line code for functions, LR 258
  - intrinsic functions, PT 10–12
  - pseudorandom number, rand function, LIB 609–610
- get areas
  - returning lower bound, streambuf::eback, XRF 924
  - returning number of characters available for fetching, streambuf::in_avail, XRF 926
  - returning pointer to byte after last, streambuf::egptr, XRF 924
  - setting pointer values, XRF 932

get member function
  input streams
    iostream classes tutorial, XUG 386–388
  istream class, XRF 878–879
Get pointers
  advancing after returning current character, streambuf::sbumpc, XRF 929
  following fetched characters, streambuf::sgetn, XRF 933
  getting value of, istream::tellg, XRF 884
  incrementing, streambuf::gbump, XRF 925
  moving back, streambuf::sputbackc, XRF 934
  moving forward one character, streambuf::stossc, XRF 935
  returning character at, streambuf::sgetc, XRF 933
  returning to next character to be fetched from streambuf, streambuf::gptr, XRF 925
  testing, streambuf::snextc, XRF 934
_getactivepage function, LIB 342–343
GetActiveWindow member function
  CWnd class, XRF 685
_getarcinfo function, LIB 344
GetAspectRatioFilter member function
  CDC class, XRF 192
GetAt member function
  CObArray class, XRF 455
  CObList class, XRF 485
  CString class, XRF 581–582
GetBitmapBits member function
  CBitmap class, XRF 113
  CGdiObject::GetObject, XRF 349
GetBitmapDimension member function
  CBitmap class, XRF 114
_getbkcolor function, LIB 345; PT 188
GetBkColor member function
  CDC class, XRF 193
GetBkMode member function
  CDC class, XRF 193
GetBrushOrg member function
  CDC class, XRF 193–194
GetBuffer member function
  CString class, XRF 582–583
  Foundation classes cookbook, XUG 260
GetBufferSetLength member function
  CString class, XRF 583–584
GetButtonStyle member function
  CButton class, XRF 131
getc function, LIB 346–347
GetCapture member function
  CWnd class, XRF 685
GetCaretPos member function
  CWnd class, XRF 686
_getch function, LIB 348–349
getchar function, LIB 346–347
GetCharWidth member function
  CDC class, XRF 194
_getche function, LIB 348–349
GetCheck member function
  CButton class, XRF 132
GetCheckedRadioButton member function
  CWnd class, XRF 686
GetChildFrame member function
  CFrameWnd class, XRF 339
  CMDIFrameWnd class, XRF 405
GetClientRect, XRF 687
GetClientRect member function
  CWnd class, XRF 686–687
GetClipboardOwner member function
  CWnd class, XRF 687
GetClipboardViewer member function
  CWnd class, XRF 687
GetClipBox member function
  CDC class, XRF 195
_getcolor function, LIB 350–351; PT 188
GetCount member function
  CComboBox class, XRF 149
  CListBox class, XRF 361
  CMapStringToOb class, XRF 380
  CObList class, XRF 485–486
GetCurrentMessage member function
  CWnd class, XRF 688
_getcurrentposition functions, LIB 352–353; PT 186, 189
GetCurrentPosition member function
  CDC class, XRF 195
_getcurrentposition_w function, PT 190

GetCurSel member function
  CComboBox class, XRF 149
  CListBox class, XRF 362
_getcwd function, LIB 354–355
GetDayofWeek member function, CTime class, XRF 641
GetDC member function
  CDC::DeleteDC, XRF 174
  CWnd class, XRF 688
GetDCOrg member function
  CDC class, XRF 196
_getdcwd function, LIB 356–358
GetDefID member function
  CDialog class, XRF 268
GetDepth member function
  CDumpContext class, XRF 276
GetDesktopWindow member function
  CWnd class, XRF 689
GetDeviceCaps member function
  CDC class, XRF 196–201
GetDlgCtrlID member function
  CWnd class, XRF 689
GetDlgItem member function
  CWnd class, XRF 689–690
GetDlgItemInt member function
  CWnd class, XRF 690–691
GetDlgItemText member function
  CWnd class, XRF 691
_getdrive function, LIB 359
GetEditSel member function
  CComboBox class, XRF 149
getenv function, LIB 360–361
GetFile member function
  CArchive class, XRF 97
_getfillmask function, LIB 362–363; PT 191
GetFocus member function
  CWnd class, XRF 692
GetFont member function
  CWnd class, XRF 692
_getfontinfo function, LIB 364; PT 197
_getgtextextent function, LIB 365
_getgtextvector function, LIB 366
GetHandle member function
  CEdit class, XRF 291
GetHead member function
  CObList class, XRF 486–487
GetHeadPosition member function
  CObList class, XRF 487
GetHorizontalExtent member function
  CListBox class, XRF 362
_getimage function, LIB 367–369; PT 192
_getimage_w function, LIB 367–369; PT 192
_getimage_wxy function, LIB 367–369; PT 192
GetItemData member function
  CComboBox class, XRF 150
  CListBox class, XRF 363
GetItemRect member function
  CListBox class, XRF 363
GetLastActivePopup member function
  CWnd class, XRF 692–693
GetLBText member function
  CComboBox class, XRF 150
GetLBTextLen member function
  CComboBox class, XRF 151
GetLength member function
  CFile class, XRF 309
  CString class, XRF 584
getline member function
  input streams
    iostream classes tutorial, XUG 388–389
GetLine member function
  CEdit class, XRF 291–292
  istream class, XRF 879–880
GetLineCount member function
  CEdit class, XRF 292
_getlinestyle function, LIB 370–371; PT 191
GetMapMode member function
  CDC class, XRF 201
GetMenu member function
  CWnd class, XRF 693
GetMenuItemCount member function
  CMenu class, XRF 424
GetMenuItemID member function
  CMenu class, XRF 425
GetMenuState member function
  CMenu class, XRF 425–426
GetMenuString member function
  CMenu class, XRF 427
GetModify member function
  CEdit class, XRF 292–293
GetNearestColor member function
  CDC class, XRF 202
GetNearestPaletteIndex member function
  CPalette class, XRF 505
GetNext member function
  CObList class, XRF 488–489
GetNextAssoc member function
  CMapStringToOb class, XRF 380–381
GetNextDlgGroupItem member function
  CWnd class, XRF 693–694
GetNextDlgTabItem member function
  CWnd class, XRF 694

GetNextWindow member function
CWnd class, XRF 695
GetObject member function
CGdiObject class, XRF 348–349
GetOpenFileName function
Foundation classes tutorial, XUG 163
GetPaletteEntries member function
CPalette class, XRF 505
CGdiObject::GetObject, XRF 349
GetParent member function
CWnd class, XRF 696
GetParentFrame member function
CFrameWnd class, XRF 340
CMDIChildWnd class, XRF 398
_getphyscoord function, PT 181, 187
_getpid function, LIB 373
_getpixel function, LIB 374–375; PT 190
GetPixel member function
CDC class, XRF 202–203
_getpixel_w function, LIB 374–375; PT 190
GetPolyFillMode member function
CDC class, XRF 203
GetPosition member function
CFile class, XRF 310
GetPrev member function
CObList class, XRF 489–490
GetRect member function
CEdit class, XRF 293
GetRgnBox member function
CRgn class, XRF 547
GetROP2 member function
CDC class, XRF 203
GetRuntimeClass member function
CObject class, XRF 470–471
gets function, LIB 376–377
GetSafeHandle member function
CGdiObject class, XRF 349
GetSafeHwnd member function
CWnd class, XRF 696
GetSaveFileName function
Foundation classes tutorial, XUG 163
GetScrollPos member function
CScrollBar class, XRF 555
CWnd class, XRF 696–697
GetScrollRange member function
CScrollBar class, XRF 556
CWnd class, XRF 697
GetSel member function
CEdit class, XRF 293
CListBox class, XRF 364
GetSelCount member function
CListBox class, XRF 364
GetSelItems member function
CListBox class, XRF 364–365
GetSize member function
CObArray class, XRF 456
GetStartPosition member function
CMapStringToOb class, XRF 381
GetState member function
CButton class, XRF 132–133
GetStatus member function
CFile class, XRF 310–312
GetStretchBltMode member function
CDC class, XRF 204
GetStyle member function
CWnd class, XRF 698
GetSubMenu member function
CMenu class, XRF 428
GetSuperWndProcAddr member function
CWnd class, XRF 698
GetSysModalWindow member function
CWnd class, XRF 698
GetSystemMenu member function
CWnd class, XRF 699
GetTabbedTextExtent member function
CDC class, XRF 204–205
GetTail member function
CObList class, XRF 490
GetTailPosition member function
CObList class, XRF 491
GetText member function
CListBox class, XRF 365

GetTextAlign member function
CDC class, XRF 205–206
GetTextCharacterExtra member function
CDC class, XRF 206
_gettextcolor function, LIB 378
GetTextColor member function
CDC class, XRF 207
_gettextcursor function, LIB 379
GetTextExtent member function
CDC class, XRF 207
GetTextFace member function
CDC class, XRF 208
GetTextLen member function
CListBox class, XRF 366
GetTextMetrics member function
CDC class, XRF 208
_gettextposition function, LIB 380–381
_gettextwindow function, LIB 382
Getting started. *See Getting Started*
Getting stream's position, XRF 904
GetTopIndex member function
CListBox class, XRF 366
GetTopWindow member function
CWnd class, XRF 700
GetTotalSeconds member function, CTimeSpan class, XRF 624
GetUpdateRect member function
CWnd class, XRF 700–701
GetUpdateRgn member function
CWnd class, XRF 701
GetUpperBound member function
CObArray class, XRF 456
_getvideoconfig function, LIB 383–385; PT 171–173, 187
_getviewcoord function, LIB 386–387; PT 181, 187
_getviewcoord_w function, LIB 386–387; PT 187
_getviewcoord_wxy function, LIB 386–387; PT 187
GetViewportExt member function
CDC class, XRF 209
GetViewportOrg member function
CDC class, XRF 209
_getvisualpage function, LIB 388
_getw function, LIB 389–390
GetWindow member function
CWnd class, XRF 702
_getwindowcoord function, LIB 391; PT 187
GetWindowDC member function
CWnd class, XRF 703
GetWindowExt member function
CDC class, XRF 209
GetWindowOrg member function
CDC class, XRF 210
GetWindowRect member function
CWnd class, XRF 703–704
GetWindowText member function
CWnd class, XRF 704
GetWindowTextLength member function
CWnd class, XRF 704–705
_getwritemode function, LIB 392–393
/Gi option
CL, LR 267
Gigabyte defined, ET 809
Global contexts
help files, linking, ET 719–720
Global functions
(list), XRF 34–35
AfxGetApp, XRF 36
AfxGetAppName, XRF 36
AfxGetInstanceHandle, XRF 36
AfxRegisterWndClass, XRF 37
diagnostic services, described, XRF 43–57
exception processing, XRF 59, 67
Global heaps
listing memory objects
CodeView, ET 382, 462–463
Global lifetime
determined by storage class, LR 43
identifiers, LR 35
Global memory handles
converting to pointers, ET 463–464
Global objects
Foundation classes tutorial, XUG 89, 111
Global register allocation
optimization option, PT 21–22
portability guidelines, PT 289
Global Search command, PWB, ET 78, 766–767
Global symbols
defined, ET 809
searching for
CodeView, ET 406–407
Global variables
afxMemDF, XRF 46
_amblksize, LIB 61–62
_cpumode, LIB 65
daytime, LIB 62
_doserrno, LIB 63–64
environment, LIB 66
errno, LIB 63–64
error codes, LIB 63–64
_fmode, LIB 64
locale macros, LIB 65

- Global variables (*continued*)
  - _osmajor, LIB 65
  - _osminor, LIB 65
  - _osmode, LIB 65
  - _osversion, LIB 65
  - _pgmptr, LIB 67
  - _psp, LIB 66–67
  - sys_errlist, LIB 63–64
  - sys_nerr, LIB 63–64
  - timezone, LIB 62
  - tzname, LIB 62
  - using, LIB 61
  - version of current operating system, LIB 14
- GlobalLock routine
  - locking memory handles, ET 386
- gmtime function, LIB 394–395
- /Gn option
  - CL, ET 520–521; LR 212, 260; PT 51
- Go command, CodeView, ET 422, 433–434
- good member function
  - ios class, XRF 859
  - ofstream class
    - iostream classes tutorial, XUG 377
- goodbit member function
  - ios class
    - ios::rdstate, XRF 862
- Goto command, PWB
  - predefined macros, ET 144
- Goto Definition command, PWB, ET 76
  - finding symbols, ET 98–99
  - function, ET 145
- Goto Error command, PWB, ET 74
- Goto Mark command, PWB, ET 73
- Goto Match command, PWB
  - described, ET 73
  - predefined macros, ET 144
- Goto Reference command, PWB, ET 76
  - function, ET 145
- goto statements
  - described, LR 157–158
  - inline assembly, PT 121–122
  - terminating for statements, LR 156
  - transferring control, LR 152
- goto statements, C++
  - jump statements, LR+ 149
  - labels, using with, LR+ 134–135
- GotoDlgCtrl member function
  - CDialog class, XRF 268
- /Gp option
  - CL, ET 521; LR 260; PT 51–52
- gptr member function
  - streambuf class, XRF 925
- /Gq option
  - CL, ET 521–522; LR 260
- /Gr option
  - CL, ET 520; LR 170, 175; PT 30
- Grammar summary, C++, LR+ 423–436
- Grandparent process defined, ET 809
- Graphic function, PWB, ET 151, 172
- Graphic objects
  - Foundation classes cookbook, XUG 346–348
- Graphics
  - adapters
    - (list), PT 167–168
    - terminate-and-stay-resident program requirements, PT 174
  - bounding rectangles, PT 185
  - character-font, using, LIB 22
  - colors
    - attributes, selecting, PT 175–176
  - coordinate systems
    - described, PT 180
    - physical coordinates, PT 180–182
    - viewport coordinates, PT 182–183
    - window coordinates, PT 184–185
  - displaying fonts, LIB 28–29
  - environment, configuring routines, LIB 22
  - error handling, LIB 13
  - fonts
    - described, PT 193–195
    - displaying, PT 197
    - library, using, PT 195
    - registering, PT 195–196
    - sample program, PT 198–199
    - setting, PT 196–197
    - using effectively, PT 199–200

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

- Graphics (*continued*)
  - function call status, returning most recent, _grstatus function, LIB 396–399
  - functions (lists)
    - animation, PT 192–193
    - color and palette control, PT 188–189
    - drawing, PT 189–191
    - pattern control, PT 191–192
    - video mode control, PT 186–188
  - getting
    - current fill masks, _getfillmask function, LIB 362–363
    - output position, _getcurrentposition functions, LIB 352–353
    - video configuration information, _getvideoconfig function, LIB 383–385
  - images
    - getting memory to store, _imagesize functions, LIB 426–427
    - storing in buffers, _getimage functions, LIB 367–369
  - image-transfer functions, LIB 28
  - library, PT 167
  - library, expanded, LIB xiii
  - low-level
    - palette routines, LIB 25
    - using, LIB 22
  - mode, configuring routines, LIB 22
  - modes
    - described, PT 167–168
    - selecting, PT 173
    - setting, PT 168–171
  - moving current positions, _moveto functions, LIB 527–528
  - output functions, LIB 26–27
  - palettes
    - CGA, PT 176–177
    - changing, PT 175–176
    - EGA, PT 179
    - MCGA, PT 179
    - Olivetti, PT 177
    - VGA, PT 177–179
  - pixel cursors, PT 186
  - presentation
    - *See also* Presentation graphics
    - analyzing charts, LIB 30
    - functions, LIB 29, 31
    - initializing, _pg_initchart function, LIB 570
    - manipulating structures, LIB 30–31
  - redefining viewports, _setviewport function, LIB 699–700
- Graphics (*continued*)
  - resolution
    - maximizing, PT 172
  - routines, LIB 22–31
  - sample programs
    - ERESBOX.C, PT 168–169
    - READVC.C, PT 171–172
    - SAMPLER.C, PT 198–199
    - YELLOW.C, PT 178
  - screen locations, PT 185
  - selecting palettes, _selectpalette function, LIB 647–649
  - setting
    - attributes, LIB 25–26
    - clipping region, _setcliprgn function, LIB 656–657
    - colors, LIB 25
    - coordinates, LIB 23–24
    - cursor toggle, _displaycursor function, LIB 179–180
  - Super VGA support, PT 173–175
  - text output routines, LIB 27–28
  - typefaces, PT 193–195
  - video configuration checking, PT 171–172
- Graphics Device Interface. *See* GDI.
- Graphics, Windows. *See* Windows graphics
- Graphs
  - *See also* Presentation graphics
  - axes
    - described, PT 204
    - structure types, PT 221–223
  - bar charts
    - described, PT 203
    - sample program, PT 208–212
    - styles, PT 204–205
  - categories, PT 202
  - chart windows, PT 204
  - column charts
    - described, PT 203
    - sample program, PT 208–212
    - styles, PT 204–205
  - data series
    - described, PT 202
  - data windows, PT 204
  - environment variables, PT 219–227
  - features described, PT 202–205
  - labels, PT 202
  - legends
    - described, PT 205
    - structure types, PT 225

- Graphs (*continued*)
  - line graphs
    - described, PT 203
    - sample program, PT 208–212
    - styles, PT 204–205
  - pie charts
    - described, PT 203
    - sample program, PT 206–208
    - styles, PT 204–205
  - scatter diagrams
    - described, PT 203–204
    - sample program, PT 212–214
    - styles, PT 204–205
  - styles
    - described, PT 204–205
    - pool, PT 216
  - types described, PT 202–205
  - values, PT 203
  - windows
    - chart, PT 204
    - data, PT 204
    - structure types, PT 223–224
- _GRAY constant, PT 180
- Gray, dark, color value, ET 273
- Gray expressions, C++, LR+ 130
- GRAYRECT structure
  - CStatic::Create, XRF 564
- GrayString member function
  - CDC class, XRF 210–212
- Greater than operator (>)
  - Redirect Input command, CodeView, ET 340, 424, 476
- Greater-than operator, C++
  - binary-operator expressions, LR+ 107–109
  - overloading, LR+ 359
- Greater-than-or-equal-to operator, C++
  - binary-operator expressions, LR+ 107–109
  - overloading, LR+ 359
- Green
  - color value, ET 273
- _GREEN constant, PT 180
- Group
  - defined, ET 809
- _grstatus function, LIB 396–399
- /Gs option
  - CL, ET 518–520; PT 21
- /Gt option
  - CL, ET 522; PT 75–76
- /GW option
  - CL, ET 522–523
- /Gw option
  - CL, ET 522–523; PT 40
- /Gx option
  - CL, ET 523–524; LR 260; PT 75–76
- /Gy option
  - CL, ET 524, 599; LR 260; PT 21
- /Gz option
  - CL, LR 170

H

- H command, CodeView, ET 422, 434
- .H files
  - *See also* Header (.H) files
  - defined, ET 809
- /H option
  - CL, ET 525; LR 6
  - CVPACK, ET 745
  - IMPLIB, ET 747
  - LIB, ET 701
- _halloc function, LIB 400–401
- Handlers
  - interrupt, LR 175
  - messages, XRF 69–75
  - symbol
    - specifying, ET 334–336
  - WM_COMMAND messages, XRF 69
- Handlers, C++
  - interrupt, LR+ 420–421
  - new, LR+ 321–323
- Handles
  - CClientDC objects, CClientDC::m_hWnd, XRF 138
  - GDI objects
    - attaching, CGdiObject::Attach, XRF 344
    - detaching, CGdiObject::Detach, XRF 347
  - operating system file, CFile::m_hFile, XRF 321

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

Handles (*continued*)
retrieving to stock Windows GDI objects, CGdiObject::CreateStockObject, XRF 345–346
returning to current instance of Windows application for accessing resources, AfxGetResourceHandle, XRF 37
current instance of Windows application, AfxGetInstanceHandle, XRF 36
specifying to Windows menu, CMenu::Attach, XRF 419
virtual memory, PT 91–92
Windows applications
current instance, CWinApp::m_hIns tance, XRF 639
previous instance, CWinApp::m_hPrevInstance, XRF 639
Windows GDI objects, attaching
CGdiObject::FromHandle, XRF 348
CGdiObject::GetSafeHandle, XRF 349
CGdiObject::m_hObject, XRF 351
Windows, detaching from CWnd object, CWnd::Detach, XRF 676
Handling errors. *See* Error handling
_hard functions, LIB 402–405
_harderr function, LIB 402–405
_hardresume function, LIB 402–405
_hardretn function, LIB 402–405
hdrstop pragma, PT 34–36
new in version 7.0, LR 258
placement, PT 37–38
syntax, PT 36–37
hdrstop pragma directive, C++, LR+ 388
/HE option
LINK, ET 581
/HEA option
EXEHDR, ET 630
Header (.H) files
described, LR 200
FLOAT.H, LR 11
graphics, PT 180, 196
LIMITS.H, LR 15, 99, 204
precompiled
CL options, ET 546–550; LR 261
consistency rules, PT 39–41
controlling, LR 211
creating, LR 266; PT 34–36
debugging, LR 266
debugging information, overriding CodeView, PT 38–39
described, PT 33
hdrstop pragma, PT 36–38
Header (.H) files (*continued*)
precompiled (*continued*)
new feature, LR 257
include path consistency, PT 40
options, PT 34–40
pragma consistency, PT 41
source file consistency, PT 41
using, PT 33–36
presentation graphics
palette structures, PT 214–215
STDARGS.H, LR 187
VARARGS.H, LR 187
Header files
contents, use, LIB 5
encapsulation, TUT 61–62
function declarations, LIB 7–8
including necessary definitions, LIB 6
suffix, TUT 60
unreferenced symbols
packing files, ET 739
using, LIB 6–8
Headers. *See* EXEHDR
Heap allocation, ET 630; XUG 252–254
/HEAP option
EXEHDR, ET 630
_heapadd function, LIB 406–409
_heapchk function, LIB 410–412
_heapmin function, LIB 413–414
Heaps
advantages of using based, LIB 49
allocating, _bheapseg function, LIB 107–109
based, PT 82
C++. *See* Free store
checking, _heapset functions, LIB 415–417
consistency checks, _heapchk functions, LIB 410–412
far
defined, LIB 48
routines, LIB 48
freeing, _bfreeseg function, LIB 106
debugging
_heapchk functions, LIB 410–412
_heapset functions, LIB 415–417
_heapwalk functions, LIB 418–421
global
listing memory objects, ET 382, 462–463
local
listing memory objects, ET 383
memory granularity variable, LIB 61
minimizing, _heapmin functions, LIB 413–414

Heaps (*continued*)
- near
  - defined, LIB 48
  - routines, LIB 48
- overlaid DOS programs, ET 598, 602
- size, specifying, ET 617–618

_heapset function, LIB 415–417

HEAPSIZE statement
- module-definition files, ET 609, 617–618

_heapwalk function, LIB 418–421

Height member function
- CRect class, XRF 525

Height switch, PWB, ET 264, 283

/HEL option
- EXEHDR, ET 630

HELLO sample program
- application class, XUG 87
- application object, XUG 87–89
- CFrameWnd class, XUG 83
- class hierarchies, XUG 89
- CModalDialog class, XUG 83
- code listings, XUG 83–107
- compiling, required files, XUG 109
- CPaintDC class, XUG 83
- CRect class, XUG 83
- CTheApp class, XUG 86
- CWinApp class, XUG 83, 86–87
- dialog boxes, adding
  - Foundation classes tutorial, XUG 105–107
- execution, sequence of, XUG 110–111
- F1 key, XUG 93
- files, supporting
  - Foundation classes tutorial, XUG 107–108
- Foundation classes cookbook, XUG 307, 310, 312
- NMAKE makefile, XUG 109
- OnPaint member function
  - sequence of steps in, XUG 104
- overview, XUG 6, 82
- PWB makefile, XUG 110
- template, using as, XUG 117, 152–153
- Windows, communication with, XUG 95–101

HELLO sample program (*continued*)
- windows
  - creating, XUG 90–93
  - painting text in, XUG 101–102
- writing
  - application class, XUG 82
  - application object, XUG 85–86
  - overview of steps, XUG 84
  - window class, XUG 82

Help
- *See also* CodeView; Help files; Microsoft Advisor; QuickHelp
- calling Microsoft. *See* the Product Assistance Request Form in LIB
- displaying in PWB, ET 198, 237, 239
- getting
  - CodeView, ET 756–765
  - HELPMAKE, ET 714
- index table
  - PWB, ET 239–240
- load state
  - PWB, ET 235
- next topic, PWB, ET 198–199, 236
- previous topic, PWB, ET 237
- QuickWin, PT 165
- searching, PWB, ET 199, 240
- structure, CodeView, ET 755
- switches, ET 313–315
- topic selection
  - PWB switch, ET 315
- topic, PWB, ET 238

Help command
- CodeView, ET 373–374, 422, 434
- PWB, ET 77

Help database
- compressing, ET 711–712
- context prefixes, ET 729
- creating, ET 711–712
- decoding, ET 713–714
- decompressing, ET 714
- overview, ET 710–711

Help delimiters (>>)
- HELPMAKE, ET 726, 728

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

- Help dialogs
  - CModalDialog object
    - Foundation classes tutorial, XUG 163
- Help File Maintenance Utility. *See* HELPMAKE
- Help files
  - closing
    - PWB, ET 213–214
    - QuickHelp, ET 769
  - concatenating, ET 772
  - creating, ET 711–712
  - decoding, ET 713–714
  - decompressing
    - specifying buffer size, ET 332
  - formats
    - described, ET 716
    - minimally formatted ASCII, ET 728
    - QuickHelp, ET 719–724
    - rich text format, ET 725–727
  - listing, ET 772
    - CodeView, ET 333
    - PWB switch, ET 314
  - locking, ET 713
  - managing, ET 771–773
  - opening
    - Microsoft Advisor, ET 765–766
    - PWB, ET 213–214
    - QuickHelp, ET 769
  - overview, ET 710–711
  - requirements
    - CodeView, ET 328–329
  - specifying, ET 713
  - splitting, ET 773
  - topics, defining, ET 716–717
- Help menu
  - CodeView, ET 374, 757
  - message handlers, adding
    - Foundation classes tutorial, XUG 222–223
  - PWB, ET 78, 146, 757
  - QuickWin, PT 151
- Help on Help command
  - CodeView, ET 374–375
  - PWB
    - described, ET 78, 758
    - predefined macros, ET 146
- /HELP option
  - BSCMAKE, ET 736
  - CL, ET 525
  - CVPACK, ET 745
  - EXEHDR, ET 630
  - EXP, ET 750
  - HELPMAKE, ET 715
- /HELP option (*continued*)
  - IMPLIB, ET 747
  - LIB, ET 701
  - LINK, ET 581
  - NMAKE, ET 648
  - RM, ET 748
  - SBRPACK, ET 740
  - UNDEL, ET 749
  - using, ET 768
- Help window
  - CodeView
    - function, ET 357–358
    - opening, ET 374
    - overview, ET 348
    - using, ET 760–761
  - PWB
    - default key assignments, ET 149
    - using, ET 760–761
  - setting size, ET 313–314
- Helpautosize switch, PWB, ET 313–314
- Helpbuffer entry
  - TOOLS.INI file
    - CodeView, ET 330, 332
- helpfile! contextstring command, HELPMAKE ET 718
- Helpfiles entry
  - TOOLS.INI file
    - CodeView, ET 330, 333
- HELPFILES environment variable
  - defined, ET 809
  - Help file location, ET 771
  - opening Help files, ET 769
  - restricting global search, ET 767
- Helpfiles switch, PWB, ET 314, 767
- Helplist switch, PWB, ET 315
- HELPMAKE
  - compatibility, ET 709
  - context prefixes, ET 729
  - decoding, ET 713–714
  - defining topics, ET 716–717
  - dot commands, ET 722–724
  - encoding, ET 711–713, 727
  - error messages. *See* "Error Messages," Part 2 of this book
  - formats
    - described, ET 716
    - minimally formatted ASCII, ET 728
    - QuickHelp, ET 716–724
    - rich text, ET 725–727
    - specifying, ET 713

HELPMAKE (*continued*)
  formatting attributes, ET 718–721
  formatting text, ET 721
  getting help, ET 714
  global contexts, ET 719–720
  local contexts, ET 720
  options
    decoding, ET 713–714
    encoding, ET 712–713
  overview, ET 710–711
  syntax, ET 715
    decoding, ET 713–714
    encoding, ET 711–712
    overview, ET 711
Helpwindow switch, PWB, ET 315
_HERCMONO constant, PT 170
hex member function
  ios class
    ios::bitalloc, XRF 855
Hexadecimal escape sequences, LR 14, 17–20
Hexadecimal numbers
  predefined expression syntax, ET 778, 780, 785
Hexadecimal defined, ET 809
HexDump member function
  CDumpContext class, XRF 276
hFileNull member function
  CFile class
    CFile::Close, XRF 308
_hfree function, LIB 422–423
/HI option
  LINK, ET 581
HideCaret member function
  CWnd class, XRF 705
Hiding
  carets, CWnd::HideCaret, XRF 705
  CWnd, called when, CWnd::OnShowWindow, XRF 783–784
  identifier names, LR 45
  list box of combo box, CComboBox::ShowDropDown, XRF 155
  names, LR+ 30–32
  scroll bars, CWnd::ShowScrollBar, XRF 817–818
Hierarchies
  abstract classes, TUT 128
  design issues, TUT 177–180
  identifying, TUT 174
  overview, TUT 113
High memory defined, ET 809
/HIGH option
  LINK, ET 581–582
Highlight defined, ET 809
Highlighting
  button control, getting, CButton::GetState, XRF 132
  setting, CButton::SetState, XRF 134
  search strings, in PWB, ET 210
  top-level menu items, CWnd::HiliteMenuItem, XRF 705–706
Hike switch, PWB, ET 264, 283–284
HiliteMenuItem member function
  CWnd class, XRF 705–706
HIMEM.SYS
  See also Getting Started
  CodeView, ET 336
  defined, ET 809
.HLP files
  defined, ET 809
Home function, PWB, ET 151, 172
Horizontal Scrollbars command, CodeView, ET 368, 370
Horizontal-tab escape sequence (\t), LR 18
Hours
  getting minutes in current, CTimeSpan::GetMinutes, XRF 623
  getting total, CTimeSpan::GetTotalHours, XRF 623
  getting, CTime::GetHour, XRF 613
  in current day, getting, CTimeSpan::GetHours, XRF 622
HPFS defined, ET 809
_HRES16COLOR constant, PT 170
_HRESBW constant, PT 170
Hscroll switch, PWB, ET 264, 284
Huge arrays
  arithmetic operations, LR 95

__huge keyword, PT 64–66
  CL, ET 489
  conversions, LR 150
  described, LR 57
  enabling, ET 550
  modifying objects, pointers LR 56
  overriding addressing modes, LR 169
  related to addressing, LR 55–57, 169
  restrictions, LR 169
  substitutes codes, LR 173
__huge keyword, C++
  described, LR+ 7, 404–405
  this pointer modification, LR+ 246
Huge memory models
  option, CL, PT 63
  described, ET 809; PT 60–61
Huge pointers, PT 57–58
Huge pointers, C++, LR+ 74
Huge variables, PT 65–66
Hyperlinks
  Microsoft Advisor
    index screens, ET 764
    navigating with, ET 759–761
_hypot function, LIB 424–425
Hypotenuses, calculating, _hypot and _hypotl functions, LIB 424–425
_hypotl function, LIB 424–425

I

I command, CodeView, ET 422, 434–435
:i command, HELPMAKE, ET 723
\i formatting attribute, HELPMAKE, ET 721, 726
/I option
  CL, ET 525–526; LR 201, 245
  CodeView, ET 338, 341–342
  LIB, ET 701
  NMAKE, ET 649
  RM, ET 748
I/O
  buffered disk file, filebuf class described, XRF 831
  called before insert operations, ostream::opfx, XRF 902
  CArchive objects, differences, XUG 33
  clearing format flags, ios::unsetf, XRF 864
  errors
    determining if error bits are set, ios::operator !(), XRF 866
    returning current specified error state, ios::rdstate, XRF 862

I/O (*continued*)
  errors (*continued*)
    testing for serious, ios::bad, XRF 855
    testing if error bits are clear, ios::good, XRF 859
    testing, ios::fail, XRF 857
  extracting from streams
    and discarding characters, istream::ignore, XRF 880
    bytes, istream::read, XRF 883
    data, istream::get, XRF 878, 880
    extraction operators, istream::operator>>, XRF 885
    white space, istream::eatwhite, XRF 877
  filebuf class described, XRF 831
  filebuf objects, XRF 838
  fill character, setting, setfill, XRF 869
  format flags
    clearing specified, resetiosflags, XRF 869
    setting, setiosflags, XRF 870
  fstream class described, XRF 836
  getting value of get pointer, istream::tellg, XRF 884
  insert operations, called after, ostream::osfx, XRF 902
  iostream class described, XRF 872, 874
  Iostream_init objects, XRF 874
  istream class described, XRF 875
  masks, padding flag bits, ios::adjustfield, XRF 867
  obtaining, flag bits, XRF 867
  ofstream class described, XRF 893
  portabilitly guidelines, PT 291
  ostream objects, XRF 873
  ostream_withassign class described, XRF 908
  ostrstream class described, XRF 911
  portability guidelines, 291
  privilege mechanism, defined, ET 810
  programming, C/C++
    alternatives, iostream classes tutorial, XUG 364
  putting extracted character back into stream, istream::putback, XRF 882
  returning character without extracting, istream::peek, XRF 882
  setting
    floating-point precision variable, ios::precision, XRF 861
    internal field width variable, XRF 865
    internal floating-point precision variable, setprecision, XRF 870
  specified format bits, ios::setf, XRF 863
    stream's mode to text, ios& text, XRF 871
  stdiobuf class described, XRF 915
  stdiostr class described, XRF 917

- I/O (*continued*)
  - stream buffers, returning number of bytes stored in, ostrstream::pcount, XRF 913
  - streambuf class described, XRF 919
  - stream objects, cin, XUG 33
  - stream objects, cout, XUG 33, 61
  - streams
    - assigning istream object to istream_withassign object, istream_withassign::operator =, XRF 889
    - called after extraction operations, istream::isfx, XRF 881
    - called prior to extraction operations, istream::ipfx, XRF 880
    - changing get pointer, istream::seekg, XRF 883
    - classes, *See* iostream classes
    - manipulators, custom, XUG 398
    - synchronizing C++ with standard C stdio, ios::sync_with_stdio, XRF 863
    - synchronizing internal buffer with external character source, istream::sync, XRF 884
  - strstream class described, XRF 939
  - strstreambuf class described, XRF 943
  - testing for end of file, ios::eof, XRF 857
  - virtual overflow function, streambuf::overflow, XRF 926–927
- I/O functions
  - based heaps, LIB 49
  - buffering, LIB 33
  - closing files, LIB 40
  - committing buffer contents to disk, LIB 37
  - console, LIB 43–44
  - increasing system limits, LIB 42
  - low-level routines, LIB 38–39
  - near and far heaps, LIB 48–49
  - opening files, LIB 39
  - port, LIB 43–44
  - reading and writing data, LIB 39
  - reading and writing operations, LIB 37–38
  - searching and sorting routines (list), LIB 54
  - stream buffering, LIB 36
  - system calls, LIB 55
  - text and binary modes, LIB 32
- I/O functions (*continued*)
  - types, LIB 31
  - using modified startup files, LIB 42
  - variable-length argument lists, LIB 59
  - virtual memory allocation, LIB 60
- Icon (.ICO) files,QuickWin, custom, PT 164–165
- Icon resource files, XUG 156
- Icons
  - activating and displaying, CWnd::OpenIcon, XRF 800
  - called, XRF 743, 773, 775, 777
  - customizing, QuickWin, PT 164–165
  - drawing on CDC device, CDC::DrawIcon, XRF 176
  - minimized document child windows, arranging, CMDIFrameWnd::MDIIconArrange, XRF 406
  - screen display, PWB, ET 68
  - windows, changing, Foundation classes cookbook, XUG 326
- ID numbers, Foundation classes tutorial, XUG 166
- Identification, getting process, _getpid function, LIB 373
- Identifiers
  - attributes, LR 37
  - block scope rules, LR 36
  - C/C++
    - predefined expression syntax, ET 778, 780, 785
  - case distinction, LINK option, ET 585
  - defined, ET 809
  - described, LR 5–9
  - enumeration tags, LR 62–63
  - external linkage, LR 37
  - function scope rules, LR 36
  - in function declarations, LR 86
  - initializing, LR 80
  - lifetime, LR 35
  - linkage, LR 7, 27, 36–37
  - lists, LR 97
  - l-values, LR 107–108
  - name spaces, LR 39–40
  - names
    - hiding, LR 45
    - in different scopes, LR 36
    - length, LR 6

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

- Identifiers (*continued*)
  - names (*continued*)
    - nested visibility, LR 39
    - restrictions, LR 7
    - with external linkage, LR 37
  - nonmodifiable, LR 52
  - parameters, naming, LR 85
  - passing, using /D option, LR 206
  - portability guidelines, PT 288
  - restrictions, LR 194, 235
  - scope, LR 7
  - searching, PWB, ET 93
  - statement labels, LR 6
  - storage, LR 34
  - types, LR 106
  - values, LR 5, 34
  - visibility, LR 35
- Identifiers, C++
  - described, LR+ 5
  - predefined macros, LR+ 7–9
  - restrictions, LR+ 6
- Identifying child windows, CWnd::IsChild, XRF 709
  - classes, TUT 173–174
- Idle loop processing, Foundation classes cookbook, XUG 309–310
- Idle state, called to inform main window, CWnd::OnEnterIdle, XRF 737
- Idle-time processing, Windows applications, CWinApp::OnIdle, XRF 637
- IEEE binary format, converting floating-point numbers to Microsoft
- binary formats, _fieeetomsbin and _fmsbintoieee funct, LIB 279
- IEEE (Institute of Electrical and Electronics Engineers) format
  - defined, ET 810
  - floating-point numbers, LR 99
  - floating-point types, PT 127
- !IF preprocessing directive, NMAKE, ET 689
- #if preprocessor directive
  - described, LR 190, 202–206
  - testing code, LR 3
- #if preprocessor directive, C++, LR+ 379–383
  - header file, TUT 61
- if statements
  - described, LR 158–159
  - nesting, LR 159
- if statements, C++
  - selection statements, LR+ 138–139
- #ifdef _DEBUG statement, Foundation classes tutorial, XUG 34
- !IFDEF preprocessing directive, NMAKE, ET 689
- #ifdef preprocessor directive
  - described, LR 190
  - equivalent to #if, LR 206
- #ifdef preprocessor directive, C++, LR+ 383–384
- !IFNDEF preprocessing directive, NMAKE, ET 689
- #ifndef preprocessor directive, LR 190
- #ifndef preprocessor directive, C++, LR+ 383–384
- ifstream class
  - described, XRF 845
  - iostream classes tutorial, XUG 383
  - member functions
    - attach, XRF 846
    - close, XRF 846
    - fd, XRF 847
    - ifstream, XRF 847–848
    - ~ifstream, XRF 849
    - is_open, XRF 849
    - open, XRF 849–850
    - rdbuf, XRF 850
    - setbuf, XRF 850
    - setmode, XRF 851
- ifstream constructor, XRF 847–848
- ifstream destructor, XRF 849
- ifstream objects
  - creating, ifstream::ifstream, XRF 847–848
  - destroying, ifstream::~ifstream, XRF 849
- .IGNORE dot directive, NMAKE, ET 687
- ignore member function, istream class, XRF 880
- /IGNORECASE option, LIB, ET 701
- Images
  - animated graphics functions (list), PT 192–193
  - graphics. *See* Graphics
  - retrieving from buffers, _putimage functions, LIB 600–601
  - storing in buffers, _getimage functions, LIB 367–369
- _imagesize function, LIB 426–427; PT 192
- _imagesize_w function, LIB 426–427; PT 192
- _imagesize_wxy function, LIB 426–427; PT 192
- IMPLEMENT_DYNAMIC macro, XRF 40, 471
- IMPLEMENT_SERIAL macro, XRF 471
- Implementation, class
  - private members, TUT 45, 54
  - source files, TUT 61–62
- Implementation file, Foundation classes cookbook, XUG 263
- IMPLEMENT_DYNAMIC Macro, Foundation classes cookbook, XUG 264–266
- IMPLEMENT_SERIAL macro
  - Foundation classes cookbook, XUG 282
  - Foundation classes tutorial, XUG 29, 33, 38, 44

IMPLIB
  case sensitivity, ET 747
  command lines, ET 746–747
  error messages. *See also* Error Messages," Part 2 of this book
  LINK import libraries, ET 568
  module-definition files, ET 609
  options, ET 747
  overview, ET 743–747
  syntax, ET 746–747
Implicit links, HELPMAKE, ET 719–720
Import libraries
  combining, ET 704
  creating, IMPLIB, ET 745–747
  defined, ET 810
  linking, ET 568
Import Library Manager. *See* IMPLIB
IMPORTS statement, module-definition files, decorated names, ET 609, 624–625, 790–791
Improving, float consistency, CL options, ET 537–538
in_avail member function, streambuf class, XRF 926
.INC files, ET 810
INCLUDE environment variable, LR 201
  BSCMAKE, ET 736
  Foundation classes tutorial, XUG 8
Include files
  browser database, PWB, ET 104
  BSCMAKE, ET 737
  C++
    defined, LR+ 365
    described, LR+ 376–378
  defined, ET 810
  finding symbols, PWB, ET 101
  naming conventions, LIB x
  project dependencies, PWB, ET 45, 48
  search directory, CL option, ET 525–526
  unreferenced symbols, packing files, ET 739
Include path, consistency rules, precompiled headers, PT 40
!INCLUDE preprocessing directive
  NMAKE, ET 689
#include preprocessor directive
  described, LR 190–191, 200
  PWB project dependencies, ET 45
#include preprocessor directive, C++, LR+ 376–378
  Foundation classes tutorial, XUG 49
  header files, TUT 62
INCLUDE statement
  module-definition files, ET 609, 627
  overlaid DOS programs, ET 600–601
Inclusive OR operator, bitwise, C++. *See* Bitwise inclusive OR operator, C++
Incomplete types, LR 100–101
Increasing
  portability. *See* Portability guidelines
  program speed, PT 9–10
Increment operator, C++
  overloading, LR+ 355–358
  postfix expressions, LR+ 90–91
  unary-operator expressions, LR+ 94–95
Incremental build, building a database, BSCMAKE, ET 733–735
Indenting
  automatic, PWBC switches, ET 312
  command line, NMAKE, ET 660
  dependency lines, NMAKE, ET 655
  text
    HELPMAKE, ET 726
    PWB, ET 296
Index, Microsoft Advisor, ET 762, 764
Index command
  CodeView, ET 374
  PWB, ET 78, 146, 757
  QuickWin, PT 151
Indexing arrays, mixed-language programming, PT 263–264
Indirect addressing, portability guidelines, PT 281–283
Indirection operator (*), LR 108, 122–124
Indirection operator, C++
  unary-operator expressions, LR+ 92
Indirection register, debugging assembly language, ET 414
Inequality operator, CRect class, XRF 532

/INF option, LINK, ET 582
Inference rules, NMAKE, ET 660, 680–687
Inferred dependents, NMAKE
- dependency line, ET 659
- inference rules, ET 685–686

Infinite loops, terminating execution, ET 387–388
InflateRect member function, CRect class, XRF 525–526
Inflating rectangles, CRect::InflateRect, XRF 525
/INFO option, LINK, ET 562, 582, 601–602
Infodialog switch, PWB, ET 264, 285–286
Information contexts, creating for specified device, CDC::CreateIC, XRF 173
Information function, PWB, ET 151, 173
/INFORMATION option, LINK, ET 562, 582
ing operator, CString class, XRF 596-600
Inheritance
- compared with composition, TUT 178
- construction order, LR+ 305
- design issues, TUT 169–171, 177–180
- multiple
  - access control, LR+ 297
  - base classes, LR+ 267–271
  - described, LR+ 264–265
  - design issues, TUT 180
  - name ambiguities, LR+ 271–274
  - overview, TUT 132–134
- overview, TUT 110
- single, described, LR+ 259–264
- unions, LR+ 250
- using C++ expressions, ET 410

Inherited macros, NMAKE, ET 679–680
.INI files
- defined, ET 810

INIT environment variable
- defined, ET 810
- PWB, ET 67, 137
- remote debugging, ET 396

InitApplication member function, CWinApp class, XRF 632
- overriding, XRF 11; XUG 308
- Windows applications, writing, XUG 305

Initialization, C++
- aggregate types, LR+ 219–222
- assignment, TUT 77–78
- automatic objects, LR+ 150
- base classes, TUT 116
- character arrays, LR+ 222–223
- class objects, TUT 77–78
- const members, TUT 60
- constructors, LR+ 284

Initialization, C++ (*continued*)
- copying objects, LR+ 333–337
- Foundation classes vs. native Windows, XRF 12
- local variable handling, LR+ 47–49
- member objects, TUT 58–60
- new operator, objects allocated with, LR+ 98
- order of execution, LR+ 43–44
- pointers to const objects, LR+ 218
- references, LR+ 223–224; TUT 27, 77
- special member functions, using
  - arrays, LR+ 328–329
  - bases and members, LR+ 329–333
  - described, LR+ 325–328
  - static objects, LR+ 329
- static members, LR+ 219
- static objects, LR+ 152–154, 329
- uninitialized objects, LR+ 218

Initialization routine
- debugging, ET 381–382

Initialize function, PWB, ET 151, 172–173
Initializers, C++, LR+ 217–218
Initializing
- aggregate types, LR 93–97
- characters of strings to given characters, _strnset and _fstrnset functions, LIB 774–775
- chart environment, _pg_defaultchart function, LIB 562–563
- declarators, LR 91–97
- fonts graphics system, _registerfonts function, LIB 618
- identifiers, LR 80
- internal static variables, LR 48
- local variables, LR 35
- menus, XRF 744
- modal dialog boxes, XRF 446-447
- presentation graphics, _pg_initchart function, LIB 570
- restrictions, LR 66
- scalar types, LR 91–93
- strings, LR 97–98
- values, LR 91–97
- virtual memory manager, PT 90–91; LIB 845–846
- Windows applications, XRF 632

InitInstance member function, CWinApp class, XRF 11, 632
- Foundation classes tutorial, XUG 88
- Windows applications, writing, XUG 305
- windows, displaying, XUG 93

init_seg pragma directive, LR+ 388
Inline assembler, LR 174

- Inline assembly
  - advantages, PT 111
  - __asm blocks
    - described, PT 112
    - __fastcall calling convention limitations, PT 120–121
    - features, PT 113–115
    - function calls, PT 122–123
    - labels, PT 121–122
    - language elements, using, PT 115–119
    - macros, defining as, PT 123–124
    - optimization, effects on, PT 124–125
    - registers, PT 120–121
  - __asm keyword, PT 112
  - comments, PT 114
  - data directives, limitations, PT 113
  - data members, PT 117–118
  - debugging with CodeView, PT 115
  - _emit pseudoinstruction, PT 115
  - expressions, using, PT 113
  - __fastcall calling convention, PT 120–121
  - function calls, PT 122–123
  - functions, writing, PT 118–119
  - instruction set, PT 113
  - labels, PT 121–122
  - macros
    - defining __asm blocks as, PT 123–124
    - limitations, PT 113
  - MASM compatibility limitations, PT 113
  - operators, limitations, PT 113–114, 116
  - optimization concerns, PT 124–125
  - registers, PT 120–121
  - segment referencing, PT 114
  - structure types, PT 117–118
  - symbols, PT 117
  - type and variable sizes, PT 114
  - using, PT 111
  - variables, PT 117–118
- Inline code
  - debugging, ET 322
- Inline emulator option, floating-point math, PT 134, 137
- Inline expansion
  - control, CL option, ET 532
  - controlling, LR 260
- Inline files, NMAKE
  - creating, ET 664–666
  - multiple, ET 667
  - reusing, ET 666–667
- Inline functions
  - compared to macros, LR 192
  - described, LR 173
- Inline functions, C++
  - described, LR+ 159–163, 246–247
  - header files, TUT 61
  - macros, TUT 16
  - member functions, TUT 52
  - overview, TUT 15, 17
- __inline keyword, PT 13
- Inline math coprocessor option, floating-point math, PT 134–137
- inline specifier, C++, LR+ 159
- inline_depth pragma directive, C++, LR+ 389
- inline_depth pragma, new in version 7.0, LR 258
- inline_recursion pragma directive, C++, LR+ 389
- inline_recursion pragma, new in version 7.0, LR 258
- Inlining, PT 13
- In-memory files, CMemFile class described, XRF 411
- _inp function, LIB 428
- Input
  - LINK, ET 564–572
  - redirecting, CodeView, ET 475, 477
  - sequential and random-access, istream class described, XRF 875
- Input command, QuickWin, PT 150
- Input control, specifying for CWnd, CWnd::OnGetDlgCode, XRF 739–740
- Input focus
  - active window, QuickWin, PT 152, 161–162
  - called after gaining, CWnd::OnSetFocus, XRF 783
  - called after window has realized logical palette, CWnd::OnPaletteChanged, XRF 773–774
  - called before losing, CWnd::OnKillFocus, XRF 747–748

Input focus (*continued*)
- called when ALT and another key pressed, CWnd::OnSysKeyDown, XRF 791–792
- called when CWnd about to receive, CWnd::OnQueryNewPalette, XRF 777
- called with release of key pressed with ALT, CWnd::OnSysKeyUp, XRF 792–794
- claiming, CWnd::SetFocus, XRF 809
- specifying
  - character value of dead key, CWnd::OnSysDeadChar, XRF 790–791
  - virtual-key code of Control menu key, CWnd::OnSysChar, XRF 786–787

Input streams
- C++, TUT 8
- described, iostream classes tutorial, XUG 382
- extraction errors, iostream classes tutorial, XUG 384
- extraction operators, iostream classes tutorial, XUG 384, 391
- ifstream class, iostream classes tutorial, XUG 383
- istream class, iostream classes tutorial, XUG 382
- istrstream class, iostream classes tutorial, XUG 383
- manipulators, iostream classes tutorial, XUG 385, 398
- objects, constructing
  - input file stream constructors, XUG 383
  - input string stream constructors, XUG 384

Inputting bytes or words from port, _inp and _inpw functions, LIB 428

_inpw function, LIB 428

Insert function, PWB, ET 151, 173–174

Insert mode, toggling, in PWB, ET 174, 277

InsertAfter member function, CObList class, XRF 491–492

InsertAt member function, CObArray class, XRF 456–457

InsertBefore member function, CObList class, XRF 492–493

Inserting
- arguments into streams, ostream::operator<<, XRF 906
- characters, into output stream, ostream::put, XRF 903
- characters, PWB, ET 173–174
- clipboard data into edit control, CEdit::Paste, XRF 297
- element in array, CObArray::InsertAt, XRF 457
- files, module statements, ET 627
- lines, PWB, ET 177–178

Inserting (*continued*)
- new menu items, CMenu::InsertMenu, XRF 428–430
- RTF formatting codes, HELPMAKE, ET 726
- space, PWB, ET 215–216

Insertion operator, TUT 6
- CArchive class, XRF 103
- CDumpContext class, XRF 278–279
- ostream class, XRF 906

Insertion operators
- CString::operator <<, XRF 593
- iostream classes tutorial, XUG 367
- overloading
  - iostream classes tutorial, XUG 380–381

InsertMenu member function
- CMenu class, XRF 428–430
  - CWnd::GetSystemMenu, XRF 699

Insertmode function, PWB, ET 151, 174

InsertString member function
- CComboBox class, XRF 151
- CListBox class, XRF 366–367

Installable file system defined, ET 810

Installing
- *See also Getting Started*
- CodeView, ET 327–329
- terminate-and-stay-resident programs, _dos_keep function, LIB 210–211

Instruction sets, generating, CL option, ET 514–515

Instructions
- inline assembler, PT 112–113, 121–122
- inline, floating-point math options, PT 137
- p-code
  - data types, PT 48–49
  - modes, PT 48
  - naming conventions, PT 47–49
  - qualifiers, PT 48
  - (table), PT 297–299
- preprocessor, LR 260
- processor, generating specific, PT 24

Insufficient memory
- testing for, C++, LR+ 321–323
- handling, portability guidelines, PT 283–284

int, searching, PWB, ET 91–93

int type
- described, LR 98–99
- portability guidelines, PT 272
- signed, LR 51

int type, C++
- described, LR+ 50–51
- size, LR+ 51–52

- _int86 function, LIB 429–430
- _int86x function, LIB 431–432
- _intdos function, LIB 433–434
- _intdosx function, LIB 435–436
- Integer constants, C++, LR+ 14–16
- Integers
  - calculating absolute value of long integers, labs function, LIB 445–446
  - converting, LR 131, 141–143
    - long integers to strings, _ltoa function, LIB 474–475
    - to strings, _itoa function, LIB 442–443
    - unsigned long integers to strings, _ultoa function, LIB 823–824
  - defined, ET 810
  - demotion, LR 239
  - described, LR 13
  - getting from stream, _getw function, LIB 389–390
  - limits, LR 15
  - range of value, LR 238
  - testing values, is functions, LIB 437–440
  - translating dialog box control text into, CWnd::GetDlgItemInt, XRF 690–691
  - types, LR 14
  - writing to streams, _putw function, LIB 603–604
- Integral constant expressions, C++
  - conversion to null pointer, LR+ 75
- Integral conversions, C++
  - floating to integral, LR+ 69
  - integral to floating, LR+ 69
  - signed to unsigned, LR+ 67
  - standard, LR+ 68
  - unsigned to signed, LR+ 67–68
- Integral promotion
  - effect of unary plus operator (*), LR 124
  - portability guidelines, PT 285–287
  - preserving value, LR 141
- Integral promotion, C++
  - described, LR+ 66–67
  - enumerators, LR+ 177–178
- Integral types
  - conversions, LR 126, 141–146
  - listed, LR 51
- Integral types, C++
  - conversion
    - signed to unsigned, LR+ 67
    - standard (to shorter types), LR+ 68
    - to floating, LR+ 69
    - unsigned to signed, LR+ 67–68
  - described, LR+ 50–51
  - limits, LR+ 62
- Interactive devices, LR 234
- Interface, class
  - header files, TUT 61–62
  - public members, TUT 45
- Interface, designing, TUT 175
- Interface file, Foundation classes cookbook, XUG 263
- Interfaces, keyboard and mouse, adding, Phone Book sample program, XUG 230–233
- Internal character arrays
  - returning pointer from stream, ostrstream::str, XRF 914
  - strstream class, returning pointer to, strstream::str, XRF 940
- Internal field width variable, setting, ios::width, XRF 865
- Internal fill character variable, setting, ios::fill, XRF 857
- Internal linkage
  - *See also* Storage classes
  - initializing variables, LR 48
  - objects, LR 43
  - overview, LR 36
  - storage-class specifiers, LR 47–50
- Internal linkage, C++
  - defined, LR+ 25–26
  - described, LR+ 33
- Internationalization routines, LIB 44
- Interoverlay calls
  - defined, ET 810
  - limiting, LINK, ET 579
- Interpreter, run-time, p-code, PT 43–46
- Interpreting, tokens, LR 4
- Interrupt call defined, ET 810
- Interrupt functions, LR 175

Interrupt handlers, LR+ 420–421
__interrupt keyword
  CL, enabling, ET 550
  invalid with fastcall, LR 171
  specifying interrupt handler, LR 175
__interrupt keyword, C++, LR+ 7, 420–421
Interrupt number, MOVE, ET 604
Interrupt vectors, setting, _dos_setvect function, LIB 229–231
Interrupting
  CodeView, ET 387, 470, 475
Interrupts
  8086
    executing and accepting segment-register values, _int86x function, LIB 431–432
    executing, _int86 function, LIB 429–430
  chaining between handlers, _chain_intr function, LIB 138–139
  disabling, _disable function, LIB 178
  enabling, _enable function, LIB 243
  getting vector values, _dos_getvect function, LIB 209
  setting signal handling, signal function, LIB 707–711
  trapping, CodeView, ET 341–342
IntersectClipRect member function, CDC class, XRF 212–213
Intersection operator, CRect class, XRF 535
IntersectRect member function, CRect class, XRF 526
INT_MAX constant, C++
  integral limits, LR+ 62
INT_MIN constant, C++
  integral limits, LR+ 62
Intrinsic functions
  calling, CodeView expressions, ET 405
  generating, PT 10–12
  new in version 7.0, LR 258
  pragma, LR 211
  specifying, LR 211
intrinsic pragma
  format, PT 11
  generating inline code for functions, LR 258
  precompiled header compilation, effect on, PT 41
  using, PT 12
intrinsic pragma, C++, LR+ 389–390
Invalidate member function, CWnd class, XRF 706–707
  Foundation classes tutorial, XUG 214
InvalidateLine member function, CWnd class, adding, XUG 241
InvalidateRect member function, CWnd class
  CEdit::SetTabStops, XRF 302
  described, XRF 707
InvalidateRgn member function, CWnd class, XRF 708
Invalidating client areas
  entire, CWnd::Invalidate, XRF 706–707
  within given rectangle, CWnd::InvalidateRect, XRF 707
  within given region, CWnd::InvalidateRgn, XRF 708
Invariant code
  removing, PT 18–19
  removing, CL option, ET 535–536
Inverting
  rectangle contents, CDC::InvertRect, XRF 213
  region colors, CDC::InvertRgn, XRF 214
InvertRect member function, CDC class, XRF 213
InvertRgn member function, CDC class, XRF 214
ios class
  constructor, ios::ios, XRF 860
  data members
    adjustfield, XRF 867
    basefield, XRF 867
    floatfield, XRF 867
    operator, XRF 866
  described, XRF 852–854
  manipulators, XRF 868–871
    ios& binary, XRF 868
    ios& dec, XRF 868
    ios& hex, XRF 868
    ios& oct, XRF 869
    ios& text, XRF 871
    resetiosflags, XRF 869
    setfill, XRF 869
    setiosflags, XRF 870
    setprecision, XRF 870
    setw, XRF 871
  member functions
    bad, XRF 855
    bitalloc, XRF 855
    clear, XRF 856
    delbuf, XRF 856–857
    eof, XRF 857
    fail, XRF 857
    fill, XRF 857–858
    flags, XRF 858–859
    good, XRF 859
    hex, XRF 855
    in, XRF 930
    ios, XRF 860
    ~ios, XRF 860

- ios class (*continued*)
  - member functions (*continued*)
    - iword, XRF 860
    - left, XRF 855
    - out, XRF 930
    - precision, XRF 861
    - pword, XRF 861
    - rdbuf, XRF 862
    - rdstate, XRF 862
    - setf, XRF 863
    - stdio, XRF 863, 902
    - sync_with_stdio, XRF 863
    - tie, XRF 864
    - unitbuf, XRF 902
    - unsetf, XRF 864
    - width, XRF 865
    - xalloc, XRF 865
  - operators, XRF 867
  - virtual destructor, ios::~ios, XRF 860
- ios constructor, XRF 860
- ios destructor, XRF 860
- ios enumerators, XRF 862
- iostream class
  - described, XRF 872–873
  - member functions, XRF 873
- iostream classes
  - advanced programming tutorial, XUG 395–405
  - flags, iostream classes tutorial, XUG 374–375
  - fstream class, iostream classes tutorial, XUG 392–393
  - hierarchy, XUG 365
  - input streams
    - described, XUG 382
    - extraction errors, XUG 384
    - extraction operators, XUG 384, 391
    - ifstream class, XUG 383
    - iostream classes tutorial, XUG 385
    - istream class, XUG 382
    - istrstream class, XUG 383
    - member functions, XUG 386–391
    - objects, constructing, XUG 383–384
  - introduction, XUG 363
- iostream classes (*continued*)
  - output streams
    - binary output files, XUG 378–379
    - buffering, effects, XUG 377–378
    - deriving, XUG 399–405
    - format control, XUG 368–373
    - insertion operators, XUG 367, 380–381
    - manipulators, XUG 381–382, 395–399
    - objects, constructing, XUG 366–367
    - ofstream class, XUG 366
    - ofstream class member functions, XUG 373–377
    - ostream class, XUG 365
    - ostrstream class, XUG 366
  - strstream class, iostream classes tutorial, XUG 392–393
  - tutorial, XUG 363–393
- iostream constructor, XRF 873
- iostream destructor, XRF 873–874
- IOSTREAM.H, TUT 5, 8
- iostream objects, destroying, iostream::~iostream, XRF 873
- Iostream_init class
  - described, XRF 874
  - member functions, XRF 874
- Iostream_init constructor, XRF 874
- Iostream_init destructor, XRF 874
- iostreams
  - cerr, TUT 7
  - cin, TUT 8
  - cout, TUT 5–6
  - manipulators, TUT 7
- ipfx member function, istream class, XRF 880–881
- is functions, LIB 437–440
- is_open member function
  - filebuf class, XRF 834
  - fstream class, XRF 842
  - ifstream class, XRF 849
  - ofstream class, XRF 895
- isalnum function, LIB 437–440
- isalpha function, LIB 437–440
- __isascii function, LIB 437–440
- _isatty function, LIB 441
- IsChild member function, CWnd class, XRF 709

- iscntrl function, LIB 437–440
- __iscsym function, LIB 437–440
- __iscsymf function, LIB 437–440
- IsDialogMessage member function, CDialog class, XRF 268–269
- IsDlgButtonChecked member function, CWnd class, XRF 709
- IsEmpty member function
  - CMapStringToOb class, XRF 382
  - CObList class, XRF 493
  - CString class, XRF 584–585
- isfx member function, istream class, XRF 881
- IsIconic member function, CWnd class, XRF 710
- IsKindOf function, using, Foundation classes cookbook, XUG 266
- IsKindOf member function, CObject class, XRF 472
  - CArchive::ReadObject, XRF 100
- IsLoading member function, CArchive class, XRF 98
  - CObject::Serialize, XRF 473
- IsRectEmpty member function, CRect class, XRF 527
- IsRectNull member function, CRect class, XRF 527
- IsSerializable member function, CObject class, XRF 473
- IsStoring member function, CArchive class, XRF 98
  - CObject::Serialize, XRF 473
  - serialization, XUG 34
- istream class
  - described, iostream classes tutorial, XUG 382
  - member functions
    - close, XUG 391
    - get, XUG 386–388
    - getline, XUG 388–389
    - open, XUG 386
    - read, XUG 389
    - seekg, XUG 390–391
    - tellg, XUG 390–391
  - described, XRF 875–876
  - extraction operators, istream::operator>>, XRF 885
  - manipulators, XRF 886
  - member functions
    - eatwhite, XRF 877
    - gcount, XRF 877
    - get, XRF 878–879
    - getline, XRF 879–880
    - ignore, XRF 880
    - ipfx, XRF 880–881
    - isfx, XRF 881
    - istream, XRF 881
- istream class (*continued*)
  - member functions (*continued*)
    - ~istream, XRF 882
    - peek, XRF 882
    - putback, XRF 882
    - read, XRF 882–883
    - seekg, XRF 883
    - sync, XRF 884
    - tellg, XRF 884
  - operators, XRF 885, 889
- istream constructor, XRF 881
- istream destructor, XRF 882, 888–889
- istream objects
  - assigning to istream_withassign object, istream_withassign::operator =, XRF 889
  - creating, istream::istream, XRF 881
  - destroying, istream::~istream, XRF 882
- istream_withassign class, XRF 888
  - described, XRF 887
  - member functions
    - istream_withassign, XRF 888
    - ~istream_withassign, XRF 888
- istream_withassign constructor, XRF 888
- istream_withassign destructor, XRF 888
- istream_withassign objects
  - creating, XRF 888
  - destroying, XRF 888
- istrstream class, XRF 890
  - iostream classes tutorial, XUG 383
  - member functions
    - istrstream, XRF 891
    - ~istrstream, XRF 891
    - rdbuf, XRF 892
    - str, XRF 892
- istrstream constructor, XRF 891
- istrstream destructor, XRF 891
- istrstream objects
  - creating, istrstream::istrstream, XRF 891
  - destroying, istrstream::~istrstream, XRF 891
- IsWindowEnabled member function, CWnd class, XRF 710
- IsWindowVisible member function, CWnd class, XRF 710
- IsZoomed member function, CWnd class, XRF 711
- Italics, HELPMAKE formatting
  - QuickHelp format, ET 721
  - rich text format, ET 726
- Iteration, collection classes, Foundation classes cookbook, XUG 272

Iteration statements, C++, LR+ 142
Iterators, TUT 91, 93
_itoa function, LIB 442–443
/Iu option, BSCMAKE, ET 737
iword member function, ios class, XRF 860

J

/J option, CL, ET 526–527; LR 52, 98, 238
_j0 function, LIB 103–105
_j0l function, LIB 103–105
_j1 function, LIB 103–105
_j1l function, LIB 103–105
_jn function, LIB 103–105
_jnl function, LIB 103–105
Jump statements, C++, LR+ 147–149 Jumping to labels, inline assembly, PT 121–122
Justification, text, setting,
  CDC::SetTextJustification, XRF 248–249
Justifying tagged expressions, ET 785

K

K command, CodeView, ET 435–436
/K option
  HELPMAKE, ET 712–713
  NMAKE, ET 649
  RM, ET 748
_kbhit function, LIB 444
KEEP, inline files, NMAKE, ET 665
Keepmem switch, PWB, ET 264, 285
Key assignments, PWB, ET 107, 119–121, 134–135
  cursor movement commands, ET 154–155
  default, ET 146–150
  Graphic function, ET 172
  menu commands, ET 142–146
  Unassigned function, ET 218
Key Assignments command, PWB, ET 75
Key box, assigning key function, PWB, ET 120
Key lookups, XRF 27
Keyboard
  choosing commands, ET 78
  executing PWB commands, ET 78–79
  hyperlinks, activating, ET 759

Keyboard (*continued*)
  input
    enabling or disabling, CWnd::EnableWindow, XRF 681
    returning active key, CWnd::OnChar, XRF 718–719
    specifying whether CWnd is enabled for, CWnd::IsWindowEnabled, XRF 710
  nagivation in CodeView, ET 349
  shortcut keys, PWB, ET 79
  Windows messages, Foundation classes cookbook, XUG 356–357
Keyboard and mouse, message handlers,
  Foundation classes tutorial, XUG 199, 234–235
Keyboard, checking console for input, _kbhit
  function, LIB 444
Keys
  nonsystem
    called on input, CWnd::OnKeyDown, XRF 745–746
    called on release, CWnd::OnKeyUp, XRF 746–747
  shortcut, PWB, ET 79
  TOOLS.INI syntax, PWB, ET 135
Keywords
  /A options, CL, ET 489
  addressing conventions, specifying, LR 56
  addressing mode, specifying, PT 64–66
  binding characteristics, LR 56
  calling conventions, specifying, LR 169
  compiling older versions, LR 264
  compressing, HELPMAKE option, ET 712–713
  declarators, LR 55–61
  described, LR 4–5
  determining addressing conventions, LR 56
  help, getting, ET 762–763
  Microsoft-specific, compiling, LR 263
  redefining, LR 4
  specific keyword, ET 489
Keywords, C++
  described, LR+ 6–7
  grammar summary, LR+ 424
  (list), LR+ 6–7

Keywords, C++ (*continued*)
  Microsoft-specific
    See also Modifiers, C++
    grammar summary, LR+ 435–436
    (list), LR+ 7
    modified pointers, conversion, LR+ 74
  this, XUG 104, 106
  virtual, XUG 90
KillTimer member function, CWnd class, XRF 711
Kilobyte defined, ET 810

L

L command, CodeView, ET 422, 436–437
:l command, HELPMAKE, ET 723
/L option
  CL, debugging multiple applications, ET 380–382
  CodeView, ET 338, 342
  HELPMAKE, ET 713
Label defined, ET 810
Label/Function command, CodeView, ET 361–362
Labeled statements, LR 157–158
Labeled statements, C++, LR+ 134–136
Labels
  charts and graphs, PT 202
  in switch statements, LR 161–164
  inline assembly, PT 121–122
  names, LR 35
  overview, LR 39
  scope, LR 35
  statements, LR 152
Labels, C++
  case statements, using with, LR+ 134–136
  switch statements, restrictions, LR+ 135–136
labs function, LIB 445–446
Language command, CodeView, ET 368, 370
Language dialog box, CodeView, ET 370
Language extensions, enabling, CL options, ET 550–552
Language Options command, PWB, ET 75; PT 134
Large memory models
  defined, ET 810
  option, CL, PT 63
Lastproject switch, PWB, ET 141, 264, 285–286
Lastselect function, PWB, ET 151, 175
Lasttext function, PWB, ET 151, 175–176
/Lc option, CL, LR 267
/Ld option, CL, ET 527; LR 260
LDBL_DIG constant, C++
  floating limits, LR+ 63–64
LDBL_EPSILON constant, C++
  floating limits, LR+ 63–64
LDBL_MANT_DIG constant, C++
  floating limits, LR+ 63–64
LDBL_MAX constant, C++
  floating limits, LR+ 63–64
LDBL_MAX_10_EXP constant, C++
  floating limits, LR+ 63–64
LDBL_MAX_EXP constant, C++
  floating limits, LR+ 63–64
LDBL_MIN constant, C++
  floating limits, LR+ 63–64
LDBL_MIN_10_EXP constant, C++
  floating limits, LR+ 63–64
LDBL_MIN_EXP constant, C++
  floating limits, LR+ 63–64
LDBL_RADIX constant, C++
  floating limits, LR+ 63–64
Ldelete function, PWB, ET 152, 176–177
ldexp function, LIB 447–448
_ldexpl function, LIB 447–448
ldiv function, LIB 449–450
ldiv_t function, LIB 449–450
Least-recently-used (LRU) algorithm, overlaid DOS programs, ET 598
Leaving
  CodeView, ET 360
  PWB, ET 171, 251
Left function, PWB, ET 152, 177
Left member function
  CString class, XRF 585
  ios class, ios::bitalloc, XRF 855
Left-shift assignment operator (<< =), LR 138
Left-shift operator, C++
  binary-operator expressions, LR+ 106–107
  overloading, LR+ 359
Legends, presentation graphics
  described, PT 205
  structure types, PT 225
_legendtype structures, presentation graphics, PT 225
Length, files
  changing, CFile::SetLength, XRF 318
  getting in bytes, CFile::GetLength, XRF 309
.length command, HELPMAKE, ET 723, 726
LENGTH operator, inline assembler use, PT 114
Less-than operator (<), Redirect Output command, CodeView, ET 340, 475
Less-than operator, C++
  binary-operator expressions, LR+ 107–109
  overloading, LR+ 359

- Less-than-or-equal-to operator, C++
  - binary-operator expressions, LR+ 107–109
  - overloading, LR+ 359
- Levels, function declarations, LR 28
- Lexical scope. *See* Scope
- _lfind function, LIB 451–452
- \li formatting code
  - HELPMAKE, ET 726
- /Li option, CL, LR 267
- /LI option, LINK, ET 582
- LIB
  - case sensitivity, ET 701
  - combining libraries, ET 704
  - command line, ET 698
  - commands, ET 702–705
  - consistency checks, ET 700
  - cross-reference listing, ET 705–706
  - error codes, ET 708
  - error messages. *See* “Error Messages,” Part 2 of this book
  - extended dictionaries, suppressing, ET 701
  - fields, specifying, ET 699–705
  - file compatibility, ET 697–698
  - help, ET 701
  - library files, creating, ET 700, 703
  - LINK import libraries, ET 568
  - object modules
    - adding, ET 703
    - copying, ET 705
    - deleting, ET 704
    - distinguishing, ET 698
    - moving, ET 705
    - replacing, ET 704–705
  - options, ET 700–701
  - output library, ET 706–707
  - overview, ET 697–698
  - page size, ET 701–702
  - prompts, ET 698–699
  - response file, ET 699
  - syntax, ET 699–705
- LIB environment variable, ET 810; XUG 8
- .LIB files defined, ET 810
- LibEntry routine, debugging, ET 381–382
- Libraries
  - *See also* LIB
  - combining, ET 704
  - default
    - defined, ET 806
    - LINK, ET 568, 583–584
  - defined, ET 810
  - floating-point math, PT 132, 137, 139
  - fonts, PT 195
  - graphics, PT 167
  - import. *See* Import libraries
  - linking, LIB 6; PT 237–238
  - load, defined, ET 811
  - managing, with LIB, ET 697–698
  - math operations, CL options, ET 508–513
  - memory models, PT 59, 74
  - MOVE, ET 600
  - output, ET 706
  - presentation graphics, PGCHART.LIB, PT 201–202
  - procedures generally, LIB 5–16
  - quick, LINK, ET 590–591
  - QuickWin, PT 145
  - removing name, CL options, ET 553–554
  - routines, calling, LIB 5–6
  - searching for files, LINK, ET 570, 583–584
  - selecting, CL options, ET 527
  - specifying, LINK, ET 566–570
  - standard, defined, ET 816
  - static, defined, ET 816
  - switches, ET 310–312
- libraries field, LINK, ET 567–570
- Library files
  - creating, ET 700, 703
  - PWB, ET 43–44
- Library files, use, LIB 5
- Library Manager. *See* LIB
- Library modules. *See* Object modules
- Library routines
  - calling, LIB 5–6
  - file and path names, LIB 8–9
  - overlaid DOS programs, ET 600
- Library selection, LR 260

LIBRARY statement
  file extension, LINK, ET 566
  module-definition files, ET 609
  segmented files, LINK, ET 564
Lifetime
  global, LR 35, 43–44
  defined, LR+ 25–26
  new operator, object allocated with, LR+ 98
  overview, LR 34–35
  scope. *See* Scope, C++
  table listing, LR 37
  variables, LR 43, 48
_LIGHTBLUE constant, PT 180
_LIGHTCYAN constant, PT 180
_LIGHTGREEN constant, PT 180
_LIGHTMAGENTA constant, PT 180
_LIGHTRED constant, PT 180
_LIGHTYELLOW constant, PT 180
LIM EMS defined, ET 810
LIMITS.H header file
  limits for integer types, LR 15
  range of signed integer values, LR 99
LimitText member function
  CComboBox class, XRF 152
  CEdit class, XRF 294
.line command, HELPMAKE, ET 723
Line concatenation, in #define directives, LR 194
Line continuation character
  NMAKE, ET 655, 660, 669
  PWB, ET 115, 117, 136
Line control, LR 207–208
Line control, C++, LR+ 384–385
#line directives, CL
  adding to output, ET 493–494
  preprocessor-output files, ET 540
Line drawing
  getting mode, _getwritemode function, LIB 392–393
  setting logical mode for, _setwritemode function, LIB 706
  to points, _lineto functions, LIB 453–454
\line formatting code, HELPMAKE, ET 727
Line graphs
  described, PT 203
  sample program, PT 208–212
  styles, PT 204–205
__LINE__ macro
  described, LR 199
  determining value, LR 207
__LINE__ macro, C++, LR+ 375, 384
Line Mode command, PWB, ET 73, 143
Line numbers
  CodeView expressions, ET 400, 418
  retrieving from multiple-line edit control, CEdit::GetLine, XRF 291–292
/LINE option, LINK, ET 582
#line preprocessor directive, LR 190
#line preprocessor directive, C++, LR+ 384–385
Line selection mode, setting, in PWB, ET 241
Line-display mode, setting, CodeView, ET 339
LineFromChar member function, CEdit class, XRF 294
LineIndex member function, CEdit class, XRF 295
LineLength member function, CEdit class, XRF 295–296
LINENUMBERS option, LINK, ET 582
Lines
  blank in NMAKE, command line, ET 660
  deleting, PWB, ET 176–177
  drawing. *See* Line drawing
  drawing from current position, CDC::LineTo, XRF 214
  drawing functions, PT 186, 189–191
  getting from streams, gets function, LIB 376–377
  getting style, _getlinestyle function, LIB 370–371
  inserting, PWB, ET 177–178
  length in edit control, CEdit::LineLength, XRF 295
  multiple statements, debugging, ET 322
  numbers, retrieving from edit control, CEdit::LineFromChar, XRF 294
  pattern control functions, PT 191–192
  setting style, _setlinestyle function, LIB 667
  trailing, display mode, in PWB, ET 301–302
LineScroll member function, CEdit class, XRF 296
linesize pragma
  described, LR 212
  precompiled header compilation, effect on, PT 41
linesize pragma, C++, LR+ 390
_lineto function, LIB 453–454; PT 186, 190
LineTo member function, CDC class, XRF 214
_lineto_w function, LIB 453–454; PT 190
LINK
  case distinction, ET 585
  CL linking options, ET 527–528
  command line, ET 564–568, 570–572
  data segments, loading, ET 579
  debugging, ET 577
  decorated names, ET 790
  defaults, ET 573
  deffile field, ET 570
  environment variable, ET 593–594, 810

LINK (*continued*)
- error bits, clearing, with EXEHDR, ET 631, 634–636
- error messages. *See* "Error Messages," Part 2 of this book
- exefile field, ET 566
- exit codes, ET 596
- export functions, LR 172
- far calls, ET 580–581, 584
- help, ET 581
- interoverlay calls, limiting, ET 579
- invoking, from CL, ET 486
- libraries
  - field, ET 567–570
  - floating-point math, PT 137
- map files, ET 582–583
- mapfile field, ET 567
- module-definition files, overview, ET 600–601, 609
- new features, ET 561–562
- /NOE linking with SETARGV.OBJ, LR 31
- objfiles field, ET 565
- optimization, controlling, PT 25–28
- optimizing relocation table, ET 580
- options
  - debugging considerations, ET 323–325
  - described, ET 575–576
- ordering functions, module-definition files, ET 626–627
- ordering segments, ET 578, 585
- output (.PXE) files, Make P-Code (MPC) utility, PT 53
- output files, ET 563–564
- overlaid programs, ET 597–602
- overview, ET 563
- packing
  - code segments, ET 587–588
  - data segments, ET 588–589
- p-code, PT 53
- prompts, ET 573–576
- PWB menu commands, ET 75
- response files, ET 573–575
- running, ET 572

LINK (*continued*)
- searching
  - for object files, ET 565
  - libraries, ET 583–584
- space allocation, ET 577–578
- syntax, ET 564–568, 570–572, 575
- temporary files, ET 595

LINK environment variable, ET 593–594, 810

/link option, CL, ET 527

LINK Options command, PWB, ET 75

Linkage
- external, LR 44–45
- identifiers, LR 7, 27, 176
- internal, LR 36
- overview, LR 36–37

Linkage, C++
- C functions, LR+ 36–37
- defined, LR+ 25–26
- described, LR+ 33
- extern "C", LR+ 37, 40–41, 179–181; PT 257
- extern "C++", LR+ 37, 179
- external
  - defined, LR+ 25–26
  - described, LR+ 33
- internal
  - defined, LR+ 25–26
  - described, LR+ 33
- rules, LR+ 34–36
- specifications, LR+ 178–181 TUT 23–24;
- types, LR+ 33

LLinker. *See* LINK

Linking
- *See also* LINK
- debugging considerations, ET 323–325
- defined, ET 810, 816
- floating-point math libraries, PT 137
- function-level, LR 260
  - CL options, ET 524
  - enabling, PT 21
- lists of structures, LR 66
- mixed-language programs, PT 237–238
- object files, LR 26
- p-code, PT 53

- Linking (*continued*)
  - topics, HELPMAKE, ET 717–719
  - with SETARGV.OBJ (PWB), LR 31
  - without C run-time startup code, LR 260
- Linking C and C++ modules, TUT 23–24
- Linking libraries, LIB 6
- Link-time possibilities, CL options, ET 508
- Linsert function, PWB, ET 108, 152, 177–178
- List boxes
  - adding filenames to, CListBox::Dir, XRF 360
  - called when control created, CWnd::OnMeasureItem, XRF 754–756
  - called with keyboard input, CWnd::OnVKeyToItem, XRF 795–796
  - CListbox class described, XRF 352
  - comparing items in, CWnd::OnCompareItem, XRF 722–724
  - creating
    - constructor, CListBox::CListBox, XRF 355
    - specifying style, CListBox::Create, XRF 356–359
  - describing deleted item, DELETEITEMSTRUCT structure, XRF 80
  - destroying, XRF 729–730
  - filling with directory listing, CWnd::DlgDirList, XRF 676–677
  - finding specified string, CListBox::FindString, XRF 361
  - getting string from, CListBox::GetText, XRF 365
  - items
    - deleting, CListBox::DeleteString, XRF 359
    - ensuring visibility, CListBox::SetTopIndex, XRF 372
    - removing, CListBox::ResetContent, XRF 367
    - retrieves zero-based index of currently selected, CListBox::GetCurSel, XRF 362
    - retrieving index of first visible, CListBox::GetItemData, XRF 363
    - retrieving index of first visible, CListBox::GetTopIndex, XRF 366
    - retrieving number of, CListBox::GetCount, XRF 361
    - retrieving selection state, CListBox::GetSel, XRF 364
    - searching for match to string, CListBox::SelectString, XRF 367
    - selecting consecutive, CListBox::SelItemRange, XRF 368
    - setting associated 32-bit values, CListBox::SetItemData, XRF 370
    - total selected, CListBox::GetSelCount, XRF 364
- List boxes (*continued*)
  - multicolumn, selecting width, CListBox::SelColumnWidth, XRF 369
  - multiple-selection, selecting strings in, CListGBox::SetSel, XRF 371
  - of combo boxes
    - retrieving current selection, CWnd::DlgDirSelectComboBox, XRF 680–681
    - returning selected items, CComboBox::GetCurSel, XRF 149
  - PWB, ET 81
  - retrieving
    - bounding rectangle dimensions, CListBox::GetItemRect, XRF 363
    - current selection, CWnd::DlgDirSelect, XRF 679–680
    - horizontal scrolling event, CListBox::GetHorizontalExtent, XRF 362
  - returning on application response, CWnd::OnCharToItem, XRF 719–720
  - scrolling
    - selected strings, CListBox::SelCurSel, XRF 369
    - setting width, CListBox::SetHorizonalExtent, XRF 370
  - strings
    - adding, CListBox::AddString, XRF 355
    - getting length, CListBox::GetTextLen, XRF 366
    - getting, CListBox::GetText, XRF 365
    - inserting, CListBox::InsertString, XRF 366
  - supplying identifiers for two items in, COMPAREITEMSTRUCT structure, XRF 77–78
- List classes, XRF 27
- List files defined, ET 810
- List References command, PWB
  - described, ET 76
  - function, ET 145
- List Watch command, CodeView, ET 423
- List Watch Expressions command, CodeView, ET 465–466
- Listing
  - assembly files, CL, ET 501–503, 505
  - C files, PWB, ET 94
  - CodeView, breakpoints, ET 429
  - compiler options, ET 525
  - cross references, in LIB, ET 705–706
  - defined, ET 811
  - expression evaluators, CodeView, ET 370
  - function prototypes, CL option, ET 552–553

- Listing (*continued*)
  - functions, CodeView, ET 435–436
  - help files, ET 333, 772
  - machine-code file, CL, ET 502–505
  - Microsoft Advisor topics, ET 765
  - modules, CodeView, ET 463
  - project files, PWB, ET 43–44
  - references, PWB, ET 102
  - source files, CL, ET 501
  - watch expressions, CodeView, ET 365, 465–466
- listing pragma, precompiled header compilation, effect on, PT 41
- Lists
  - adding element or list to tail, CObList::AddTail, XRF 480–481
  - argument type, LR 51
  - classes described, XRF 27
  - COblist class described, XRF 477
  - CPtrList class described, XRF 519
  - creating, CObList::CObList, XRF 482–483
  - CString objects, CStringList class described, XRF 603
  - elements, XRF 486–487
    - adding after specified position, CObList::InsertAfter, XRF 491
    - adding before specified position, CObList::InsertBefore, XRF 492
    - getting number of, CObList::GetCount, XRF 485
    - head, getting position, CObList::GetHead, XRF 487
    - indicating if empty, CObList::IsEmpty, XRF 493
    - next, getting position, CObList::GetNext, XRF 488
    - previous, getting position, CObList::GetPrev, XRF 489
    - removing all, CObList::RemoveAll, XRF 493
    - removing head, CObList::RemoveHead, XRF 495
    - removing specified, CObList::RemoveAt, XRF 494
    - removing tail, CObList::RemoveTail, XRF 496
    - scanning index for, CObList::FindIndex, XRF 484
    - tail, getting, CObList::GetTail, XRF 490

- Lists (*continued*)
  - elements (*continued*)
    - tail, getting position, CObList::GetTailPosition, XRF 491
    - writing pointer to specified position, CObList::SetAt, XRF 496
  - retrieving pointer to given position, CObList::GetAt, XRF 485
  - scanning index for specified element, CObList::FindIndex, XRF 484
  - searching for first matching CObject pointer, CObList::Find, XRF 483
  - identifiers, LR 97
  - linking, LR 66
  - iteration of, XUG 272
  - objects, deleting, XUG 273
- Literal characters
  - makefiles, NMAKE, ET 653
  - searching, PWB, ET 91
- Literals, C++. *See* Constants, C++
- Live expressions, creating, CodeView, ET 357
- /Ln option, CL, ET 528
- .LNK files, defined, ET 811
- Load command, CodeView, ET 362–364
- Load dialog box, CodeView, ET 363–364
- Load libraries
  - defined, ET 811
  - specifying, LINK, ET 566
- Load Other Files option, CodeView, ET 342
- Load switch, PWB, ET 264, 286
- LoadAccelTable member function, XRF 340; XUG 93
- LoadBitmap member function, CBitmap class, XRF 114
  - CBrush::CreatePatternBrush, XRF 123
  - CWnd::CreateCaret, XRF 665
- LoadCursor member function, CWinApp class, XRF 633, 775
- __loadds keyword, LR 175
  - in calling convention, LR 176
  - naming conventions, CL option, ET 528–529
- __loadds keyword, C++, LR+ 421–422
- Loader, replacing, module-definition files, ET 615

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

LoadIcon member function, CWinApp class, XRF 633–634
  CDC::DrawIcon, XRF 176
  CWnd::OnQueryDragIcon, XRF 775
Loading
  accelerator tables, CFrameWnd::LoadAccelTable, XRF 340
  bitmap resources, CBitmap::LoadBitmap, XRF 114
  child process and executing, _exec functions, LIB 246–250
  data, LINK, ET 579
  menu resources, CMenu::LoadMenu, XRF 430
  object or primitive type from archive, CArchive::operator, XRF 102
  predefined cursor resources
    CWinApp::LoadStandardCursor, XRF 635
    CWinApp::LoadOEMCursor, XRF 634
    icon resource, predefined, XRF 634, 636
    specified cursor resource, CWinApp::LoadCursor, XRF 633
  source files, CodeView, ET 359
  symbolic information, CodeView, ET 342
  virtual memory block into DOS memory and locking, _vlock function, LIB 851–853
  virtual memory blocks, PT 92
  virtual memory blocks into DOS memory _vload function, LIB 848–850
LoadMenu member function, CMenu class, XRF 430
LoadMenuIndirect member function, CMenu class, XRF 431
LoadOEMBitmap member function, CBitmap class, XRF 115–116
LoadOEMCursor member function, CWinApp class, XRF 634
LoadOEMIcon member function, CWinApp class, XRF 634–635
  CDC::DrawIcon, XRF 176
LOADONCALL keyword, module-definition files, ET 621
LoadStandardCursor member function, CWinApp class, XRF 635–636
LoadStandardIcon member function, CWinApp class, XRF 636–637
  CDC::DrawIcon, XRF 176
LoadString member function, CString class, XRF 585–586
Load-time relocation table, optimizing, LINK, ET 580
Local command, CodeView, ET 373–374
Local contexts, help files, linking, ET 720
Local heaps, listing memory objects CodeView, ET 383, 462
Local memory handles, converting to pointers, ET 466–467
Local Options command, CodeView, ET 368–369
Local Options dialog box, CodeView, ET 369
Local scope, C++, LR+ 28
Local symbols
  building a database, BSCMAKE, ET 732, 736
  defined, ET 811
Local variables
  allocations, p-code, PT 52
  C++, initialization, LR+ 47–49
  defined, LR 181
  initializing, LR 35
  listing, CodeView, ET 354, 369
Local window
  CodeView
    function, ET 354
    opening, ET 374
    overview, ET 348
  defined, ET 811
localeconv function, LIB 455–457
Locales
  defining, setlocale function, LIB 668–669
  macros, LIB 65
  settings, getting information on, localeconv function, LIB 455–457
Locality, design principle, TUT 167
LocalLock routine, locking memory handles, ET 386
localtime function, LIB 458–459
Locator values, represented by l-value, LR 107
Locking
  help files, ET 713
  range of bytes in open file, CFile::LockRange, XRF 312
  virtual memory blocks, PT 93
Locking bytes in file, _locking function, LIB 460–462
_locking function, LIB 460–462
LockRange member function, CFile class, XRF 312–313
Locks, returning number held on virtual memory block, _vlockcnt function, LIB 854–856
Log command, PWB
  described, ET 73
  predefined macros, ET 144
log function, LIB 463–464; PT 11
Log Search Complete dialog box, PWB, ET 88
log10 function, LIB 463–464; PT 11
_log10l function, LIB 463–464; PT 11

Logged search, PWB, ET 86, 178, 241–242, 278
Logical AND operator (&&)
  described, LR 135
  sequence points, LR 113
Logical AND operator, C++
  binary-operator expressions, LR+ 111–112
  overloading, LR+ 358
Logical NOT operator, C++
  unary-operator expressions, LR+ 94
Logical operators
  described, LR 135–136
  order of evaluation, LR 114
Logical OR operator, C++
  binary-operator expressions, LR+ 112
  overloading, LR+ 359
Logical palettes. *See* Palettes
Logical segment defined, ET 811
_logl function, LIB 463–464; PT 11
Logsearch function, PWB, ET 152, 178
long double functions, LIB 465
long double type
  described, LR 100
  portability guidelines, PT 273
  supportive functions, PT 130
  unsupported, LR 267
  variables, declaring as, PT 127–129
long double type, C++, LR+ 50–52
Long filenames, NMAKE, ET 654
long int type, C++
  size, LR+ 51–52
Long integer defined, ET 811
Long jumps, MOVE programs, ET 604
long keyword, LR 51
long type
  byte ordering, PT 292–293
  conversion, LR 142
  floating-point constants, LR 11
  forcing, LR 14
long type, C++, LR+ 50–52
longjmp function, LIB 466–467
longjump function, MOVE programs, ET 604
LONG_MAX constant, C++
  integral limits, LR+ 62
LONG_MIN constant, C++
  integral limits, LR+ 62
Lookup member function, CMapStringToOb class, XRF 382
loop_opt pragma, LR 213, 266; PT 18
loop_opt pragma directive, C++, LR+ 390
Loops
  exiting, LR 158
  infinite, ET 387–388; LR 157
  optimizing
    CL options, ET 535–537
    described, PT 18–19
    disabling unsafe, PT 20
  returning to start of, LR 154
Low memory defined, ET 811
/Lp option, CL, LR 267
LPRECT operator, CRect class, XRF 531
LPRECT structure
  CDC::Arc, XRF 166
  CDC::Chord, XRF 171
  CRect::CRect, XRF 524
  CRect::operatorLPRECT, XRF 531
LPtoDP member function, CDC class, XRF 215
/Lr option (CL), ET 528
.LRF files defined, ET 811
_lrotl function, LIB 468
_lrotr function, LIB 468
LRU algorithm, overlaid DOS programs, ET 598
_lsearch function, LIB 469–470
_lseek function, LIB 471–473
.LST files defined, ET 811
_ltoa function, LIB 474–475
/Lu option, CL, LR 260
L-values
  accessing identifiers as, LR 52
  assignment operations, LR 138
  C++, LR+ 60–61
  defined, ET 811
  described, LR 107–108
  Microsoft C extension, LR 108
  prefix increment and decrement operators, LR 122
  primary expressions, LR 106
/Lw option, CL, ET 527; LR 260

M

- :m command, HELPMAKE, ET 723
- m. context prefix, HELPMAKE, ET 729
- /M option
  - CL, setting breakpoints, ET 429–432
  - CodeView, ET 338, 342
  - CVPACK, ET 745
  - LINK, ET 583
  - NMAKE, ET 649
  - PWB, ET 142
- /MA option
  - CL, ET 528
  - EXEHDR, ET 630
- Machine code
  - defined, ET 811
  - mixing with p-code, PT 50
  - transition to p-code, PT 47
  - translating source code, CL, ET 502–505
- Macro Assembler
  - inline assembly. *See* Inline assembly
  - NMAKE options macro, ET 676
- Macros
  - AND_CATCH, XRF 65; XUG 51, 63
  - ANSI, LR 187
  - __asm blocks, defining as, PT 123–124
  - ASSERT, XRF 53
    - Foundation classes cookbook, XUG 288–289
    - Foundation classes tutorial, XUG 33, 127, 138, 203
  - assert, LR+ 43
  - ASSERT_VALID, XRF 53; XUG 32, 126, 138
  - BEGIN_MESSAGE_MAP
    - Foundation classes cookbook, XUG 313–315
    - Foundation classes tutorial, XUG 95
  - benefits over functions, LIB 9–11
  - CATCH
    - exception argument, XUG 63
    - Foundation classes cookbook, XUG 298–302
    - Foundation classes tutorial, XUG 51, 63
  - changing key assignment, PWB, ET 119–121, 135
  - commands, NMAKE, ET 675
  - DEBUG_NEW, XRF 38; XUG 296
  - debugging, LR 196; ET 322
  - DECLARE_DYNAMIC, XRF 38–39, 468; XUG 264–266
  - DECLARE_MESSAGE_MAP
    - Foundation classes cookbook, XUG 313–314, 343, 352
    - Foundation classes tutorial, XUG 95

Macros (*continued*)

  - DECLARE_SERIAL, XRF 39–40, 468
    - Foundation classes cookbook, XUG 281
    - Foundation classes tutorial, XUG 24, 33, 38, 44
  - #define directive, LR+ 368–370
  - defined, ET 811; LIB 9
  - defining, LR 192
    - CL, ET 492–493
    - PWB, ET 157–158
  - diagnostic services, described, XRF 49–52, 55, 57
  - END_CATCH, XRF 66; XUG 51, 63
  - END_MESSAGE_MAP
    - Foundation classes cookbook, XUG 313–315
    - Foundation classes tutorial, XUG 95
  - environment variables, NMAKE, ET 678
  - exception processing, XRF 59, 67
  - exception-handling, XRF 30
  - executing, PWB, ET 106–108, 170
  - expansion, LR 192–193, 197
  - filename, NMAKE, ET 672–673
  - flow control statements, PWB, ET 112–114
  - IMPLEMENT_DYNAMIC, XRF 471; XUG 264–266
  - IMPLEMENT_SERIAL, XRF 471
    - Foundation classes cookbook, XUG 282
    - Foundation classes tutorial, XUG 29, 33, 38, 44
  - #include directive, LR+ 376–378
  - inherited, NMAKE, ET 679–680
  - inheriting, NMAKE, ET 650
  - inline assembly
    - limitations, PT 113
    - using in, PT 115–116
  - inline functions, TUT 16
  - key assignments, PWB, ET 146–150
  - list, XRF 32–33
  - locale, LIB 65
  - name space, LR 192
  - null, NMAKE, ET 670
  - ON_COMMAND
    - Foundation classes cookbook, XUG 315, 317
    - message handlers, XUG 204
  - ON_WM_CHAR, XUG 356
  - ON_WM_KEYDOWN, XUG 356
  - ON_WM_KEYUP, XUG 356
  - ON_WM_NCDESTROY, XUG 323
  - ON_WM_PAINT
    - Foundation classes cookbook, XUG 343
    - Foundation classes tutorial, XUG 101–102
  - options, NMAKE, ET 676
  - overview
    - NMAKE, ET 667–668
    - PWB, ET 109

- Macros (*continued*)
  - precedence, NMAKE, ET 680
  - predefined, LR 198–200
    - __DATE__, LR 198
    - __FILE__, LR 199, 207
    - __LINE__, LR 199, 207
    - PWB, ET 142–146, 222–225, 227–263
    - __STDC__, LR 199
    - __TIME__, LR 199
  - predefined, C++
    - line control, LR+ 384
    - (table), LR+ 374–376
  - predefined identifiers, LR+ 7–9
  - preprocessing, LR+ 366–368
  - recording, PWB, ET 109–112, 203–204, 252–253
  - recursion, NMAKE, ET 674–675
  - redefining, LR 194
  - redefinition, warnings, LR 265
  - RUNTIME_CLASS, XRF 474; XUG 265–266
  - .SBR files, PWB, ET 104
  - shortcut keys, PWB, ET 79
  - side effects, portability guidelines, PT 290–291
  - special, NMAKE, ET 671–672
  - stringizing operator (#), LR 195
  - substitution, NMAKE, ET 677–678
  - THROW, XRF 66; XUG 64
  - THROW_LAST, XRF 67; XUG 64
  - TOOLS.INI syntax, PWB, ET 134
  - TRACE, XRF 56
    - Foundation classes cookbook, XUG 288
    - Foundation classes tutorial, XUG 20–21, 34, 50
  - TRY, XRF 67
    - Foundation classes cookbook, XUG 298–302
    - Foundation classes tutorial, XUG 51, 63
  - #undef directive, LR+ 373–374 undefined
    - NMAKE, ET 670
    - PWB, ET 224, 226
  - user input statements, PWB, ET 114–117
  - user-defined, NMAKE, ET 668–670
  - using, NMAKE, ET 671
  - VERIFY, XRF 57
    - Foundation classes cookbook, XUG 289
    - Foundation classes tutorial, XUG 203
- Macros (*continued*)
  - without arguments, LR 192
  - XENIX, LR 187
- macros field, NMAKE, ET 647
- Magenta, color value, ET 273
- _MAGENTA constant, PT 180
- Main application class, CWinApp, XRF 11
- main function
  - calling conventions, CL options, ET 517
  - errors generated by, LR 266
  - overview, LR 30–34
- main function, C++
  - described, LR+ 38–42
  - initialization considerations, LR+ 43–44
- MainWndProc member function, CWinApp class, Windows applications, writing, XUG 305
- .MAK files
  - *See also* Makefiles
  - defined, ET 811
- Make P-Code (MPC) utility, PT 44, 51–53
- MAKE recursion macro, NMAKE, ET 674
- MAKEDIR recursion macro, NMAKE, ET 674
- Makefiles
  - association with .PIF files, ET 67
  - build process, ET 56, 58
  - building browser database, non-PWB projects, ET 104–105
  - contents, ET 653–654
  - customizing, ET 58–61
  - debug mode builds, XUG 9
  - debugging, ET 648–649
  - defaults, Foundation classes tutorial, XUG 9
  - defined, ET 646
  - dependency lines, ET 655
  - loading, PWB, ET 142
  - locations, XUG 8
  - NMAKE
    - DMTEST, required for, XUG 65
    - filenames, XUG 8, 11
    - Foundation classes tutorial, XUG 7
    - HELLO, required for, XUG 109
  - non-PWB, ET 61–63
  - opening, ET 141

Makefiles (*continued*)
PWB
DMTEST, required for, XUG 65
filenames, XUG 8
Foundation classes tutorial, XUG 7
HELLO, required for, XUG 109
release mode builds, XUG 9
sample, ET 694–696
sequence of operations, ET 692–694
MAKEFLAGS recursion macro, NMAKE, ET 674
MakeLower member function, CString class, XRF 586
_makepath function, LIB 476–478
MakeReverse member function, CString class, XRF 586
MakeUpper member function, CString class, XRF 587
malloc, TUT 63–64, 99
malloc functions, LIB 479–482; PT 83–84
Managing memory. *See* Memory management
Manifest constants, LR 191
Manipulation of strings, Foundation classes cookbook, XUG 256–261
Manipulators, TUT 7
custom, input streams, iostream classes tutorial, XUG 398
derived stream classes, using with iostream classes tutorial, XUG 399
input streams, iostream classes tutorial, XUG 385
ios class, XRF 868–871
istream class, XRF 886
ostream class, XRF 907
output stream, custom, iostream classes tutorial, XUG 381–382
parameters, more than one, iostream classes tutorial, XUG 397
with one parameter, iostream classes tutorial, XUG 395–397
Mantissas
C++ floating-point constants, LR+ 20
floating-point variables, PT 128
Map files
creating
CL, ET 505–507
LINK, ET 582–583
defined, ET 811
/MAP option, LINK, ET 562, 583, 790
MapDialogRect member function, CDialog class, XRF 270
mapfile field, LINK, ET 567
Mapping
device contexts, logical palettes to system palettes, CDC::RealizePalette, XRF 224
fonts, logical to physical, CDC::SetMapperFlags, XRF 241
mode, retrieving current, CDC::GetMapMode, XRF 201
point coordinates, CDC::DPtoLP, XRF 175
setting mode, CDC::SetMapMode, XRF 240–241
Maps
16-bit words keyed to void pointers, XRF 375
classes described, XRF 27
CMapPtrToPtr class described, XRF 373
CMapPtrToWord class described, XRF 375
CMapStringToOb class described, XRF 377
CMapStringToPt class described, XRF 387
CMapStringToString class described, XRF 389
CMapWordToOb class described, XRF 391
CMapWordToPtr class described, XRF 393
CObject pointers keyed by 16-bit words, XRF 391
constructing CString-to-CObject, CMapStringToOb::CMapStringToOb, XRF 379
CString objects keyed to CString objects, XRF 389
CString objects to CObject pointers, XRF 377
defined, XRF 27
elements, deleting, Foundation classes cookbook, XUG 274
finding element with matching key, CMapStringToOb::Lookup, XRF 382
getting number of elements, CMapStringToOb::GetCount, XRF 380
inserting elements
CMapStringToOb::operator[ ], XRF 386
CMapStringToOb::SetAt, XRF 384
iterating through all elements, CMapStringToOb::GetNextAssoc, XRF 380–381
iteration of Foundation classes cookbook, XUG 272
iteration, starting, CMapStringToOb::GetStartPosition, XRF 381
looking up entry corresponding to supplied key, CMapStringToOb::RemoveKey, XRF 383–384
message
See also Message maps
cross-reference, XRF 69
defined, XRF 13
removing elements and destroying CString key objects, CMapStringToOb::RemoveAll, XRF 383
testing if empty, CMapStringToOb::IsEmpty, XRF 382

Maps (*continued*)
  to void pointers keyed by void pointers, XRF 373
  void pointers
    keyed by 16-bit words, XRF 393
    keyed by CString objects, XRF 387
.mark command, HELPMAKE, ET 723
Mark command, QuickWin, PT 148–149
Mark file, PWB menu commands, ET 73
Mark function, PWB, ET 86, 152, 178–179
Markfile switch, PWB, ET 122, 264, 287
Marks
  manipulating, in PWB, ET 178–179
  saving, in PWB, ET 287
Masks
  current radix flag bits, ios::basefield, XRF 867
  file-permission-setting, _umask function, LIB 825–826
  floating-point format flag bits, ios::floatfield, XRF 867
  padding flag bits, ios::adjustfield, XRF 867
MASM
  CL options, ET 528
  debugging assembly language, ET 412–415
  inline assembly. *See* Inline assembly
  radix, ET 444–445
Match case, search option, CodeView, ET 361
Match Case command, PWB, ET 76
Matches, searching, PWB, ET 87–90
Matching
  See also Overloading
  characters, regular expression syntax, ET 779
  regular expressions, ET 307–308, 788
Math
  error handling, _matherr and _matherrl functions, LIB 483–485
  error messages. *See* "Error Messages," Part 2 of this book
  floating-point operations, CL options, ET 508–512
  routines, LIB 44, 46
Math coprocessor floating-point math package
  options; CL, PT 135–136
  described, PT 131
Math coprocessors
  defined, ET 803
  displaying registers, CodeView, ET 355–356
  dumping register contents, ET 473–474
Math packages, floating-point. *See* Floating-point math packages
_matherr function, LIB 483–485; PT 11
_matherrl function, LIB 483–485
__max function, LIB 486
/MAX option, EXEHDR, ET 630
_MAXCOLORMODE constant, PT 169
Maximization, CWnd, determining, CWnd::IsZoomed, XRF 711
Maximize command
  CodeView, ET 373–374
  PWB
    described, ET 77
    predefined macros, ET 145
Maximize function, PWB, ET 152, 179–180
Maximizing
  color, graphics, PT 172
  efficiency, optimization, PT 24–25
  MDI child windows, CMDIChildWnd::MDIMaximize, XRF 399
  resolution, graphics, PT 172
  windows, PWB, ET 179–180, 242
Maximum, returning larger of two values, __max function, LIB 486
_MAXRESMODE constant, PT 169
MAXVAL keyword, module-definition files, ET 617–618
MB_CUR_MAX constant, LIB 65
mblen function, LIB 487–488
MB_LEN_MAX constant, LIB 65; LR 8, 267
MB_LEN_MAX constant, C++
  integral limits, LR+ 62
mbstowcs function, LIB 489–490; LR 8
mbtowc function, LIB 491–493; LR 8
MC command, CodeView, ET 422, 437–438
m_cause data member
  CArchiveException class, XRF 106
  CFileException class, XRF 327–328
  CFileException::ErrnoToException, XRF 324

MD command, CodeView, ET 422, 438–439, 444–445
/MD option, CL, LR 267
MDC command, CodeView, ET 439
MDI
  client window handle, CMDIFrameWnd::m_hWndMDIClient, XRF 410
  client windows, arranging in cascade, CMDIFrameWnd::MDICascade, XRF 406
  window, child, activating, CMDIFrameWnd::MDIActivate, XRF 405
MDI child windows
  deallocating memory, Foundation classes cookbook, XUG 323
  frame windows, matching, Foundation classes cookbook, XUG 324
MDI parent windows
  accessing, Foundation classes cookbook, XUG 323
MDI window classes, Foundation classes cookbook, XUG 323
MDIActivate member function
  CMDIChildWnd class, XRF 398
  CMDIFrameWnd class, XRF 405
    CWnd::OnMDIActivate, XRF 754
MDICascade member function, CMDIFrameWnd class, XRF 406
MDIDestroy member function, CMDIChildWnd class, XRF 399
MDIGetActive member function, CMDIFrameWnd class, XRF 406
MDIIconArrange member function, CMDIFrameWnd class, XRF 406
  CWnd::ArrangeIconicWindows, XRF 659
MDIMaximize member function
  CMDIChildWnd class, XRF 399
  CMDIFrameWnd class, XRF 407
MDINext member function, CMDIFrameWnd class, XRF 407
MDIRestore member function
  CMDIChildWnd class, XRF 399
  CMDIFrameWnd class, XRF 408
MDISetMenu member function, CMDIFrameWnd class, XRF 408–409
MDITile member function, CMDIFrameWnd class, XRF 409
ME command, CodeView, ET 422, 440–441
  and Restart command, ET 436–437
  input radix, ET 444–445
MEASUREITEMSTRUCT structure, XRF 83–84
  CWnd::OnMeasureItem, XRF 754–755
Medium memory models
  defined, ET 811
  null pointers, PT 61–62
  option, CL, PT 63
Megabyte defined, ET 811
_memavl function, LIB 494–495
Member functions
  based addressing, PT 107–109
  called by constructors, TUT 54
  CArchive class, XRF 95–101
  CArchiveException class, XRF 105
  CBitmap class, XRF 109–117
  CBrush class, XRF 119–125
  CButton class, XRF 128–134
  CClientDC class, XRF 138
  CComboBox class, XRF 142–155
  CDataBase class
    CDataBase, XUG 136
  CDC class, XUG 164–260
    SelectObject, XRF 348
  CDialog class, XRF 264, 272
    EndDialog, XUG 164
    OnInitDialog, XUG 165
  CDumpContext class, XRF 275–277
  CEdit class, XRF 285–302
  CFile class, XRF 306–320
    Close, XUG 278
    Open, XUG 277
    Read, XUG 278
    Write, XUG 278
  CFileException class, XRF 324–326
  CFrameWnd class, XRF 338–340
    LoadAccelTable, XUG 93
  CGdiObject class, XRF 344–350
  CListBox class, XRF 355–372
  CMainWindow class
    CMainWindow, XUG 199
    OnAbout, XUG 105–107
    OnAdd, XUG 216, 219
    OnClose, XUG 209–210, 216
    OnCreate, XUG 225–226
    OnDBClose, XUG 209, 216
    OnDelete, XUG 217
    OnDown, XUG 231, 234
    OnEdit, XUG 218, 220
    OnExit, XUG 210, 216
    OnFind, XUG 217, 221
    OnFindAll, XUG 218, 221
    OnHelp, XUG 223
    OnHScroll, XUG 228–229

- Member functions (*continued*)
  - CMainWindow class (*continued*)
    - OnKeyDown, XUG 229, 233
    - OnLButtonDblClk, XUG 232, 235
    - OnLButtonDown, XUG 232, 235, 351, 354–356
    - OnNew, XUG 206, 213
    - OnOpen, XUG 206, 214
    - OnPaint, XUG 101–104, 235–238, 241
    - OnPrint, XUG 210
    - OnRButtonDown, XUG 351
    - OnSave, XUG 215
    - OnSaveAs, XUG 207, 215
    - OnSize, XUG 225–226
    - OnUp, XUG 230, 234
    - OnVScroll, XUG 227, 229
    - Save, XUG 215, 240
  - CMapStringToOb class, XRF 379–385
  - CMDIChildWnd class, XRF 397–399
  - CMDIFrameWnd class, XRF 403–409
  - CMemFile class, XRF 412
  - CMemoryException class, XRF 413
  - CMemoryState class
    - Difference, XUG 292
    - Checkpoint, XUG 292
  - CMenu class, XRF 416–437
  - CMetaFileDC class, XRF 440–442
  - CModalDialog class, XRF 446–448
    - About, XUG 305
    - DoModal, XUG 163, 329
    - OnCancel, XUG 164
    - OnOK, XUG 164
  - CNotSupportedException class, XRF 449
  - CObArray class, XRF 453–462
  - CObject class, XRF 466–476
    - AssertValid, XUG 133, 138, 289
    - Dump, XUG 286–287
    - Serialize, XUG 24, 282–284
  - CObList class, XRF 480–497
  - constructors. *See* Constructors
  - CPaintDC class, XRF 499–500
  - CPalette class, XRF 503–507
  - CPen class, XRF 509–511

- Member functions (*continued*)
  - CPoint class, XRF 513–516
  - CRect class, XRF 523–530
  - CResourceException class, XRF 536
  - CRgn class, XRF 539–550
  - CScrollBar class, XRF 553–557
  - CSize class, XRF 559
  - CStatic class, XRF 563–566
  - CStdioFile class, XRF 568–570
  - CString class, XRF 576–591
    - CString, XUG 164
    - GetBuffer, XUG 260
    - ReleaseBuffer, XUG 260
    - Seek, XUG 278
  - CTime class, XRF 608
  - CTimeSpan class, XRF 620–624
  - CWinApp class, XRF 631–638
    - InitApplication, XUG 305, 308
    - InitInstance, XUG 89, 93, 305
    - MainWndProc, XUG 305
    - OnIdle, XUG 113
  - CWindowDC, XRF 642–643
  - CWnd class, XRF 659–821
    - Invalidate, XUG 214
  - described, LR+ 240–243
  - destructors. *See* Destructors
  - dialog classes, Foundation classes tutorial, XUG 167
  - encapsulation, TUT 54
  - File menu, described, XUG 213–215
  - filebuf class, XRF 832–835
  - friends, declaring as, LR+ 293–294
  - fstream class, XRF 838–844
  - ifstream class, XRF 846–851
  - inline, LR+ 246–247; TUT 52
  - ios class, XRF 855–865
  - iostream class, XRF 873–874
  - Iostream_init, XRF 874
  - istream class, XRF 855–865
    - close, XUG 391
    - get, XUG 386–388
    - getline, XUG 388–389
    - open, XUG 386
    - read, XUG 389

Member functions (*continued*)
istream class (*continued*)
seekg, XUG 390–391
tellg, XUG 390–391
istrstream class, XRF 891–892
istream_withassign, XRF 888
nonstatic, described, LR+ 243
ofstream class, XRF 894–899
bad, XUG 377
clear, XUG 377
close, XUG 376–377
described, XUG 373–374
eof, XUG 377
fail, XUG 377
good, XUG 377
put, XUG 375
rdstate, XUG 377
seekp, XUG 376
tellp, XUG 376
write, XUG 375–376
ostream class, XRF 902–905
ostream_withassign, XRF 909
ostrstream class, XRF 912–914
overloading. *See* Overloading
overview, TUT 42–43, 45–47, 51–52
Phone Book sample program
database, XUG 135
file handling, XUG 136
pure virtual, TUT 127–129
returning references, TUT 55–56
scrolling, adding
Foundation classes tutorial, XUG 227–229
special
described, LR+ 299–300
initialization using, LR+ 325–333
static, LR+ 243–244; TUT 88
stdiobuf class, XRF 916
stdiostream class, XRF 918
streambuf class, XRF 922–938
strstream class, XRF 940–942
strstreambuf class, XRF 944–946
this pointer, LR+ 244–246
type-safe, defining, Foundation classes cookbook, XUG 333
unions, in, LR+ 250
utility
adding, XUG 238–241
CheckForSave, adding, XUG 240
FileDlg, adding, XUG 238–239
InvalidateLine, adding, XUG 241
SetMenu, adding, XUG 240

Member functions (*continued*)
virtual, TUT 120–123
WinMain
Foundation classes cookbook, XUG 305
WM_CREATE
Foundation classes tutorial, XUG 224, 226
WM_SIZE
Foundation classes tutorial, XUG 225–226
Member initializer, TUT 58–60
Member objects
initialization, TUT 58–60
overview, TUT 58
Member variables
CDataBase object
Foundation classes tutorial, XUG 136
extra bytes, adding
Foundation classes cookbook, XUG 327
Members
See also Bit fields; Data members; Member functions
access. *See* Access, C++
arrays, unsized
declaring in member lists, LR+ 239
categories (list), LR+ 235
data storage, LR+ 239
declaration, LR+ 237–238
described, LR+ 235–236
initializing using special member functions, LR+ 329–333
naming restrictions, LR+ 240
pointers to
declarators, LR+ 196–198
types defined, LR+ 55
pointers to, conversion
base to derived, LR+ 76
to base, LR+ 72–73
protected, accessing, LR+ 295–296
Member-selection operator, C++
overloading, LR+ 363
postfix expressions, LR+ 89
Member-selection expressions
described, LR 119–120
operators, LR 108
Memberwise assignment
See also Assignment, C++
described, TUT 69
_memccpy function, LIB 496–497
memchr function, LIB 498–499
memcmp function, LIB 500–502
memcpy function, LIB 503–505
_memicmp function, LIB 506–507

- _memmax function, LIB 508–509
- memmove function, LIB 510–512
- Memory
  - accessing special locations, LR 53
  - adding to heaps, _heapadd functions, LIB 406–409
  - addressing with based pointers, LR 79
  - allocating, LR 253
  - arrays
    - allocating, calloc functions, LIB 131–132
    - using huge, LIB 16
  - availability assumptions, portability guidelines, PT 283–284
  - blocks
    - allocating, _dos_allocmem function, LIB 183–184
    - allocating, _halloc function, LIB 400–401
    - changing size, _expand functions, LIB 255–257
    - deallocating, free functions, LIB 306–308
    - deallocating virtual, _vfree function, LIB 844
    - finding size of largest, _memmax function, LIB 508–509
    - loading into DOS memory, _vload function, LIB 848–850
    - resizeable, XUG 252
    - returning size allocated in heap, _msize function, LIB 529–530
    - virtual, allocating, _vmalloc function, LIB 857–858
    - virtual, loading into DOS memory and locking, _vlock function, LIB 851–853
    - virtual, returning number of locks on, _vlockcnt function, LIB 854–856
    - virtual, returning size of, _vmsize function, LIB 859
    - virtual, unlocking, _vunlock function, LIB 862
  - changing segment size, _dos_setblock function, LIB 216–217
  - CodeView
    - comparing, ET 437–438
    - displaying, ET 356–357
    - dumping data, ET 438–439
    - entering data, ET 440–441
    - filling, ET 441–442
    - moving data, ET 442–443

- Memory (*continued*)
  - CodeView (*continued*)
    - searching, ET 443–444
    - viewing, ET 455–457
  - compaction, specifying time currently spent in, CWnd::OnCompacting, XRF 722
  - configuration. *See Getting Started*
  - conventional. *See Getting Started*
  - corrupted, matching with contents, CMemoryState::Du mpAllObjectsSince, XRF 55
  - diagnostics, XRF 29; XUG 291
  - exceptions, CMemoryException objects, XRF 413
  - expanded. *See Getting Started*; EMS
  - expression evaluator requirements, ET 403–404
  - extended
    - *See also Getting Started*; XMS
    - defined, ET 808
    - Keepmem switch, PWB, ET 285
    - LINK options, ET 591
  - files
    - closing, CMemFile::~CMemFile, XRF 412
    - opening, CMemFile::CMemFile, XRF 412
  - finding leaks, DEBUG_NEW macro, XRF 38
  - format, changing, ET 356–357
  - freeing from fonts, _unregisterfonts function, LIB 833
  - freeing, _hfree function, LIB 422–423
  - getting to store images, _imagesize functions, LIB 426–427
  - handles
    - converting objects to pointers, ET 462–464, 466—467
    - dereferencing, CodeView, ET 386
    - retrieving for multiple-line edit control, CEdit::GetHandle, XRF 291
  - heaps, minimizing, _heapmin functions, LIB 413–414
  - high, defined, ET 809
  - holes, LR 70
  - increasing capacity, CL, ET 491
  - in-memory files, CMemFile class described, XRF 411

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

Memory (*continued*)
  leaks
    CString, avoiding, XUG 304
    DEBUG_NEW macro, Foundation classes cookbook, XUG 296
    detecting, XUG 290–295
    testing CObject objects, XRF 598
  locations
    aliasing, PT 13–18
    pointers. *See* Pointers
  low, detecting, CWnd:OnCompacting, XRF 722
  manager. *See* Memory manager
  management. *See* Memory management
  MOVE library, ET 599
  NMAKE, running, ET 649
  overlaid DOS programs, ET 597
  releasing, _dos_freemem function, LIB 194–195
  returning amount available for allocation, _freect function, LIB 309–310
  returning available, _memavl function, LIB 494–495
  setting amount available to compiler, LR 260
  space allocation
    LINK, ET 577–578
    MOVE, ET 602–603
  stacks, getting available, _stackavail function, LIB 733
  taking snapshot, CMemoryState::Checkpoint, XRF 54
  testing address, AfxIsMemoryBlock, XRF 50
Memory 1 command, CodeView, ET 373–374
Memory 2 command, CodeView, ET 373–374
Memory allocation
  See also Memory
  arrays, freeing extra memory, CObArray::FreeExtra, XRF 455
  blocks, _vmalloc function, LIB 857–858
    functions (list), LIB 60
  C++
    failed, testing for, LR+ 321–323
    new operator, LR+ 318–320
  checking for corrupt guard bytes, AfxCheckMemory, XRF 47
  CObject class, optimizing allocation, CObject::operator new, XRF 475
  controlling heap granularity, _amblksize variable, LIB 61
  deallocating
    blocks, free functions, LIB 306–308
    delete operator, LR+ 323–325
    virtual memory blocks, _vfree function, LIB 844

Memory allocation (*continued*)
  debugging
    forcing program halt on specified sequence numbers, AfxSetAllocStop, XRF 52
    tuning allocation diagnostics, afxMemDF variable, XRF 46
  freeing memory
    CObject::operator delete, XRF 475
    from fonts, _unregisterfonts function, LIB 833
    from heaps, _bfreeseg function, LIB 106
  huge array functions (list), LIB 16
  malloc functions, LIB 479–482
  _memmax function, LIB 508–509
  memory files, XRF 412
  _msize functions, LIB 529–530
  multiple-line edit control, handles, retrieving, CEdit::GetHandle, XRF 291
  preventing memory deletion for strstreambuf object with dynamic array, strstreambuf::freeze, XRF 944
  releasing memory, _dos_freemem function, LIB 194–195
  resizable memory blocks, Foundation classes cookbook, XUG 255
  returning amount available for, _freect function, LIB 309–310
  routines, LIB 46, 48
  setting
    handle to local memory, CEdit::SetHandle, XRF 297
    hook, AfxSetAllocHook, XRF 51
  setting, in EXEHDR, ET 630
  stacks
    _alloca function, LIB 84–85
    _stackavail function, LIB 733
  testing to ensure memory blocks are contained in program's memory space, AfxIsValidAddress, XRF 51
  types, Foundation classes cookbook, XUG 252
  virtual blocks, number of times locked, _vlock function, LIB 851–853
Memory Compare command, CodeView, ET 422, 437–438
Memory Dump Code command, CodeView, ET 439
Memory Dump command, CodeView, ET 422, 438–439, 444–445
Memory Enter command, CodeView, ET 422, 436–437, 440–441
  input radix, ET 444–445
Memory Fill command, CodeView, ET 422, 436–437, 441–442

- Memory handlers, C++, setting, LR+ 321–323
- Memory management
  - C++
    - free store, PT 103–107
    - memory models for classes, PT 97–103
  - CodeView, ET 336
  - described, Foundation classes cookbook, XUG 251–254
  - DOS, ET 733–734
  - frame allocation
    - Foundation classes cookbook, XUG 251
  - heap allocation
    - Foundation classes cookbook, XUG 252
  - pointers. *See* Pointers
  - strategies (list), PT 55
  - virtual memory
    - handles, PT 91–92
    - using, techniques, PT 93–96
  - virtual memory blocks
    - dirty vs. clean, specifying, PT 92
    - loading, PT 92
    - locking and unlocking, PT 93
  - virtual memory manager
    - described, PT 90
    - initializing, PT 90–91
    - terminating, PT 91
- Memory manager
  - initializing virtual, _vheapinit function, LIB 845–846
  - terminating virtual, _vheapterm function, LIB 847
- Memory models
  - ambient
    - described, PT 98–99
    - overriding default, PT 99–100
  - classes, PT 97–103
  - compact
    - option, CL, PT 63
    - null pointers, PT 61–62
  - customizing
    - code pointer sizing, PT 70–71
    - code segments, specifying, PT 77–78
    - options; CL, PT 70
    - data placement, PT 74–76
    - data pointer sizing, PT 71
- Memory models (*continued*)
  - customizing (*continued*)
    - declarations, defining and referencing, PT 74–76
    - library support, PT 74
    - module naming, PT 76–77
    - segment naming, PT 76–77
    - segment setup options, PT 71–74
  - default, PT 56, 62–63
  - defined, ET 811
  - determining pointer default size, LR 60
  - floating-point options, ET 513–514
  - huge
    - option, CL, PT 63
    - described, PT 60–61
  - large, option, CL, PT 63
  - limitations, PT 59
  - medium
    - option, CL, PT 63
    - null pointers, PT 61–62
  - mixed
    - described, PT 63–64
    - functions, declaring, PT 66–68
    - pointer problems, PT 64–65
    - pointer size conversion, PT 68–69
    - variables, declaring, PT 65–66
  - modifiers, C++, LR+ 398–402
  - null pointers, PT 61–62
  - options, CL, PT 62–63, 70, 76–77
  - segmenting, CL options, ET 488–490
  - selecting
    - options, CL, PT 62–63
    - standard six, PT 58
  - small
    - option, CL, PT 63
  - standard six
    - (list), PT 59
    - selecting, PT 58
  - this pointer, overloading, PT 100–101
  - tiny
    - option, CL, PT 63
    - described, PT 60
- Memory Move command, CodeView, ET 422, 442–443

Memory objects
listing, CodeView, ET 382–383, 462
Memory operators
CodeView, ET 405
debugging assembly language, ET 412–415
Memory Search command, CodeView, ET 422
Memory statistics
dumping, Foundation classes cookbook, XUG 293
Memory Window command, CodeView, ET 369
Memory Window Options dialog box
CodeView, ET 369
Memory windows, CodeView
described, ET 327
function, ET 356–357
opening, ET 374
overview, ET 348
saving addresses, ET 344
specifying, ET 455–457
Memory1 Window command, CodeView, ET 368
Memory2 Window command, CodeView, ET 368
memset function, LIB 513–514
Menu bars
CodeView, overview, ET 346
defined, ET 811
PWB
activating, ET 180
screen display, ET 67–68
redrawing, CWnd::DrawMenuBar, XRF 681
Menu commands
adding
PWB, ET 125–127
Browse menu
PWB, ET 76, 145, 200
choosing
PWB, ET 78–79
Data menu
CodeView, ET 364–368
Edit menu
CodeView, ET 361–362
PWB, ET 73
executing
PWB, ET 78–79
File menu
CodeView, ET 358–360
PWB, ET 72, 142
Help menu
CodeView, ET 374
PWB, ET 78, 146
help, getting, ET 764

Menu commands (*continued*)
Options menu
CodeView, ET 368–372
PWB, ET 75
predefined macros
PWB, ET 142–146
Project menu
PWB, ET 74, 144
Run menu
CodeView, ET 362–363
PWB, ET 74, 144
Search menu
PWB, ET 73, 144
Window menu
PWB, ET 77, 145
Windows menu
CodeView, ET 373
Menu items
adding, PWB Run menu, ET 304–306
custom, PWB, ET 260
help, getting, ET 758
HELPMAKE context prefix, ET 729
Menu-command messages, Foundation classes cookbook, XUG 314–315
Menukey function, PWB, ET 152, 180
Menus
bars, redrawing, CWnd::DrawMenuBar, XRF 681
Browse menu
PWB, ET 76, 145, 200
called when about to become active, CWnd::OnInitMenu, XRF 744
calling owner when menu changes, CWnd::OnDrawItem, XRF 732–735
Calls menu
CodeView, ET 372–373
closing
PWB, ET 79
CMenu class described, XRF 414
Control
allowing application access, CWnd::GetSystemMenu, XRF 699
called when Maximize or Minimize button selected, CWnd::OnSysCommand, XRF 788–790
creating
called when item selected, CWnd::OnCommand, XRF 721
empty, CMenu::CreateMenu, XRF 421
pop-up, CMenu::CreatePopupMenu, XRF 421
CWnd
retrieving pointer to, CWnd::GetMenu, XRF 693

Menus (*continued*)
Data menu
CodeView, ET 364–368
deleting items, CMenu::DeleteMenu, XRF 422
destroying
CMenu::~CMenu, XRF 421
specified, CMenu::DestroyMenu, XRF 423
detaching from, CMenu object, CMenu::Detach, XRF 423
Edit menu
CodeView, ET 360
PWB, ET 73, 143
File menu
CodeView, ET 358–360
PWB, ET 72, 142
Help menu
CodeView, ET 374, 757
PWB, ET 78, 146, 757
highlighting, activating or removing from top-level items, CWnd::HiliteMenuItem, XRF 705–706
items
adding, CMenu::InsertMenu, XRF 428–430
appending new, CMenu::AppendMenu, XRF 416–418
associating bitmaps with, CMenu::SetMenuItemBitmaps, XRF 435–436
called when user selects, CWnd::OnMenuSelect, XRF 757–758
changing, CMenu::ModifyMenu, XRF 432–433
copying label to, CMenu::GetMenuString, XRF 427
determining number, CMenu::GetMenuItemCount, XRF 424
enabling, CMenu::EnableMenuItem, XRF 423–424
removing, CCMenu::RemoveMenu, XRF 434
resources, loading and attaching to CMenu object, CMenu::LoadMenu, XRF 430
resources, loading from menu template and attaching to CMenu object, CMenu::LoadMenuIndirect, XRF 431
specifying items to be checked, CMenu::CheckMenuItem, XRF 419–420

Menus (*continued*)
items (*continued*)
specifying position of active, CMenu::GetMenuItemID, XRF 425
specifying status, CMenu::GetMenuState, XRF 425–426
MDI, replacing, CMDIFrameWnd::MDISetMenu, XRF 408–409
menu bars
CodeView, ET 346
mnemonic character, called when user presses, CWnd::OnMenuChar, XRF 756–757
Options menu
CodeView, ET 368–372
PWB, ET 75
pop-up
called when about to become active, CWnd::OnInitMenuPopup, XRF 744–745
check mark control, CMenu::CheckMenuItem, XRF 419–420
creating, CMenu::CreatePopupMenu, XRF 421
determining number of items, CMenu::GetMenuItemCount, XRF 424
displaying floating, CMenu::TrackPopupMenu, XRF 436–437
replacing, CMDIFrameWnd::MDISetMenu, XRF 408–409
retrieving CMenu object, CMenu::GetSubMenu, XRF 428
Project menu
PWB, ET 74, 144
QuickWin. *See* QuickWin
Run menu
CodeView, ET 362–363
PWB, ET 74, 144, 260, 304–306
screen display
PWB, ET 67–68
Search menu
CodeView, ET 361–362
PWB, ET 73, 144
setting current to specified, CWnd::SetMenu, XRF 810

- Menus (*continued*)
  - template, loading resource and attaching to CMenu object, CMenu::LoadMenuIndirect, XRF 431
  - Window menu
    - PWB, ET 77, 145
  - Windows menu
    - CodeView, ET 373
  - Windows, specifying handle to, CMenu::Attach, XRF 419
- Merge command, PWB, ET 72
- Merging operator. *See* Token-pasting operator
- Message boxes, called when about to be drawn, CWnd::OnCtlColor, XRF 726–727
- Message classes
  - CodeView options, ET 429–432
- Message function, PWB, ET 152, 180
- Message handler functions
  - *See also* Member functions
  - CWnd class
    - OnChar, XUG 356
    - OnKeyDown, XUG 356
    - OnKeyUp, XUG 356
    - OnPaint, XUG 343, 345
  - using to modify behavior
    - Foundation classes cookbook, XUG 338
- Message handlers
  - afx_msg prefix
    - Foundation classes tutorial, XUG 203, 205
  - code listings
    - Foundation classes tutorial, XUG 243
  - constructors, adding
    - Foundation classes tutorial, XUG 202
  - creation
    - Foundation classes tutorial, XUG 224–226
    - Phone Book sample program, XUG 199
  - CWnd class defaults
    - Foundation classes tutorial, XUG 203
  - default values
    - Foundation classes tutorial, XUG 100
  - default window procedure, calling
    - Foundation classes cookbook, XUG 319–320
  - described
    - Foundation classes tutorial, XUG 95
  - functions
    - Foundation classes tutorial, XUG 204
  - guidelines and requirements
    - Foundation classes tutorial, XUG 98–99
  - keyboard and mouse
    - described, XUG 234–235
    - interface, adding, XUG 230–233
- Message handlers (*continued*)
  - keyboard and mouse (*continued*)
    - introduced, XUG 199
  - menu commands, adding to
    - Foundation classes tutorial, XUG 199, 205–223
  - message maps
    - adding, XUG 199–202
  - mouse-click messages
    - Foundation classes cookbook, XUG 351
  - naming conventions, XUG 204–205
  - painting
    - Foundation classes tutorial, XUG 199
  - Phone Book sample program, XUG 197
  - planning for
    - Foundation classes tutorial, XUG 197–199
  - scrolling
    - Foundation classes tutorial, XUG 199
  - scrolling member functions, adding
    - Foundation classes tutorial, XUG 227–229
  - sizing
    - Foundation classes tutorial, XUG 224–226
    - Phone Book sample program, XUG 199
  - utility support
    - Foundation classes tutorial, XUG 199
  - WM_PAINT
    - adding, XUG 235–237, 241
- Message loops
  - WinMain function
    - Foundation classes tutorial, XUG 112
- Message maps
  - adding
    - message handlers, for, XUG 199–202
  - BEGIN_MESSAGE_MAP macro
    - Foundation classes tutorial, XUG 95
  - child windows
    - Foundation classes cookbook, XUG 316–317
  - cross-reference, XRF 69
  - DECLARE_MESSAGE_MAP macro, XUG 95
  - described
    - Foundation classes tutorial, XUG 96–101
  - dialog classes, for
    - Foundation classes tutorial, XUG 166
  - END_MESSAGE_MAP macro
    - Foundation classes tutorial, XUG 95
  - files, adding to
    - Foundation classes tutorial, XUG 94–95
  - function categories, XRF 70
  - in modal dialog boxes
    - Foundation classes cookbook, XUG 331

Message maps (*continued*)
- macros
  - BEGIN_MESSAGE_MAP, XUG 313, 315
  - DECLARE_MESSAGE_MAP, XUG 313–314, 352
  - END_MESSAGE_MAP, XUG 313, 315
  - Foundation classes tutorial, XUG 94–97
  - ON_COMMAND, XUG 315, 317
  - ON_WM_CHAR, XUG 356
  - ON_WM_KEYDOWN, XUG 356
  - ON_WM_KEYUP, XUG 356
  - ON_WM_PAINT, XUG 102
- Phone Book sample program, XUG 204
- routing messages, XUG 97
- scrolling
  - Foundation classes cookbook, XUG 322
- v-tables, similarity to, XUG 96
- Windows classes, using in
  - Foundation classes cookbook, XUG 313

Message numbers
- HELPMAKE context prefix, ET 729

message pragma
- precompiled header compilation, effect on, PT 41

message pragma, C++, LR+ 390

!MESSAGE preprocessing directive
- NMAKE, ET 689

Message processing, Foundation classes vs. native Windows, XRF 9

MessageBox member function
- CWnd class, XRF 711–714

Messages
- applications
  - creating and displaying, CWnd::MessageBox, XRF 711–714
- boxes, called with displaying, CWnd::OnCancelMode, XRF 717
- called when CWnd first created, CWnd::OnNcCreate, XRF 762–763
- calling default window procedure
  - CWnd::Default, XRF 673
  - CWnd::DefWindowProc, XRF 674
- control, XRF 14
- cross-reference map, XRF 69

Messages (*continued*)
- CWnd, placing message in queue, CWnd::PostMessage, XRF 800
- determining whether intended for modeless dialog box, CDialog::IsDialogMessage, XRF 268–269
- direct calls to Windows, XRF 16
- idle-time processing, CWinApp::OnIdle, XRF 637
- map defined, XRF 13
- notification, XRF 13–14
- #pragma, LR 212
- providing Windows procedure for, CWnd::WindowProc, XRF 821
- returning pointer to current, CWnd::GetCurrentMessage, XRF 688
- sending to specified control, CWnd::SendDlgItemMessage, XRF 804
- sending to window, CWnd::SendMessage, XRF 805
- used to translate CWinApp window messages, CWnd::PreTranslateMessage, XRF 801
- Windows applications
  - filtering, CWinApp::PreTransl ateMessage, XRF 638
  - providing default loop, CWinApp::Run, XRF 638
- Windows types and class, ET 384–385

Messages, Windows
- BN_CLICKED
  - naming conventions, XUG 205
- idle loop processing
  - Foundation classes cookbook, XUG 309–310
- notification
  - Foundation classes tutorial, XUG 98–99
  - naming conventions, XUG 205
- system-generated
  - Foundation classes tutorial, XUG 99
- user-generated
  - Foundation classes tutorial, XUG 98
- Windows, handling
  - Foundation classes cookbook, XUG 313–319
- WM_COMMAND
  - naming conventions, XUG 204
- WM_CREATE
  - Phone Book sample program, XUG 198
- WM_HSCROLL
  - Phone Book sample program, XUG 198

Messages, Windows (*continued*)
  WM_LBUTTONDBLCLK
    Phone Book sample program, XUG 198
  WM_LBUTTONDOWN
    Phone Book sample program, XUG 198
  WM_PAINT
    Foundation classes tutorial, XUG 88
    Phone Book sample program, XUG 198
  WM_SIZE
    Phone Book sample program, XUG 198
  WM_VSCROLL
    Phone Book sample program, XUG 198
Messaging
    Windows, XRF 8
Meta function, PWB, ET 107–108, 152, 181
Metacharacters
  searching, PWB, ET 91
Metafiles
  closing device context and creating handle to play, CMetaFileDC::Close, XRF 440
  CMetafileCD class described, XRF 438
  playing on given device, CDC::PlayMetaFile, XRF 221
  selecting object into CMetaFileDC, CMetaFileDC::SelectObject, XRF 441
  selecting predefined stock of pens, brushes, fonts, CMetaFileDC::SelectStockObject, XRF 442
MF command, CodeView, ET 422, 436–437, 441–442
Mgrep function
  PWB, ET 87, 152, 181–182
Mgreplist macro, PWB, ET 224, 226
m_hAccelTable data member
  CFrameWnd class, XRF 341
m_hFile data member
  CFile class, XRF 321
m_hInstance data member
  CWinApp class, XRF 639
m_hObject data member
  CGdiObject class, XRF 351
m_hPrevInstance data member
  CWinApp class, XRF 639
m_hWnd data member
  CClientDC class, XRF 138
  CPaintDC class, XRF 500
  CWindowDC class, XRF 643
  CWnd class, XRF 822
m_hWndMDIClient data member
  CMDIFrameWnd class, XRF 410
MI option
  EXEHDR, ET 630
_M_I286 macro, C++, LR+ 375
_M_I386 macro, C++, LR+ 375
_M_I8086 macro, C++, LR+ 375
_M_I86 macro, C++, LR+ 375
_M_I86*m*M macro, C++, LR+ 375
Microsoft Advisor
  copying text, ET 761
  error help, ET 764
  global searches, ET 766–767
  help files
    concatenating, ET 772
    listing, ET 772
    managing, ET 771–773
    opening, ET 765–766
    splitting, ET 773
  Help menus, ET 757
  help, getting, ET 756–765
  hyperlinks, ET 759–761
  index, ET 762
  keyword help, ET 762–763
  menu items, ET 757
  mouse functions, ET 757
  pasting text, ET 761
  Pwbhelp function, ET 198
  structure, ET 755
Microsoft Basic Compiler
  NMAKE command macro, ET 675
  NMAKE options macro, ET 676
Microsoft Browser Database Maintenance Utility. *See* BSCMAKE
Microsoft Browser Information Compactor. *See* SBRPACK
Microsoft C Compiler
  NMAKE command macro, ET 675–676
  NMAKE options macro, ET 676
Microsoft C/C++
  See also Getting Started
  character sets, LR 191
  differences from version 6.0, LR 257
  keywords listed, LR 5
  new features, LR 257–267
  registers, LR 48
Microsoft COBOL Compiler
  NMAKE command macro, ET 675
  NMAKE options macro, ET 676

Microsoft Debugging Information Compactor. *See* CVPACK
Microsoft EXE File Header Utility. See EXEHDR
Microsoft extensions
  casts of l-values, LR 108
  defaults, disabling with /Za option, LR 108
  effect on storage classes, LR 177
Microsoft File Expunge Utility. *See* EXP
Microsoft File Removal Utility. *See* RM
Microsoft File Undelete Utility. *See* UNDEL
Microsoft FORTRAN Compiler
  NMAKE command macro, ET 676
  NMAKE options macro, ET 676
Microsoft Foundation classes. *See* Foundation classes
Microsoft Import Library Manager. *See* IMPLIB
Microsoft Library Manager. *See* LIB
Microsoft Macro Assembler
  CL options, ET 528
  inline assembly. *See* Inline assembly
  NMAKE command macro, ET 675
  NMAKE options macro, ET 676
Microsoft Overlaid Virtual Environment. *See* MOVE
Microsoft Pascal Compiler
  NMAKE command macro, ET 676
  NMAKE options macro, ET 676
Microsoft product support. *See* the Product
Assistance Request Form in LIB
Microsoft Program Maintenance Utility. *See* NMAKE
Microsoft Programmer's Workbench. *See* PWB
Microsoft Relocatable Object-Module Format (OMF), LINK, ET 563
Microsoft Resource Compiler
  NMAKE command macro, ET 676
  NMAKE options macro, ET 676
Microsoft Segmented Executable Linker
  LINK, ET 561
Microsoft Specific margin notation described, LR+ 21
Microsoft Static Overlay Manager, ET 600, 604–605
Microsoft Symbolic Debugging Information. *See* Symbolic Debugging Information
Microsoft Windows
  debugging, ET 377–382
  default key assignments
    PWB, ET 150
Microsoft Word
  rich text format, HELPMAKE, ET 725–726
Mid member function
  CString class, XRF 587–588
_min function, LIB 515
/MIN option
  EXEHDR, ET 630
Minimally formatted ASCII, ET 713, 716, 728
Minimize command
  CodeView, ET 373–374
  PWB
    described, ET 77
    predefined macros, ET 145
Minimize function, PWB, ET 152, 182–183
Minimizing
  CWnd, CWnd::CloseWindow, XRF 664
  heaps, __heapmin functions, LIB 413–414
  windows in PWB, ET 243–244
/MINIMUM option
  CVPACK, ET 745
Minimum, returning smallest of two values, __min function, LIB 515
Minus sign (–)
  options, NMAKE, ET 688–689
Minutes
  getting total, CTimeSpan::GetTotalMinutes, XRF 624
  getting, CTime::GetMinute, XRF 613
  in current hour, getting, CTimeSpan::GetMinutes, XRF 623
m_isDirty member variable
  Foundation classes tutorial, XUG 39
Mixed memory models
  described, PT 63–64
  functions, declaring, PT 66–68
  pointer problems, PT 64–65

Mixed memory models (*continued*)
pointer size conversion, PT 68–69
variables, declaring, PT 65–66
Mixed mode defined, ET 812
Mixed-language programming
addresses, PT 266–267
arrays
declaring and indexing, PT 263–265
passing, PT 263
assembly language
See also Inline assembly, PT 248
described, procedures, PT 248–257
BASIC, PT 240–243
C and C++, TUT 23–24
C++ linkage specification, PT 257
calling conventions, PT 234–235, 258
common blocks, PT 267–269
compiling, PT 237
described, PT 229–231
external data, PT 265–266
FORTRAN, PT 238–240, 243–246
high-level languages, PT 238–240
language conventions, PT 231, 257
language equivalents (table), PT 231
linking, PT 237–238
naming conventions, PT 231–234, 258
parameterpassing requirement, PT 235–236
Pascal, PT 238–240, 246–248
pointers, PT 266–267
QuickBASIC, PT 242
records, PT 265
strings, PT 259–262
structures, PT 265
types, user-defined, PT 265
variable declaration, PT 258–259
_mkdir function, LIB 516–517
_mktemp function, LIB 21, 518–520
mktime function, LIB 521–522
/ML option
CL, LR 267
Mlines function, PWB, ET 152, 183
m_lOsError data member
CFileException class, XRF 328
m_lpCmdLine data member
CWinApp class, XRF 639
MM command, CodeView, ET 422, 442–443
m_msgCur data member
CWinApp class, XRF 639
m_nCmdShow data member
CWinApp class, XRF 640, 819
Mnemonics, assembling, ET 424–426
Modal dialog boxes
Foundation classes tutorial, XUG 162, 165
Model entry
TOOLS.INI file
CodeView, ET 327–332
debugging p-code, ET 389–390
remote debugging, ET 393–395
Modeless dialog boxes
Foundation classes tutorial, XUG 162, 165
Models, memory. *See* Memory models
Modes
addressing
based, PT 58, 88–90
based, member functions, C++, PT 107–109
based, pointers, PT 78–86
far, PT 57
huge, PT 57–58
indirect, portability guidelines, PT 281–283
keywords, PT 64–66
p-code instructions, PT 48
return objects, PT 101–102
this pointer, PT 100–101
v-table pointers, PT 103
near, PT 56–57
indicating by _cpumode variable, LR 259
processor arithmetic
portability guidelines, PT 280–281
specifying
module-definition files, ET 617
text, PT 167–168
video
controlling functions (list), PT 186–188
described, PT 167–168
selecting, PT 173
setting, PT 168–171
Super VGA support, PT 173–175
modf function, LIB 523–524
modfl function, LIB 523–524
Modification flag, setting for edit control, CEdit::SetModify, XRF 298
Modifiers, C++
__based keyword, LR+ 405–412
calling and naming convention, LR+ 415–416
__export keyword, LR+ 419–420
__far keyword, LR+ 403
grammar summary, LR+ 435–436
__huge keyword, LR+ 404–405
__interrupt keyword, LR+ 420–421
(list), LR+ 397–398
__loadds keyword, LR+ 421–422
memory-model, LR+ 398–402

Modifiers, C++ (*continued*)
 __near keyword, LR+ 402–403
 __saveregs keyword, LR+ 422
 __segname function, LR+ 415
Modifying
 external name length default, LR 6
 function declarations, LR 56
 function identifier, LR 87
 identifiers, LR 52
 meanings of declarations, LR 59
 menu items, CMenu::ModifyMenu, XRF 432–433
 pointers, LR 77
 types, LR 55
 variable declarations, LR 56
 viewport extents, CDC::ScaleViewportExt, XRF 228
 window extents, CDC::ScaleWindowExt, XRF 229
ModifyMenu member function
 CMenu class, XRF 432–433
  CWnd::GetSystemMenu, XRF 699
Module-definition (.DEF) files
 building DLL, LR 172
 defined, ET 806, 812
 HELLO sample program
  Foundation classes tutorial, XUG 107–109
 LINK, ET 564, 570, 598, 600–601
 module statements
  adding to executable files, ET 614
  exporting functions, ET 623
  files, specifying, ET 612–613
  heap size, ET 617–618
  importing functions, ET 624–625
  inserting files, ET 627
  inserting text, ET 613–614
  mode, specifying, ET 617
  operating system, specifying, ET 615–616
  ordering functions, ET 626–627
  purpose, ET 609–627
  replacing loader, ET 615
  reserved words, ET 611
  searching DLLs, ET 622
  segment attributes, ET 618–620
  specifying files, ET 611–612
Module-definition (.DEF) files (*continued*)
 module statements (*continued*)
  stack size, ET 617
  syntax, ET 610
 MOVE, ET 598, 604
 new features, ET 607–608
 overview, ET 608
 preparing
  Foundation classes tutorial, XUG 242
 PWB, ET 43
Module Outline command, PWB
 described, ET 76
 function, ET 145
Module statements
 module-definition files
  adding to executable files, ET 614
  exporting functions, ET 623
  files, specifying, ET 612–613
  heap size, ET 617–618
  importing functions, ET 624–625
  inserting files, ET 627
  inserting text, ET 613–614
  mode, specifying, ET 617
  operating system, specifying, ET 615–616
  ordering functions, ET 626–627
  purpose, ET 609–627
  replacing loader, ET 615
  reserved words, ET 611
  searching DLLs, ET 622
  segment attributes, ET 618–620
  specifying files, ET 611–612
  stack size, ET 617
  syntax, ET 610
Modules
 configuring, CodeView, ET 363–364
 defined, ET 812
 listing, CodeView, ET 383, 463
 naming
  CL option, ET 528–530
  custom memory models, PT 76–77
 object
  adding, in LIB, ET 703
  distinguishing, ET 698

- Modulus operator, C++
  - binary-operator expressions, LR+ 103–104
  - overloading, LR+ 358
- Monitors
  - CodeView
    - black-and-white display, line-display mode, redrawing, ET 479
    - screen exchange, ET 341, 343, 371, 445–447, 479
    - specifying, ET 338–339
    - suppressing snow, ET 341
  - PWB, specifying color, ET 271–273
  - remote debugging, ET 396
- Monochrome adapter defined, ET 812
- Months, getting, CTime::GetMonth, XRF 613
- Mouse
  - *See also* Keyboard and mouse
  - capture
    - called by CWnd, CWnd::OnNcHitTest, XRF 763–764
    - retrieving CWnd, CWnd::GetCapture, XRF 685
  - choosing commands, ET 78
  - cursor
    - called on press of button, CWnd::OnMouseActivate, XRF 758–759
    - called when input isn't captured, CWnd::OnSetCursor, XRF 782
    - called when moved, CWnd::OnMouseMove, XRF 760
  - disabling
    - CodeView option, ET 342
  - double-clicking
    - Foundation classes tutorial, XUG 198
  - enabling
    - PWB, ET 288
  - enumerated values (list), XRF 88
  - executing PWB commands, ET 78–79
  - help, getting, ET 756–757
  - hyperlinks, activating, ET 759
  - input
    - causing all subsequent to be sent to current CWnd object, CWnd::SetCapture, XRF 806
    - enabling or disabling, CWnd::EnableWindow, XRF 681–682
    - specifying whether CWnd is enabled for, CWnd::IsWindowEnabled, XRF 710
  - scrolling with
    - Foundation classes tutorial, XUG 228
  - tracking in windows
    - Foundation classes cookbook, XUG 353–356
- Mouse (*continued*)
  - Windows classes, creating, XUG 319
  - windows, handling in
    - Foundation classes cookbook, XUG 351–352
- Mouse button
  - called when clicked over child window, CWnd::OnParentNotify, XRF 774–775
  - tracking pop-up menu item selection, CMenu::TrackPopupMenu, XRF 436–437
- Mouse button, left
  - called when double-clicked, CWnd::OnLButtonDblClk, XRF 748–749
  - called when double-clicked in nonclient area, CWnd::OnNcLButtonDblClk, XRF 765
  - called when pressed in nonclient area, CWnd::OnNcLButtonDown, XRF 765–766
  - called when pressed, CWnd::OnLButtonDown, XRF 749
  - called when released, CWnd::OnLButtonUp, XRF 750
- Mouse button, middle
  - called when double-clicked, CWnd::OnMButtonDblClk, XRF 751
  - called when pressed in nonclient area, CWnd::OnNcMButtonDown, XRF 767–768
  - called when pressed, CWnd::OnMButtonDown, XRF 752
  - called when released in nonclient area, CWnd::OnNcMButtonUp, XRF 768
  - called when released, CWnd::OnMButtonUp, XRF 752–753
- Mouse button, right
  - called when double-clicked within nonclient area, CWnd::OnNcRButtonDblClk, XRF 770
  - called when double-clicked, CWnd::OnRButtonDblClk, XRF 778
  - called when pressed within nonclient area, CWnd::OnNcRButtonDown, XRF 770–771
  - called when pressed, CWnd::OnRButtonDown, XRF 779
  - called when released within nonclient area, CWnd::OnNcRButtonDown, XRF 771
  - called when released, CWnd::OnRButtonUp, XRF 780
- Mouse clicks, simulating in QuickWin menus, PT 163–164
- Mouse pointer defined, ET 812
- Mousemode switch, PWB, ET 264, 288
- MOVABLE keyword, module-definition files, ET 621

- MOVE
  - advantages, ET 605
  - dynamic and static overlays, ET 604–605
  - library, ET 599
  - library routines, ET 600
  - overlaid programs, ET 597–601
    - compatibility, ET 603
    - memory allocation, ET 602–603
    - segments, ET 603–604
- Move command
  - CodeView, ET 373–374
  - LIB, ET 705
  - PWB, ET 77, 145
- _movedate function, LIB 525–526
- _moveinit routine, ET 602–603
- _movepause routine, ET 602–603
- _moveresume routine, ET 602–603
- _movesetcache routine, ET 602–603
- _movesetheap, ET 602
- _moveto function, LIB 527–528; PT 186, 190
- MoveTo member function
  - CDC class, XRF 215–216
- _moveto_w function, LIB 527–528; PT 190
- Movewindow function, PWB, ET 152, 183–184
- MoveWindow member function
  - CWnd class, XRF 714–715
- Moving
  - buffers, memmove and _fmemmove functions, LIB 510–512
  - characters to another segment, _movedate function, LIB 525–526
  - clipping region, CDC::OffsetClipRgn, XRF 216
  - current point position, CDC::MoveTo, XRF 215
  - CWnd, called when, CWnd::OnMove, XRF 760–761
  - file pointers, _lseek function, LIB 471–473
  - files
    - PWB, ET 49
    - RM, ET 748–749
  - graphics position, _moveto functions, LIB 527–528
  - view-coordinate origins, _setvieworg function, LIB 697–698
  - memory blocks
    - CodeView, ET 442–443
- Moving (*continued*)
  - object modules, ET 705
  - rectangles, XRF 527, 533
  - regions, CRgn::OffsetRgn, XRF 547
  - windows, PWB, ET 183–184, 244
- m_pMainWnd data member
  - CWinApp class, XRF 640
- m_pMDIFrameWnd data member
  - CMDIChildWnd class, XRF 400
- m_ps data member, CPaintDC class, XRF 500
- m_pStream data member
  - CStdioFile class, XRF 571
- m_pszAppName data member
  - CWinApp class, XRF 640
- Mpage function, PWB, ET 152, 184
- Mpara function, PWB, ET 152, 184–185
- MPC utility, PT 44, 51–53
  - *See also* Microsoft Pascal Compiler
  - debugging p-code, ET 390
  - error messages. *See* "Error Messages," Part 2 of this book
- /Mq option, ET 528; PT 154–155
- Mreplace function, PWB, ET 152, 185, 289–290
- Mreplaceall function, PWB, ET 152, 185–186
- _MRES16COLOR constant, PT 170
- _MRES256COLOR constant, PT 170
- _MRES4COLOR constant, PT 169
- _MESRESNOCOLOR constant, PT 169
- MS command, CodeView, ET 443–444
- MS32EM87.DLL defined, ET 812
- MS32KRNL.DLL defined, ET 812
- _MSC_VER macro, C++, LR+ 375
- MS-DOS
  - *See also* DOS
  - device names, conventions, XRF 174
- MSDOS macro, C++, LR+ 375
- MSDPMI defined, ET 812
- MSDPMI.EXE
  - *See also Getting Started*
  - CodeView, ET 336
- MSDPMI.IN
  - *See also Getting Started*
  - defined, ET 812

Msearch function
  PWB, 87, PWB, ET 152, 186
Msgdialog switch, PWB, ET 264, 288
Msgflush switch, PWB, ET 264, 289
_msize function, LIB 529–530
_MT macro, C++, LR+ 375
/MT option
  CL, LR 267
Mtlibs switch, PWB, ET 310–311
Multidimensional arrays
  declaring, LR 75
  described, LR 117–119
  using subscript expressions, LR 118
Multimodule programs, PWB
  building, ET 45–46
  compiler options, ET 53–56
  creating projects, ET 42
  editing, ET 47–49, 55
  extending projects, ET 58–61
  non-PWB makefiles, ET 61–63
  overview, ET 41–42
  project contents, ET 43–44
  project dependencies, ET 45, 48
  using existing projects, ET 49
Multiple applications
  debugging, ET 379–382
Multiple document interface. *See* MDI
Multiple inheritance, TUT 132–134
  access control, LR+ 297
  base classes, LR+ 267–271
  described, LR+ 264–265
  names, LR+ 271–274
MULTIPLE keyword
  module-definition files, ET 621
Multiple-line edit control
  character index, retrieving line number, CEdit::LineFromChar, XRF 294
  character line index, retrieving, Edit::LineIndex, XRF 295
  formatting rectangle
    setting, CEdit::SetRect, XRF 299
    setting new dimensions, CEdit::SetRectNP, XRF 300
  line numbers, retrieving, CEdit::GetLine, XRF 291–292
  number of lines, retrieving, CEdit::GetLineCount, XRF 292
  retrieving local memory handle, CEdit::GetHandle, XRF 291
  scrolling text, CEdit::LineScroll, XRF 296
Multiple-line edit control (*continued*)
  setting
    handle to local memory, CEdit::SetHandle, XRF 297
    tab stops, CEdit::SetTabStops, XRF 301
  soft line-break characters, inserting, CEdit::FmtLines, XRF 290
Multiplication assignment operator (*=)
  table listing, LR 138
Multiplication operator, C++
  binary-operator expressions, LR+ 103–104
  overloading, LR+ 358
Multiplicative operators, LR 126
Multitasking operating system defined, ET 812
Mword function, PWB, ET 152, 187

N

\n (escape sequence), new line character, LR 18
:n command, HELPMAKE, ET 724
N command, CodeView, ET 422, 444–445
n. context prefix, HELPMAKE, ET 729
/N option
  CodeView, ET 338, 341–343
  CVPACK, ET 745
  NMAKE, ET 649
/n option
  BSCMAKE, ET 737
Name decorations
  debugging considerations, ET 324, 409
  overview, ET 789–790
  specifying, ET 790–791
  __stdcall calling convention, LR 170
Name spaces
  C++,
    described, LR+ 61
  overloading, LR 40
  overview, LR 39–40
NAME statement
  module-definition files, ET 609–612, 630
Named tags
  TOOLS.INI file
    PWB, ET 133–134
Names
  conflicts, LR 7
  decorated
    overview, ET 789–790
    specifying, ET 790–791
  environment, LR 253
  files, LR 201, 245–246, 250
  functions, LR 28, 46

- Names (*continued*)
  - identifiers, LR 39
  - labels, LR 36
  - p-code segment, LR 260
  - removing, CL option, ET 542–544
  - restricting length, CL option, ET 525
  - restrictions, LR 5–7
  - segment, LR 80
  - structure members, LR 39
  - tags, LR 39
  - typedef, LR 39, 102–103
  - union members, LR 39
- Names, C++
  - ambiguity, LR+ 271–274, 282
  - classes, LR+ 232–234
  - defined, LR+ 25–26
  - dominance, LR+ 273–274
  - enumerators, LR+ 176
  - function arguments, LR+ 283
  - global when, LR+ 282
  - hiding, LR+ 30–32
  - linkage rules
    - no linkage, LR+ 35–36
    - with block scope, LR+ 34
    - with class scope, LR+ 34
    - with file scope, LR+ 34
  - members, restrictions, LR+ 240
  - multiple inheritance ambiguities, LR+ 271–274
  - primary expressions, LR+ 79–80
  - qualified names
    - described, LR+ 282–283
    - primary expressions, LR+ 80
  - scope, LR+ 25–26
  - types. *See* Types
- Naming
  - files, SBRPACK, ET 740
  - libraries, ET 706–707
  - modules, custom memory models, PT 76–77
  - p-code instructions, PT 47–49
  - segments
    - custom memory models, PT 76–77
    - debugging considerations, ET 322
    - described, ET 528–530
- Naming conventions
  - CL options, ET 518
  - Foundation classes cookbook, XUG 286
  - Foundation classes tutorial, XUG 9
  - member variables
    - Foundation classes tutorial, XUG 31, 89
  - message handler functions
    - Foundation classes tutorial, XUG 204–205
  - message maps, macros used in, XUG 205
  - mixed-language programming, PT 231–234, 258
  - modifiers, C++, LR+ 415–416
  - OnPaint function, XUG 103
  - Pascal naming
    - CL option, ET 552
  - segments
    - CL option, ET 528–529
- NAN defined, ET 812
- Native command, CodeView, ET 368, 372, 391–392
- Native entry
  - TOOLS.INI file
    - CodeView, ET 329–336
    - remote debugging, ET 393–395
- Native entry points, p-code functions
  - described, PT 47
  - removing, ET 520–521; PT 51
- Native execution model
  - specifying, CodeView, ET 333
- native_caller pragma, PT 51
  - new in version 7.0, LR 258
  - p-code entry points, removing, ET 520–521
  - precompiled header compilation effect on, PT 41
- native_caller pragma directive, C++, LR+ 390
- Native-code entry points, LR 212
- Navigation
  - CodeView windows, ET 349
  - cursor movement commands, PWB, ET 154–155
  - Microsoft Advisor, ET 756–765
  - QuickSearch, ET 770
  - windows, menu commands, PWB, ET 77
- _ncalloc function, LIB 131–132
- /ND option
  - CL, ET 528–530; LR 175; PT 76–77

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

NDP stack. *See* Numeric data processor stack
/NE option
  EXEHDR, ET 630
Near address defined, ET 812
Near function calls
  translating far calls to, PT 25–26
near functions, PT 64, 66–68
__near keyword, PT 64–68, 239–240
  CL
    /A options, ET 489
    data allocation, ET 523–524
    enabling, ET 550
  default code segment, LR 169
  described, LR 56
  implicit conversions, LR 150
  modifying objects, LR 56
  overriding addressing modes, LR 169
  related to addressing, LR 55
  restrictions, LR 169
  usage with __based keyword, LR 173
__near keyword, C++
  described, LR+ 7, 402–403
  this pointer modification, LR+ 246
Near objects
  accessing, PT 56–57
Near pointers, PT 56–57
Near pointers, C++, LR+ 74
Near variables
  declaring, PT 65–66
Negated set
  regular expression syntax, ET 779
Negation operator, unary, C++
  overloading, LR+ 355
  unary-operator expressions, LR+ 93–94
Negation operators
  described, LR 124
  placement and association, LR 122
Nested structures
  accessing, LR 66
  expanding and contracting, ET 479
Nesting
  classes, LR+ 254–257
  comments, LR 2–3
  else clauses, LR 158–159
  if statements, LR 158–159
  include files, LR 201; LR+ 378
  initializer lists, LR 94
  levels, LR 264
  switch statements, LR 162
  unsized arrays, LR 68
New command
  Phone Book sample program, XUG 213
New command, PWB, ET 72, 77
  predefined macros, ET 142–143, 145
New features in version 7.0. *See Getting Started*
New handlers, C++, LR+ 321–323
New line
  starting
    PWB, ET 168
New line character
  \n (escape sequence), LR 18
new operator, C++, PT 103–105
  array allocation, TUT 66, 95
  CObject class, XRF 475
  class scope, TUT 103–104, 106
  dynamic allocation, LR+ 318–320
  Foundation classes cookbook, XUG 296
  Foundation classes tutorial, XUG 31, 41, 58, 88
  overloading, TUT 101–102
  overview, TUT 64–65
  unary-operator expressions, LR+ 97–101
New Project command, PWB, ET 74
Newfile function, PWB, ET 152, 187–188
NEWFILES keyword
  NAME statement, module-definition files, ET 611–612
/NEWFILES option
  EXEHDR, ET 630
Newline character
  defined, ET 812
  module statement syntax, ET 610
Newline function, PWB, ET 152, 188
Newwindow switch, PWB, ET 264, 289–290
_nexpand function, LIB 255–257
.next command
  HELPMAKE, ET 723
Next command, PWB, ET 72, 76, 78
  function, ET 145
  predefined macros, ET 142–143, 146
Next Error command, PWB, ET 74
  multimodule builds, ET 46
  predefined macros, ET 144
Next Match command, PWB
  described, ET 73
  predefined macro, ET 144
  searching, ET 89
NextDlgCtrl member function
  CDialog class, XRF 270
Nextmsg function, PWB, ET 152, 188–189, 289–290

Nextsearch function, PWB, ET 152, 189–190, 289–290
_nfrealloc function, LIB 613–615
_nfree function, LIB 306–308
_nheapchk function, LIB 410–412
_nheapmin function, LIB 413–414
_nheapset function, LIB 415–417
_nheapwalk function, LIB 418–421
/NM option
  CL, ET 528–530
/NM option, CL, PT 76–77
NMAKE
  building programs
    Foundation classes tutorial, XUG 11
  building projects, ET 45
    PWB, ET 58–60
  command file, ET 650–651
  command line, ET 647
    command file, ET 650–651
    macros, defining, ET 669–670
    suppressing, ET 688
  commands
    exit codes, ET 662–664
    inline files, ET 664–667
    macros, ET 675
    modifiers, ET 661–662
    syntax, ET 660
  debug mode, XUG 12
  description blocks
    dependency lines, ET 655
    dependents, ET 659–660
    overview, ET 655
    targets, ET 656–658
  directives
    dot, ET 687–688
    preprocessing, ET 688, 690–692
  DOS programs, building, XUG 13
  error messages. *See* “Error Messages,” Part 2 of this book
  exit codes, ET 696
    commands, ET 662–664
    ignoring, ET 649, 687
NMAKE (*continued*)
  inference rules
    inferred dependents, ET 685–686
    overview, ET 680–681
    precedence, ET 686–687
    predefined, ET 684–685
    search paths, ET 682
    syntax, ET 681–682
    user-defined, ET 682–684
  macros
    commands, ET 675
    environment variables, ET 678
    filename, ET 672–673
    inherited, ET 679–680
    null, ET 670
    options, ET 676
    overview, ET 667–668
    precedence, ET 680
    recursion, ET 674–675
    special, ET 671
    substitution, ET 677–678
    undefined, ET 670
    user-defined, ET 668–670
    using, ET 671
  makefiles
    contents, ET 653–654
    DMTEST sample program, XUG 65
    filenames, XUG 8, 11
    Foundation classes tutorial, XUG 7
    HELLO sample program, XUG 109
    PWB, ET 62–63
  new features, ET 645
  options
    macros, ET 676
    turning on, ET 688–689
  overview, ET 646
  PWB, using, Foundation classes tutorial, XUG 9
  release mode, XUG 12
  running, ET 647
  sample makefile, ET 694–696
  sequence of operations, ET 692–694
  TOOLS.INI file, ET 652
  using, Foundation classes tutorial, XUG 11

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User’s Guide

NMAKE Options command, PWB, ET 75
_nmalloc function, LIB 479–482
NMD1PCD.DCC, ET 389–390
NMWOPCD.DCC, ET 389–390
/NO option
  EXEHDR, ET 630
NO87 environment variable
  CL, ET 513
NO87 environment variable, floating-point math, PT 138
/NOD option
  LINK, ET 583–584; PT 137
NODATA keyword
  module-definition files, ET 623
/NODEFAULTLIBRARYSEARCH option
  LINK, ET 583–584; PT 137
/NOE option
  LIB, ET 701
  LINK, ET 584; LR 31
Noedit function, PWB, ET 152, 190
/NOEXTDICTIONARY option
  LIB, ET 701
  LINK, ET 584
NO_EXT_KEYS macro, C++, LR+ 375
/NOF option
  LINK, ET 584
/NOFARCALL option
  LINK, ET 584
/NOFARCALLTRANSLATION option
  LINK, ET 584
/NOG option
  LINK, ET 584
/NOGROUP option
  LINK, ET 584
/NOGROUPASSOCIATION option
  LINK, ET 584
/NOI option
  IMPLIB, ET 747
  LIB, ET 701
  LINK, ET 585
/NOIGNORECASE option
  IMPLIB, ET 747
  LIB, ET 701
  LINK, ET 585
Noise switch, PWB, ET 264, 290
NOKEEP
  inline files
    NMAKE, ET 665
/NOL option
  IMPLIB, ET 747
  LIB, ET 701
  LINK, ET 585, 594
/NOLOGO option
  BSCMAKE, ET 737
  CL, ET 530
  CVPACK, ET 745
  EXEHDR, ET 630
  HELPMAKE, ET 715
  IMPLIB, ET 747
  LIB, ET 701
  LINK, ET 585, 594
  NMAKE, ET 649
  SBRPACK, ET 740
/NON option
  LINK, ET 585
Nonclient areas
  calculating size, CWnd::OnNcCalcSize, XRF 762
  called when destroyed, CWnd::OnNcDestroy, XRF 763
  called when mouse button pressed in, CWnd::OnNcLButtonDown, XRF 765–766
  called when needing painting, CWnd::OnNcPaint, XRF 769
NONDISCARDABLE keyword
  module-definition files, ET 620
NONE keyword
  module-definition files, ET 621
Nonmaskable-interrupt
  CodeView option, ET 341–342
Nonmaskable-Interrupt Trapping option
  CodeView, ET 343
NONSHARED keyword
  module-definition files, ET 622
Nonstatic member functions
  described, LR+ 243
Nonsystem key
  called when pressed, CWnd::OnKeyDown, XRF 745–746
  called when released, CWnd::OnKeyUp, XRF 746–747
/NONULLS option
  LINK, ET 585
/NONULLSDOSSEG option
  LINK, ET 585
Non-UNIX predefined expressions
  syntax, ET 780

- Non-UNIX regular expressions
  - matching method, ET 788
  - syntax, ET 303–304, 780, 786
- /NOPACKC option
  - LINK, ET 562, 586
- /NOPACKCODE option
  - LINK, ET 562, 586
- /NOPACKF option
  - LINK, ET 562, 586; PT 28
- /NOPACKFUNCTIONS option
  - LINK, ET 562, 586; PT 28
- Normalizing, LR 149
- NOT operator, logical, C++. *See* Logical NOT operator, C++
- Notation in expressions, C++, LR+ 130–131
- Notification messages
  - naming conventions
    - Foundation classes tutorial, XUG 205
- /NQ option
  - CL, ET 528–530; LR 260
- _nstrdup function, LIB 753–754
- /NT option
  - CL, ET 528–530; LR 169
- /NT option, CL, PT 76–77
- NUL, CL options, appending to, ET 497–498
- Null character, LR 249
  - defined, ET 812
- Null macros, LR 247
  - NMAKE, ET 670
- Null pointer
  - defined, ET 812; LR 133
  - if constant zero, LR 149
  - invalidates pointer value, LR 123
  - memory models, using with, PT 61–62
  - portability guidelines, PT 282
  - produced by conversions, LR 146
- Null pointer, C++, TUT 99
  - conversion
    - from integral constant expressions, LR+ 75
    - from null values, LR+ 71
  - delete operator, TUT 65
  - new operator, TUT 65
- Null preprocessor directive, C++, LR+ 384
- Null statements
  - described, LR 159–160
  - empty, LR 152, 155
- Null statements, C++, LR+ 136–137
- Null strings, user-defined macros, NMAKE, ET 668
- Number sign (#)
  - custom builds, ET 59
  - HELPMAKE syntax, ET 712–713
  - inference rules, NMAKE, ET 681
  - makefile comments, NMAKE, ET 654
  - preprocessing directives, using in, LR 24
  - substituting for equal sign, CL, ET 492
  - Tab Set command, CodeView, ET 423, 470
  - TOOLS.INI file syntax, ET 652
  - usage, LR 209
  - user-defined macros, NMAKE, ET 669
- Numbers
  - converting double to strings, _ecvt function, LIB 239–240
  - predefined expression syntax, ET 778, 785
  - pseudorandom, generating, rand function, LIB 609–610
  - real, computing from mantissa and exponent, ldexp and _ldexpl functions, LIB 447–448
  - signed real, LR 10
- Numeric arguments
  - LINK, ET 576
- Numeric constants
  - CodeView expression evaluators, ET 407–408
- Numeric data processor stack
  - floating-point return values, PT 130
- Numeric switches
  - PWB, ET 122
- Numerical limits, C++
  - floating, LR+ 63–64
  - integral, LR+ 62
- /NV option
  - CL, ET 528–530; LR 260

O

- O command, CodeView, ET 422, 445–448
- /O option
  - CL, ET 530–531, 539; LR 212

- /O option (*continued*)
  - HELPMAKE, ET 712, 714
- /o option
  - BSCMAKE, ET 733, 737
- O3 command, CodeView, ET 445–447
- OA command, CodeView, ET 347, 445–447
- /Oa option
  - CL, ET 530–532; PT 13–18
- OB command, CodeView, ET 445–447
- /Ob option
  - CL, ET 530, 532; LR 260
- /Ob0 option, CL, PT 13
- /Ob1 option, CL, PT 13
- /Ob2 option
  - CL, LR 210–211, 258; PT 13
- Object context
  - dumping
    - Foundation classes cookbook, XUG 286
- Object diagnostics
  - class-oriented, XRF 24
  - described, XRF 24
  - dump context, XRF 24
  - validity checking, XRF 25
- Object dump
  - interpreting
    - Foundation classes cookbook, XUG 294–295
- Object module format defined, ET 812
- Object modules
  - adding, ET 703
  - copying, ET 705
  - defined, ET 698, 812
  - deleting, ET 704
  - moving, ET 705
  - replacing, ET 704–705
- Object names, C++
  - introduction by declaration statements, LR+ 149
- Object (OBJ.) files
  - defined, ET 812
  - linking, LR 26
  - linking with library files, LIB 6
  - object modules, distinguishing, ET 698
  - output files, LINK, ET 563–564
  - overview, ET 563
  - precompiled headers, PT 38–39
  - PWB, ET 43
  - renaming, CL, ET 498
  - specifying, LINK, ET 565
- Object persistence described, XRF 23
- Object-Module Format
  - LINK, ET 563
- Object-oriented decomposition, TUT 172
- Object-oriented design
  - example, TUT 181–182
  - principles, TUT 161, 172–173
- Object-oriented programming
  - data-access functions
    - Foundation classes tutorial, XUG 32
  - design concepts, TUT 161
- Objects
  - adding to a list
    - Foundation classes tutorial, XUG 41
  - address space
    - far objects, PT 57
    - near objects, PT 56–57
  - afxDump
    - Foundation classes cookbook, XUG 287
  - automatic, LR 44
  - based, LR+ 411–414
  - C++
    - return, addressing modes, specifying, PT 101–102
    - v-table pointers, PT 103
  - CArchive class
    - CArchive, XUG 61
    - creating, XUG 284
  - CDataBase class
    - member variables, XUG 136
  - CFile class
    - Foundation classes tutorial, XUG 60
  - CFileException class
    - Foundation classes tutorial, XUG 60
  - class-type
    - described, LR+ 230–231
  - CModalDialog class
    - Foundation classes tutorial, XUG 163
    - Phone Book sample program, XUG 151
  - CObject class described, XRF 463
  - converting types, LR 126
  - copying, LR+ 333–337
  - CPersonList class
    - Foundation classes tutorial, XUG 139
    - Phone Book sample program, XUG 134
  - creating, CObject::CObject, XRF 467
  - creation and destruction, TUT 49–50
  - CString class
    - exceptions, XUG 304
  - data model, XUG 19
  - declaration, TUT 43
  - declaring, TUT 48
    - as automatic, LR+ 149–151
    - as static, LR+ 152–154

Objects (*continued*)
- defined, LR+ 21, 25–26; TUT 43
- deserialization of
  - Foundation classes cookbook, XUG 285
- destroying, CObject::~CObject, XRF 467
- dialog
  - Foundation classes tutorial, XUG 165
- dumping
  - Foundation classes cookbook, XUG 294
  - to CObject objects, CObject::Dump, XRF 469–470
- externally linked
  - multiple declarations, LR 43
- far, LR 56
- getting run-time structure, CObject::GetRunTimeClass, XRF 470
- global
  - Foundation classes tutorial, XUG 111
- huge, LR 57
- initializing, LR+ 218
- internally linked, LR 43
- lifetime,, LR+ 25–26; TUT 49–50
- linking, LR 267
- modifying with __near, __far, __huge and __based, PT 65–66
- passing by reference, return types, LR+ 53–54
- pointers to, modifying with __near, __far, __huge and __based, PT 65–66
- reading or writing to archive, CObject::Serialize, XRF 473–474
- serialization
  - Foundation classes cookbook, XUG 283–285
  - Foundation classes tutorial, XUG 44
- static
  - Foundation classes tutorial, XUG 111
  - initializing using special member functions, LR+ 329
- storage class
  - defined, LR+ 25–26
- temporary
  - described, LR+ 311–312
- testing
  - for class, CObject::IsKindOf, XRF 472
  - if eligible for serialization, CObject::IsSerializable, XRF 473

Objects (*continued*)
- type cast, LR 148
- type conversions, LR+ 312–313
- validity checking, CObject::AssertValid, XRF 466
- variables, compared to, LR+ 25–26
- void, LR 148
- Windows graphics, XUG 346

objfiles field, LINK, ET 565

OC command, CodeView, ET 445–447

/Oc option
- CL, ET 530, 533

/Oc option, CL, PT 23

Octal
- character specifications, LR 19
- escape sequences, LR 18
- *See* integer constants

/Od option
- CL, ET 530, 533
  - debugging considerations, ET 324

/Oe option
- CL, ET 531, 533; PT 21–22, 125

OEM,converting characters to ANSI character set, CString::OemToAnsi, XRF 588

OemToAnsi member function
- CString class, XRF 588

OF command, CodeView, ET 445–447

/Of option
- CL, ET 531, 533–534; LR 261
- debugging p-code, ET 390

/Of- option, CL, ET 531

Offset
- *See also* Based pointers
- based addressing, LR 121
- pointer with type __based (void), LR 121
- relational operators, LR 132–133

Offset member function
- CPoint class, XRF 514

OFFSET operator
- MASM, ET 414

Offset defined, ET 812

OffsetClipRgn member function
- CDC class, XRF 216

OffsetRect member function
  CRect class, XRF 527–528
OffsetRgn member function
  CRgn class, XRF 547–548
OffsetViewportOrg member function
  CDC class, XRF 217
OffsetWindowOrg member function
  CDC class, XRF 217
ofstream class
  described, XRF 893
    iostream classes tutorial, XUG 366
  flags
    iostream classes tutorial, XUG 374–375
  member functions
    attach, XRF 894
    bad, XUG 377
    clear, XUG 377
    close, XRF 894; XUG 376–377
    described, XUG 373–374
    eof, XUG 377
    fail, XUG 377
    fd, XRF 895
    good, XUG 377
    is_open, XRF 895
    ofstream, XRF 895–897
    ~ofstream, XRF 897
    open, XRF 898; XUG 374
    put, XUG 375
    rdbuf, XRF 898
    rdstate, XUG 377
    seekp, XUG 376
    setbuf, XRF 899
    setmode, XRF 899
    tellp, XUG 376
    write, XUG 375–376
ofstream constructor, XRF 895–897
ofstream destructor, XRF 897
ofstream objects
  creating, ofstream::ofstream, XRF 896–897
  destroying, fstream::~fstream, XRF 841
  destroying, ofstream::~ofstream, XRF 897
/Og option
  CL, ET 531, 533; PT 23, 125
OH command, CodeView, ET 445–447
/Oi option
  CL, ET 531, 534–535; LR 211, 258; PT 10–12
OK button, overriding in dialog boxes,
    CModalDialog::OnOk, XRF 448
OL command, CodeView, ET 445–447
/OL option
  LINK, ET 586
/Ol option
  CL, ET 531, 535–536; PT 18–20, 125
OLD statement
  module-definition files, ET 609, 622
/OLDOVERLAY option
  LINK, ET 562, 586
ON command, CodeView, ET 445–447
/On option
  CL, ET 531
OnAbout member function
  CMainWindow class
    Foundation classes tutorial, XUG 105–107
OnActivate member function
  CWnd class, XRF 715
OnActivateApp member function
  CWnd class, XRF 716
OnAdd member function
  CMainWindow class
    adding to menus, XUG 216
    described, XUG 219
OnAskCbFormatName member function
  CWnd class, XRF 716–717
OnCancel member function
  CModalDialog class, XRF 448
    Foundation classes cookbook, XUG 331
    Foundation classes tutorial, XUG 164
OnCancelMode member function
  CWnd class, XRF 717
OnChangeCbChain member function
  CWnd class, XRF 717–718
OnChar member function
  CWnd class, XRF 718–719
    Foundation classes cookbook, XUG 356
OnCharToItem member function
  CWnd class, XRF 719–720
OnChildActivate member function
  CWnd class, XRF 720
OnClose member function
  CMainWindow class
    described, XUG 216
    adding to File menu, XUG 209–210
ON_COMMAND macro
  CWnd class, XRF 720
  Foundation classes cookbook, XUG 315, 317
  message handlers
    Foundation classes tutorial, XUG 204
OnCommand member function
  CWnd class, XRF 721
OnCompacting member function
  CWnd class, XRF 722

OnCompareItem member function
CWnd class, XRF 722–724
OnCreate member function
CMainWindow class
creation and sizing message handlers, XUG 225–226
described, XUG 226
CWnd class, XRF 724–726
OnCtlColor member function
CWnd class, XRF 726–727
OnDBClose member function
CMainWindow class
adding to File menu, XUG 209
described, XUG 216
OnDeadChar member function
CWnd class, XRF 727–729
OnDelete member function
CMainWindow class
adding to menus, XUG 217
OnDeleteItem member function
CWnd class, XRF 729–730
OnDestroy member function
CWnd class, XRF 730
OnDestroyClipboard member function
CWnd class, XRF 731
OnDevModeChange member function
CWnd class, XRF 731
OnDown member function
CMainWindow class
adding, XUG 231
described, XUG 234
OnDrawClipboard member function
CWnd class, XRF 731–732
OnDrawItem member function
CWnd class, XRF 732–735
One's-complement arithmetic
portability guidelines, PT 280–281
One-byte instructions, p-code (table), PT 297–299
One-dimensional arrays, LR 116–117
OnEdit member function
CMainWindow class
adding to menus, XUG 218
described, XUG 220
OnEnable member function
CWnd class, XRF 735–736
OnEndSession member function
CWnd class, XRF 736
OnEnterIdle member function
CWnd class, XRF 737
OnEraseBkgnd member function
CWnd class, XRF 737–738
One's complement operator, C++
unary-operator expressions, LR+ 94
_onexit function, LIB 531–532
OnExit member function
CMainWindow class
adding to File menu, XUG 210
described, XUG 216
OnFind member function
CMainWindow class
adding to menus, XUG 217
described, XUG 221
OnFindAll member function
CMainWindow class
adding to menus, XUG 218
described, XUG 221
OnFontChange member function
CWnd class, XRF 738–739
OnGetDlgCode member function
CWnd class, XRF 739–740
OnGetMinMaxInfo member function
CWnd class, XRF 740–741
OnHelp member function
CMainWindow class
adding to Help menu, XUG 223
described, XUG 223
OnHScroll member function
CMainWindow class
adding, XUG 228
described, XUG 229
CWnd class, XRF 741–742
OnHScrollClipboard member function
CWnd class, XRF 742–743
OnIconEraseBkgnd member function
CWnd class, XRF 743

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

OnIdle member function
CWinApp class, XRF 637
Foundation classes cookbook, XUG 309–310
Foundation classes tutorial, XUG 113
overridable member function, XRF 11
OnInitDialog member function
CDialog class, XRF 271
Foundation classes cookbook, XUG 330, 332
Foundation classes tutorial, XUG 165
OnInitMenu member function
CWnd class, XRF 744
OnInitMenuPopup member function
CWnd class, XRF 744–745
OnKeyDown member function
CMainWindow class
adding, XUG 233
described, XUG 229
CWnd class, XRF 745–746; XUG 356
OnKeyUp member function
CWnd class, XRF 746–747; XUG 356
OnKillFocus member function
CWnd class, XRF 747–748
OnLButtonDblClk member function
CMainWindow class
adding, XUG 232
described, XUG 235
CWnd class, XRF 748–749
OnLButtonDown member function
CMainWindow class
adding, XUG 232
described, XUG 235
CWnd class, XRF 749
Foundation classes cookbook, XUG 351, 354–356
OnLButtonUp member function
CWnd class, XRF 750
OnMButtonDblClk member function
CWnd class, XRF 751
OnMButtonDown member function
CWnd class, XRF 752
OnMButtonUp member function
CWnd class, XRF 752–753
OnMDIActivate member function
CWnd class, XRF 753–754
OnMeasureItem member function
CWnd class, XRF 754–756
OnMenuChar member function
CWnd class, XRF 756–757
OnMenuSelect member function
CWnd class, XRF 757–758
OnMouseActivate member function
CWnd class, XRF 758–759
OnMouseMove member function
CWnd class, XRF 760
OnMove member function
CWnd class, XRF 760–761
OnNcActivate member function
CWnd class, XRF 761
OnNcCalcSize member function
CWnd class, XRF 762
OnNcCreate member function
CWnd class, XRF 762–763
OnNcDestroy member function
CWnd class, XRF 763
OnNcHitTest member function
CWnd class, XRF 763–764
OnNcLButtonDblClk member function
CWnd class, XRF 765
OnNcLButtonDown member function
CWnd class, XRF 765–766
OnNcLButtonUp member function
CWnd class, XRF 766
OnNcMButtonDblClk member function
CWnd class, XRF 767
OnNcMButtonDown member function
CWnd class, XRF 767–768
OnNcMButtonUp member function
CWnd class, XRF 768
OnNcMouseMove member function
CWnd class, XRF 769
OnNcPaint member function
CWnd class, XRF 769
OnNcRButtonDblClk member function
CWnd class, XRF 770
OnNcRButtonDown member function
CWnd class, XRF 770–771
OnNcRButtonUp member function
CWnd class, XRF 771
OnNew member function
CMainWindow class
adding to File menu, XUG 206
described, XUG 213
OnOK member function
CModalDialog class, XRF 448
Foundation classes tutorial, XUG 164
OnOpen member function
CMainWindow class
adding to File menu, XUG 206
described, XUG 214
deserialization, XUG 207

- OnPaint member function
  - CMainWindow class
    - adding, XUG 235–237, 241
    - described, XUG 101–104, 237–238
    - Windows graphics, XUG 343, 345
  - CWnd class, XRF 772
- OnPaintClipboard member function
  - CWnd class, XRF 772–773
- OnPaintIcon member function
  - CWnd class, XRF 773
- OnPaletteChanged member function
  - CWnd class, XRF 773–774
- OnParentNotify member function
  - CWnd class, XRF 774–775
- OnPrint member function
  - CMainWindow class
    - adding to File menu, XUG 210
- OnQueryDragIcon member function
  - CWnd class, XRF 775–776
- OnQueryEndSession member function
  - CWnd class, XRF 776
- OnQueryNewPalette member function
  - CWnd class, XRF 777
- OnQueryOpen member function
  - CWnd class, XRF 777
- OnRButtonDblClk member function
  - CWnd class, XRF 778
- OnRButtonDown member function
  - CWnd class, XRF 779
    - Foundation classes cookbook, XUG 351
- OnRButtonUp member function
  - CWnd class, XRF 780
- OnRenderAllFormats member function
  - CWnd class, XRF 780–781
- OnRenderFormat member function
  - CWnd class, XRF 781
- OnSave member function
  - CMainWindow class, XUG 215
- OnSaveAs member function
  - CMainWindow class
    - adding to File menu, XUG 207
    - described, XUG 215
- OnSetCursor member function
  - CWnd class, XRF 782
- OnSetFocus member function
  - CWnd class, XRF 783
- OnSetFont member function
  - CDialog class, XRF 271–272
- OnShowWindow member function
  - CWnd class, XRF 783–784
- OnSize member function
  - CMainWindow class
    - creation and sizing message handlers, XUG 225
    - described, XUG 226
  - CWnd class, XRF 784–785
- OnSizeClipboard member function
  - CWnd class, XRF 785
- OnSpoolerStatus member function
  - CWnd class, XRF 786
- OnSysChar member function
  - CWnd class, XRF 786–787
- OnSysColorChange member function
  - CWnd class, XRF 788
- OnSysCommand member function
  - CWnd class, XRF 788–790
- OnSysDeadChar member function
  - CWnd class, XRF 790–791
- OnSysKeyDown member function
  - CWnd class, XRF 791–792
- OnSysKeyUp member function
  - CWnd class, XRF 792–794
- OnTimeChange member function
  - CWnd class, XRF 794
- OnTimer member function
  - CWnd class, XRF 795
- OnUp member function
  - CMainWindow class
    - adding, XUG 230
    - described, XUG 234
- OnVKeyToItem member function
  - CWnd class, XRF 795–796
- OnVScroll member function
  - CMainWindow class
    - adding, XUG 227
    - described, XUG 229

- ON_WM_CHAR macro
  - Foundation classes cookbook, XUG 356
- ON_WM_KEYDOWN macro
  - Foundation classes cookbook, XUG 356
- ON_WM_KEYUP macro
  - Foundation classes cookbook, XUG 356
- ON_WM_NCDESTROY macro
  - Foundation classes cookbook, XUG 323
- ON_WM_PAINT macro
  - Foundation classes cookbook, XUG 343
  - Foundation classes tutorial, XUG 102
  - CWnd class, XRF 796–797
- OnVScrollClipboard member function
  - CWnd class, XRF 797–798
- OnWinIniChange member function
  - CWnd class, XRF 798–799
- /Oo option, CL, ET 531, 536–537; LR 261
- /Oo- option, CL, ET 531, 536–537
- *ooo* (escape sequence), octal notation, LR 18
- /Op option
  - CL, ET 531, 537–538; PT 23–24, 52
- Opcodes
  - frame sorting, CL option, ET 539
- opcodes, p-code, PT 52, 297–299
- Open command
  - Phone Book sample program, XUG 214
- Open command, PWB, ET 72
- Open Custom command, PWB, ET 76
- Open dialog box, standard
  - Foundation classes tutorial, XUG 167
- _open function, LIB 533–536
- Open member function
  - CFile class, XRF 313–314; XUG 277
- open member function
  - filebuf class, XRF 834–835
  - fstream class, XRF 842
  - ifstream class, XRF 849–850
  - input streams
    - iostream classes tutorial, XUG 386
  - ofstream class, XRF 898
    - iostream classes tutorial, XUG 374
- Open Module command, CodeView, ET 358–359
- Open Module dialog box
  - CodeView, ET 359
- Open Project command, PWB, ET 74
- Open Source command, CodeView, ET 358
- Open Source File dialog box
  - CodeView, ET 358
- OpenClipboard member function
  - CWnd class, XRF 799
- Openfile function, PWB, ET 152, 191, 289–290
- OpenIcon member function
  - CWnd class, XRF 800
- Opening
  - child windows
    - QuickWin, PT 151, 157–158
  - clipboard, CWnd::OpenClipboard, XRF 799
  - databases
    - Foundation classes tutorial, XUG 139
  - file streams for QuickWin windows, _fwopen function, LIB 335–337
  - files
    - CFile::Open, XRF 313
    - _dos_open function, LIB 212–213
    - fopen function, LIB 290–292
    - for attachment to stream's filebuf object, fstream::open, XRF 842
    - for attachment to stream's filebuf object, ifstream::open, XRF 849
    - for attachment to stream's filebuf, ofstream::open, XRF 898
    - for connection to filebuf objects, filebuf::open, XRF 834–835
    - for file sharing, _sopen function, LIB 714–716
    - Foundation classes cookbook, XUG 277
    - memory, CMemFile::CMemFile, XRF 412
    - _open function, LIB 533–536
    - PWB, ET 72, 141, 191, 289–290
  - help files
    - Microsoft Advisor, ET 765–766
    - PWB, ET 213–214
    - QuickHelp, ET 769
  - projects
    - automatically, ET 285–286
    - PWB, ET 49–50, 195–196
  - QuickWin windows, _wopen function, LIB 882–884
  - source files
    - CodeView, ET 358
  - source window
    - CodeView, ET 350
  - streams with file sharing, _fsopen function, LIB 323–325
  - windows
    - CodeView, ET 373
    - PWB, ET 201, 220, 245–246
- Operand data types
  - p-code instructions, PT 49
- Operands, C++
  - *See also* Operators, C++
  - compatibility with operators, LR+ 130–131
  - conversions, LR+ 69–71
- Operands defined, LR 105

- Operating system
  - error codes
    - CFileException::CFileException, XRF 324
    - CFileException::m_lOsError, XRF 328
    - CFileException::OsErrorToException, XRF 325
    - CFileException::ThrowOsError, XRF 326
  - handle for open file, CFile::m_hFile, XRF 321
  - specifying in module-definition files, ET 615–616
- Operating system prompt
  - DOS Shell command, ET 359
- Operating system tags
  - TOOLS.INI file
    - PWB, ET 132
- Operating systems
  - case sensitivity, LIB 9
  - file and path names, LIB 8–9
  - general considerations, LIB 13–14
  - specifying versions, LIB 65
  - variable mode, LIB 65
- Operations
  - regular expressions, PWB, ET 94
- Operations line
  - extending, ET 702
- operator data member
  - ios class, XRF 866
- operator delete
  - array deallocation, TUT 66, 96, 99
  - class scope, TUT 103–104, 106
  - overloading, TUT 101–102
  - overview, TUT 65
- operator member function
  - CObArray class, XRF 462
  - CObject class, XRF 475–476
  - CPoint class, XRF 515–516
- operator new
  - array allocation, TUT 66, 95
  - class scope, TUT 103–104, 106
  - overloading, TUT 101–102
  - overview, TUT 64–65
- Operator overloading
  - friend functions, TUT 143–144
  - guidelines, TUT 138–139, 144
  - member functions, TUT 139–142, 145–147
- Operator overloading (*continued*)
  - overview, TUT 135–139
  - restrictions, TUT 137
- Operator+, overloading, TUT 140, 142
- Operator[ ], TUT 144, 146–147
- Operator=
  - copy constructors, TUT 79
  - default behavior for objects, TUT 69
  - overloading
    - overview, TUT 71–73, 76
    - this pointer, TUT 75
- Operators
  - *See also* Symbols
  - additive, LR 128–130
  - arithmetic
    - table, LR 124
  - assignment, LR 138
  - bitwise, LR 134–135
  - bitwise AND (&), LR 134
  - bitwise exclusive OR (^), LR 134
  - bitwise inclusive OR (|), LR 134
  - bitwise shift, LR 130–132; PT 287
  - cast, LR 126
  - complement, LR 124
  - compound assignment, LR 139
  - conditional, LR 136–138
  - equality, LR 132–133
  - inline assembly limitations, PT 113–114, 116
  - multiplicative, LR 126–128
  - postfix, LR 116–121
  - precedence and associativity table, LR 23
  - prefix decrement, LR 122
  - prefix increment, LR 122
  - regular expressions, PWB, ET 91–93, 95
  - relational
    - described, LR 132–133
    - testing relationships, LR 132
  - simple assignment (=), LR 139
  - sizeof, LR 125–126
  - unary
    - defined, LR 111
    - described, LR 122–126
    - table, LR 12458, 88

Operators, C++
  addition operator
    CRect class, XRF 534
    CString class, XRF 594
    CTime class, XRF 615–616
    CTimeSpan class, XRF 625
  assignment of addition operator
    CRect class, XRF 532
    CString class, XRF 595
    CTime class, XRF 616
    CTimeSpan class, XRF 626
  assignment of intersection operator
    CRect class, XRF 533
  assignment of subtraction operator
    CRect class, XRF 533
  assignment of union operator
    CRect class, XRF 533–534
  assignment operator
    CRect class, XRF 531
    CSize class, XRF 560–561
    CString class, XRF 592
    CTime class, XRF 615
    CTimeSpan class, XRF 625
    istream class, XRF 889
    ostream class, XRF 910
  assignment, overloaded, XUG 30, 221
  associativity, LR+ 11–14
  base, LR+ 406
  binary
    additive, LR+ 104–106
    assignment, LR+ 112–116
    bitwise AND, LR+ 110–111
    bitwise exclusive OR, LR+ 110–111
    bitwise inclusive OR, LR+ 110–111
    bitwise shift, LR+ 106–107
    comma, LR+ 116–117
    equality, LR+ 107–110
    (list), LR+ 102–103
    logical AND, LR+ 111–112
    logical OR, LR+ 112
    multiplicative, LR+ 103–104
    overloading, LR+ 358–363
    relational, LR+ 107–110
  cast. *See* Conversion functions
  conditional, LR+ 117–118
  const char* () operator
    CString class, XRF 592–593
  conversion, TUT 150–151
  delete, LR+ 323–325; XUG 31, 41, 46–47
  described, LR+ 10

Operators, C++ (*continued*)
  equality
    CRect class, XRF 531
  evaluation order, LR+ 11–14
  explicit type conversion, LR+ 119–121
  extraction
    CArchive class, XRF 102
    CString class, XRF 593
    CTime class, XRF 617
    CTimeSpan class, XRF 627
    istream class, XRF 885
    overloading, XUG 391
  flow control, PWB, ET 112–114
  function-call, overloading, LR+ 361
  inequality operator
    CRect class, XRF 532
  insertion
    CArchive class, XRF 103
    CDumpContext class, XRF 278–279
    ostream class, XRF 906
    overloading, XUG 380–381
  intersection operator
    CRect class, XRF 535
  lookup operator, CMapStringtoOb class, XRF 386
  LPRECT operator
    CRect class, XRF 531
  member-selection, overloading, LR+ 363
  new
    CObject class, XRF 475
    dynamic allocation, LR+ 318–320
    Foundation classes cookbook, XUG 296
    Foundation classes tutorial, XUG 31, 41, 58, 88
  operand compatibility, LR+ 130–131
  overloading
    assignment, LR+ 360
    binary, LR+ 358–360
    described, LR+ 351–353
    function-call, LR+ 361
    member-selection, LR+ 363
    overview, LR+ 77
    rules, LR+ 354–355
    subscript, LR+ 362–363
    unary, LR+ 355–358
  pointer-to-member, LR+ 124–126
  postfix
    decrement, LR+ 90–91
    described, LR+ 81
    function-call, LR+ 83–88
    increment, LR+ 90–91
    member-selection, LR+ 89
    subscript, LR+ 81–83

- Operators, C++ (*continued*)
  - precedence, LR+ 11–14
  - preprocessor
    - charizing, LR+ 372
    - defined, LR+ 365, 381
    - described, LR+ 370
    - stringizing, LR+ 371–372
    - token-pasting, LR+ 373
  - subtraction operator
    - CRect class, XRF 533–534
  - subscript operator, CString class, XRF 597
  - syntax, LR+ 11–14
  - unary
    - address-of, LR+ 92–93
    - associativity, LR+ 91
    - decrement, LR+ 94–95
    - delete, LR+ 101–102
    - increment, LR+ 94–95
    - indirection, LR+ 92
    - (list), LR+ 91
    - logical NOT, LR+ 94
    - new, LR+ 97–101
    - one's complement, LR+ 94
    - overloading, LR+ 355–358
    - sizeof, LR+ 95–96
    - unary negation, LR+ 93–94
    - unary plus, LR+ 93
  - union operator
    - CRect class, XRF 531, 535
  - void* operator
    - ios class, XRF 867
- Operators functions
  - using C++ expressions, ET 412
- opfx member function
  - ostream class, XRF 902
- Optimization
  - aggressive, enabling, PT 20
  - __asm blocks, effect of, PT 124–125
  - controlling
    - from PWB, PT 5–6
    - from the command line, PT 6
    - from the linker, PT 25–28
    - using pragmas, PT 6–7
- Optimization (*continued*)
  - customizing, PT 9
  - defaults, PT 8–9
  - disabling
    - all, PT 9
    - loop, unsafe, PT 20
  - environmental considerations, PT 28
  - floating-point math packages, PT 133
  - LINK options, PT 25–28
  - maximum efficiency, PT 24–25
  - options
    - aggressive, enabling, PT 20
    - aliasing assumptions, PT 13–18
    - calling conventions, PT 29–32
    - code segment packing, PT 26–27
    - common subexpression elimination, PT 23
    - data segment packing, PT 27
    - entry points, removing, PT 51
    - entry tables, specifying, PT 51–52
    - executable file packing, PT 27
    - far call translation, PT 25–26
    - floating-point result handling, PT 23–24
    - frame sorting, PT 52
    - function-level linking, PT 21
    - inlining, PT 13
    - intrinsic function generation, PT 10–12
    - loops, PT 18–19
    - loops, disabling unsafe, PT 20
    - maximizing efficiency, PT 24–25
    - overview, PT 6
    - p-code, PT 44
    - processor selection, PT 24
    - register allocation, PT 21–22
    - size, PT 9–10, 24–25
    - speed, PT 9–10, 24–25
    - stack probe removal, PT 21
    - unreferenced function removal, PT 28
  - pragmas, PT 6–7
  - precautions
    - debuggers, PT 9
    - DOS programs, PT 28
    - math intrinsics, PT 11
    - Windows programs, PT 28

Optimization (*continued*)
  PWB options, PT 5–6
  restoring to former state, PT 9
  types described, PT 5
optimize pragma
  described, PT 7
  disabling, PT 9
  options
    common subexpression elimination, PT 23
    disabling subexpression elimination, PT 23
    floating-point result handling, PT 24
    p-code, PT 50
    register allocation, PT 22
    speed vs. size, PT 10
  p-code optimization, ET 538
  replaces loop_opt pragma, LR 266
  subexpression optimization, ET 533
  unsafe optimizations, CL option, ET 532
Optimizing
  assuming no aliasing, CL option, ET 531–532
  common expressions, CL option, ET 533
  compiler options, PWB, ET 52–56
  debugging considerations, ET 324
  entry codes, CL options, ET 515
  entry/exit code, LR 260
  execution time, CL option, ET 539
  exit codes, CL options, ET 515
  exit sequence, CL options, ET 538
  far calls, LINK, ET 580–581, 584, 587–588
  file size, CL option, ET 538
  frame sorting, CL option, ET 539
  inline expansion control, CL option, ET 532
  intrinsic function generation, CL option, ET 534–535
  load-time relocation table, LINK, ET 580
  loops, CL option, ET 537
  maximum optimization, CL, ET 539
  p-code, ET 533, 538; LR 260–261
  post-code generation, CL option, ET 536–537
  pragmas, LR 212
  register allocation, CL option, ET 533
  space, SBRPACK, ET 739–740
  turning off, CL, ET 533
Option button, PWB, ET 81
Optional items, LR xiii; XUG viii
Options
  BSCMAKE, ET 732, 736–737
  CL, ET 488–557
    debugging considerations, ET 323–324
    interaction between, ET 496, 512
    specifying, ET 557–559

Options (*continued*)
  CodeView
    described, ET 338–343, 445–447
    remote debugging, ET 396
    setting, ET 344
  compiler
    changing in PWB, ET 53–56
    debugging considerations, ET 323–324
  CVPACK, ET 744
  debug
    finding symbols, PWB, ET 101
  EXP, ET 750
  floating-point math packages, PT 131–136, 138
  HELPMAKE
    decoding, ET 713–714
    encoding, ET 712–713
  IMPLIB, ET 747
  LIB, ET 701
  LINK, PT 25–28
    debugging considerations, ET 323–325
    described, ET 575–576
    new features, ET 561–562
  memory models, PT 62–63, 70–77
  NMAKE
    Foundation classes tutorial, XUG 11
    macros, ET 676
    turning on, ET 688–689
  optimization
    aggressive, enabling, PT 20
    aliasing assumptions, PT 13–18
    calling conventions, PT 29–32
    code segment packing, PT 26–27
    common subexpression elimination, PT 23
    data segment packing, PT 27
    entry points, removing, PT 51
    entry tables, specifying, PT 51–52
    executable file packing, PT 27
    far call translation, PT 25–26
    floating-point result handling, PT 23–24
    frame sorting, PT 52
    function-level linking, PT 21
    inlining, PT 13
    intrinsic function generation, PT 10–12
    loops, PT 18–19
    loops, disabling unsafe, PT 20
    maximizing efficiency, PT 24–25
    overview, PT 6
    p-code, PT 44
    processor selection, PT 24
    register allocation, PT 21–22
    size, PT 9–10, 24–25

Options (*continued*)
  optimization (*continued*)
    speed, PT 9–10, 24–25
    stack probe removal, PT 21
    unreferenced function removal, PT 28
  p-code compiling, PT 50–52
  precompiled headers, PT 34–36, 38–40
  PWB, ET 141–142
  RM, ET 748
  SBRPACK, ET 740
  UNDEL, ET 749
Options command, CodeView, ET 422, 445–447
options field
  BSCMAKE, ET 735
  NMAKE, ET 647
  SBRPACK, ET 740
Options menu
  CodeView, ET 368–372
  PWB, ET 75
/Oq option
  CL, ET 531, 538; LR 260; PT 44
OR operators
  bitwise exclusive (^), LR 134
  bitwise inclusive (|), LR 134
OR operators, C++
  bitwise exclusive. *See* Bitwise exclusive OR
operator, C++
  bitwise inclusive. *See* Bitwise inclusive OR
operator, C++
  logical. *See* Logical OR operator, C++
/Or option
  CL, ET 531, 538
Order of construction, LR+ 305
Order of destruction, LR+ 308–310
Order of evaluation
  See also Precedence
  C++
    expressions, LR+ 127–129
    operators, LR+ 11–14
  portability guidelines, PT 289–290
  sequence points, LR 113
Ordering segments
  debugging considerations, ET 322
_ORES256COLOR constant, PT 170
_ORESCOLOR constant, PT 170
Origin, coordinate systems
  defined, PT 180
  location, changing, PT 181
OS command, CodeView, ET 445–447
/Os option
  CL, ET 531, 538; PT 9–10
OS2INIT.CMD
  PWB configuration, ET 137
OS2libs switch, PWB, ET 310–311
OsErrorToException data member
  CFileException class, XRF 325
osfx member function
  ostream class, XRF 902
_osmajor variable, LIB 65
_osminor variable, LIB 65
_osmode variable, LIB 65
ostream, tying stream to, ios::tie, XRF 864
ostream class
  described, XRF 900–901
  iostream classes tutorial, XUG 365
  manipulators
    &endl, XRF 907
    &ends, XRF 907
    &flush, XRF 907
  member functions
    flush, XRF 902
    opfx, XRF 902
    osfx, XRF 902
    ostream, XRF 903
    ~ostream, XRF 903
    put, XRF 903
    seekp, XRF 904
    tellp, XRF 904–905
    write, XRF 905
  operators, XRF 906, 910
ostream constructor, XRF 903
ostream destructor, XRF 903, 909
ostream objects
  assigning to ostream_withassign object,
      ostream_withassign::operator=, XRF 910

- ostream objects (*continued*)
  - creating
    - iostream::iostream, XRF 873
    - ostream::ostream, XRF 903
  - destroying, ostream::~ostream, XRF 903
- ostream_withassign class
  - described, XRF 908
  - member functions
    - ostream_withassign, XRF 909
    - ~ostream_withassign, XRF 909
  - operators, XRF 910
- ostream_withassign constructor, XRF 909
- ostream_withassign destructor, XRF 909
- ostream_withassign objects
  - assigning specified ostream object to, ostream_withassign::operator=, XRF 910
  - creating, ostream_withassign::ostream_withassign, XRF 909
  - destroying, ostream_withassign::~ostream_withassign, XRF 909
- ostrstream class
  - described, XRF 911
  - iostream classes tutorial, XUG 366
  - member functions
    - ostrstream, XRF 912
    - ~ostrstream, XRF 913
    - pcount, XRF 913
    - rdbuf, XRF 913
    - str, XRF 914
  - returning pointer to internal character array, ostrstream::str, XRF 914
- ostrstream constructor, XRF 912
- ostrstream destructor, XRF 913
- ostrstream objects
  - creating, ostrstream::ostrstream, XRF 912
  - destroying, ostrstream::~ostrstream, XRF 913
- _osversion variable, LIB 65
- /Ot option
  - CL, ET 531, 539; PT 9–10
- out_waiting member function
  - streambuf class, XRF 926
- _outgtext function, LIB 537–539
- _outmem function, LIB 540–541
- _outp function, LIB 542–544
- Output
  - diagnostic, XRF 29
  - redirecting, CodeView, ET 476–477
  - sequential and random-access, ostream class described, XRF 900
  - viewing, CodeView, ET 374
- Output files
  - alternate, setting, ET 495
  - LINK, ET 563–564
    - name, ET 566
  - preprocessing, creating, ET 540
- Output libraries, LIB, ET 706–707
- Output screen defined, ET 813
- Output stream, C++, TUT 5–6
- Output streams
  - binary output files
    - iostream classes tutorial, XUG 378–379
  - buffering, effect
    - iostream classes tutorial, XUG 377–378
  - deriving
    - streambuf class, XUG 399–405
  - format control
    - Foundation classes tutorial, XUG 368–373
  - insertion operators
    - iostream classes tutorial, XUG 367, 380–381
  - manipulators
    - iostream classes tutorial, XUG 381–382
    - parameters, more than one, XUG 397
    - with one parameter, XUG 395–397
  - objects, constructing
    - output file stream constructors, XUG 366
    - output string stream constructors, XUG 367
  - ofstream class
    - flags, XUG 374–375
    - iostream classes tutorial, XUG 366
  - ofstream member functions
    - bad, XUG 377
    - clear, XUG 377
    - close, XUG 376–377
    - described, XUG 373–374
    - eof, XUG 377
    - fail, XUG 377
    - open, XUG 374
    - put, XUG 375
    - rdstate, XUG 377
    - seekp, XUG 376
    - write, XUG 375–376
  - ostream class
    - iostream classes tutorial, XUG 365
  - ostrstream class
    - iostream classes tutorial, XUG 366
- OutputDebugString function
  - Windows programs
    - Foundation classes tutorial, XUG 36
- Outputting bytes at port, _outp and _outpw functions, LIB 542–544
- _outpw function, LIB 542–544

_outtext function, LIB 545–546
OV command, CodeView, ET 445–447
/OV option
  CL, debugging p-code, ET 390
  LINK, ET 562, 587
/Ov option, CL, ET 531, 539; LR 261
/Ov- option, CL, ET 531, 539
Overflow conditions
  bitwise shift operators may cause, LR 131
  warnings, LR 266
overflow member function
  streambuf class, XRF 926–927
Overlaid programs
  compatibility, MOVE, ET 603
  creating
    LINK, ET 598–601
    module-definition files, ET 619–620
    MOVE, ET 598–601
  interoverlay calls, limiting with LINK, ET 579
  linking, LINK, ET 601–602
  memory allocation
    MOVE, ET 602–603
  module-definition files, ET 607–608
  overlays, ET 604–605
  overview
    LINK, ET 597
    MOVE, ET 597
  segments, ET 603–604
  space restrictions, ET 603
  specifying, LINK, ET 570
Overlaid Virtual Environment. *See* MOVE
Overlay caches
  overlaid DOS programs, ET 598, 602–603
Overlay heaps
  overlaid DOS programs, ET 598, 602
OVERLAY keyword
  module-definition files, ET 619–620
Overlay manager
  overlaid DOS programs, ET 603
Overlay number
  LINK, ET 564
/OVERLAYINTERRUPT option
  LINK, ET 562, 587
Overlays
  compiling, ET 599–600
  defined, ET 813
  DOS program, linking, ET 566
  module-definition files, ET 607–608
  reducing swapping, LR 173
  specifying, MOVE, ET 604
Overloaded assignment operator
  CPerson class
    Foundation classes tutorial, XUG 29–30
  defined, Foundation classes tutorial, XUG 30
  Foundation classes tutorial, XUG 221
  Phone Book sample program, XUG 129
Overloaded functions
  using C++ expressions, ET 411
Overloading
  addition operator, TUT 140, 142
  assignment operator, TUT 71–73, 75–76
  constructors, TUT 48, 53
  delete operator, PT 105
  described, LR+ 339
  extraction operators
    iostream classes tutorial, XUG 391
  functions, TUT 20–23
    address return, LR+ 351
    argument matching, LR+ 344–350
    argument type differentiation, LR+ 340
    declaration matching, LR+ 342–343
    memory-model specifiers, LR+ 401
    restrictions, LR+ 341
  insertion operators
    iostream classes tutorial, XUG 380–381
  new operator, PT 103
  operators
    binary, LR+ 358–360
    described, LR+ 351–353
    friend functions, TUT 143–144
    function-call, LR+ 361
    guidelines, TUT 138–139, 144
    member functions, TUT 139–142, 145–147
    member-selection, LR+ 363
    overview, LR+ 77; TUT 135–139

Overloading (*continued*)
  operators (*continued*)
    restrictions, TUT 137
    rules, LR+ 354–355
    subscript, LR+ 362–363
    unary, LR+ 355–358
  subscript operator, TUT 144, 146–147
  this pointer, PT 100–101
Overriding
  default addressing, LR 168
  Foundation classes, XUG 307
OVL keyword
  module-definition files, ET 619–620
/Ow option
  CL, ET 530–532; PT 13–18
/Ox option, CL, ET 531, 539; PT 24–25
/Oz option, CL, ET 531, 537–538; PT 20

P

P command, CodeView, ET 422, 449, 452–453
:p command, HELPMAKE, ET 723
\p formatting attribute
  HELPMAKE, ET 719, 721
/P option
  CL, ET 540; LR 196
  CodeView, ET 396
  CVPACK, ET 745
  EXEHDR, ET 630
  LIB, ET 701–702
  NMAKE, ET 649
p option, optimize pragma, PT 24
P register
  CodeView syntax, ET 419, 450
pack pragma
  data structure, ET 554–555
  precompiled header compilation, effect on, PT 41
pack pragma, C++, LR+ 390
Packaged functions
  creating, CL options, ET 524
  defined, ET 813
  ordering, module-definition files, ET 626–627
  overlaid DOS programs, ET 599
Packaged INCLUDE
  ordering
    module-definition files, ET 627
/PACKC option
  LINK, ET 587–588; PT 26–27
/PACKCODE option
  LINK, ET 587–588; PT 26–27
/PACKD option
  LINK, ET 588–589; PT 27
/PACKDATA option
  LINK, ET 588–589; PT 27
/PACKF option
  LINK, ET 562, 589; PT 28
/PACKFUNCTIONS option
  LINK, ET 562, 589; PT 28
Packing
  code segments, PT 26–27
  data, options, LR 70
  data segments, PT 27
  executable files, PT 27
  files
    CVPACK, ET 743–744
    SBRPACK, ET 739–740
  preventing
    BSCMAKE, ET 732
  size, warning, LR 265
  structure members, CL options, ET 554–555
  structures, LR 212
  unreferenced functions, PT 28
Padding
  effect of sizeof operator, LR 125
  structure members, LR 243
Page numbers
  active, setting _setactivepage function, LIB 650–651
  current active, getting _getactivepage function, LIB 342–343
  current visual, getting, _getvisualpage function, LIB 388
page pragma
  precompiled header compilation, effect on, PT 41
page pragma, C++, LR+ 391
Page size
  specifying, with LIB, ET 701–702
Pages, visual, setting, _setvisualpage function, LIB 701
/PAGESIZE option
  LIB, ET 701–702
pagesize pragma
  precompiled header compilation, effect on, PT 41
pagesize pragma directive, C++, LR+ 391
Paint message handlers
  Phone Book sample program, XUG 199
Painting
  called to prepare invalidated region, CWnd::OnEraseBkgnd, XRF 737–738
  called when repainting CWnd, CWnd::OnPaint, XRF 772

- Painting (*continued*)
  - client area associated with CPaintDC object, CPaintDC::m_ps, XRF 500
  - client area of window, PAINTSTRUCT structure, XRF 84–85
  - CPaintDC class described, XRF 498
  - CWnd, called when client area needs repainting, CWnd::OnPaintClipboard, XRF 772–773
  - Foundation classes, using, XUG 343–344
  - icon background, CWnd::OnIconEraseBkgnd, XRF 743
  - icons, called when painting, CWnd::OnPaintIcon, XRF 773
  - nonclient areas, called when needing, CWnd::OnNcPaint, XRF 769
  - preparing CWnd for, CWnd::BeginPaint, XRF 660
  - text, HELLO sample program, XUG 101–102
  - windows, marking end, CWnd::EndPaint, XRF 682
- PaintRgn member function
  - CDC class, XRF 218
- PAINTSTRUCT structure, XRF 84–85
  - CPaintDC::CPaintDC, XRF 499
  - CPaintDC::m_ps, XRF 500
  - CWnd::BeginPaint, XRF 660
  - CWnd::EndPaint, XRF 682
  - CWnd::OnPaintClipboard, XRF 772–773
- Palettes
  - CPalette class described, XRF 501
  - creating CPalette objects, CPalette::CreatePalette, XRF 504
  - CWnd, called when receiving input focus, CWnd::OnQueryNewPalette, XRF 777
  - getting colors, lines, styles, patterns, _pg_getpalette function, LIB 565–567
  - graphics
    - CGA, PT 176–177
    - changing, PT 175–176
    - controlling functions (list), PT 188–189
    - EGA, PT 179
    - MCGA, PT 179
    - Olivetti, PT 177
    - VGA, PT 177–179
- Palettes (*continued*)
  - logical
    - mapping entries to system palette, CDC::RealizePalette, XRF 224
    - replacing entries, CPalette::AnimatePalette, XRF 503
    - retrieving closest matching entry, CPalette::GetPaletteEntries, XRF 505
    - retrieving range of entries, CPalette::GetPaletteEntries, XRF 505
    - setting RGB color values and flags, CPalette::SetPaletteEntries, XRF 506
  - presentation graphics
    - character pool, PT 219
    - color pool, PT 215–216
    - described, PT 214–215
    - fill pattern pool, PT 217–218
    - style pool, PT 216
  - remapping colors, _remapallpalette and _remappalette functions, LIB 619–623
  - resetting, CGdiObject::UnrealizeObject, XRF 350
  - resetting to default, _pg_resetpalette function, LIB 571
  - resizing, CPalette::ResizePalette, XRF 506
  - returning pointer to CPalette object, XRF 504
  - selecting graphics, _selectpalette function, LIB 647–649
  - selecting logical, CDC::SelectPalette, XRF 233
  - setting values, _pg_setpalette function, LIB 574
  - system, called after change, CWnd::OnPaletteChanged, XRF 773–774
- \par formatting code
  - HELPMAKE, ET 727
- Paragraphs
  - setting number, LINK, ET 577–578
- Parameter declarations
  - with abstract declarators, LR 180
- Parameters
  - *See also* Arguments; Arguments, C++
  - CString
    - specifying, ET 577; XUG 259
  - defined, LR 30, 179; ET 813
  - described, LR 85, 180–181
  - ellipsis notation, LR 180

Parameters (*continued*)
  macros, LR 193
  mixed-language programming, PT 269
  names in replacement-list, LR 193
  order, LR 180
  passing, mixed-language programming, PT 235–236
  references, TUT 29–31
  token-pasting operator (##), LR 197
  types, LR 180
Parent process defined, ET 813; LIB 51
Parent windows
  called when child window created or destroyed, CWnd::OnParentNotify, XRF 774–775
  changing parent of child, CWnd::SetParent, XRF 810
  retrieving, CWnd::GetParent, XRF 696
Parentheses ( )
  around identifier names, LR 89
  balancing, in PWB, ET 192–193
  enclosing expression arguments, LR 160–161
  enclosing operands, LR 107
  ensure precedence of complicated arguments, LR 193
  in complex abstract declarators, LR 88
  in identifiers, LR 55
  modifying functions, LR 88
  modifying identifiers, LR 42
  overriding defaults of declarations, LR 60
  preserving precedence in expressions, LR 192
  searching, PWB, ET 91, 93
Parsing
  command-line arguments, LR 32–33
Parsing, C++
  command-line arguments
    startup code, LR+ 40–42
  tokens
    described, LR+ 2–3
.PAS files defined, ET 813
Pascal
  calling convention, PT 29
  mixed-language programming, PT 238–240, 246–248
PASCAL command macro, NMAKE, ET 676
Pascal Compiler
  NMAKE command macro, ET 676
  NMAKE options macro, ET 676
__pascal keyword, PT 29, 239–240, 247–248
  calling conventions, LR 55
  CL
    calling conventions, ET 516–518
    enabling, ET 550
__pascal keyword (*continued*)
  modifying function names, LR 57
  restrictions, LR 57, 171, 266
  specifying, LR 170
__pascal keyword, C++
  calling convention, LR+ 418
  described, LR+ 7
Passing
  arguments to functions, LR 119, 185–187
  arrays, mixed-language programming, PT 263
  control to functions, LR 183–185
  execution control, LR 166
  objects by reference to functions, return types, LR+ 53–54
  pointers
    to arrays, LR 185
    to functions, LR 185
    to structures, LR 178
  values in different translation units, LR 190
Password character, setting or removing in edit control, CEdit::SetPasswordChar, XRF 299
Paste buffer, QuickWin, PT 149–150
Paste command
  CodeView, ET 360
  PWB
    described, ET 73
    predefined macros, ET 143
.paste command
  QuickWin, PT 149
Paste function, PWB, ET 152, 191–192
  executing, ET 108
  replacing text, ET 94
  HELPMAKE, ET 723
Paste member function
  CComboBox class, XRF 152
  CEdit class, XRF 297
Pasting
  clipboard data into edit control, CEdit::Paste, XRF 297
  text
    Microsoft Advisor, ET 761
    QuickHelp, ET 771
Pasting text, QuickWin, PT 149
PatBlt member function
  CDC class, XRF 218–219
PATH environment variable
  CodeView, installing, ET 327
  starting PWB, ET 67

Path names
  breaking into components, _splitpath function, LIB 723–724
  creating, _makepath function, LIB 476–478
  delimiters, LIB 9
  getting current directory, _getdcwd function, LIB 356–358
  making absolute from relative names, _fullpath function, LIB 333–334
  operating system conventions, LIB 8–9
Path specifications
  fully qualified, LR 201
Paths
  Curfile predefined macro, PWB, ET 224
  defined, ET 813
  predefined expression syntax, ET 778, 780, 786
  search, NMAKE, ET 660, 682
  specifying, ET 88, 496
Patterns
  See also Regular expressions
  fill patterns, presentation graphics, PT 217–218
  graphics, functions (list), PT 191–192
/PAU option
  LINK, ET 589–590
Pause command
  QuickWin, PT 148–149
Pause command, CodeView, ET 423, 470
/PAUSE option
  LINK, ET 589–590
Pausing
  Trace Speed command, CodeView, ET 369
pbackfail member function
  streambuf class, XRF 927
Pbal function, PWB, ET 152, 192–193
pbase member function
  streambuf class, XRF 928
pbump member function
  streambuf class, XRF 928
.PCH files
  See also Precompiled headers
  defined, ET 813
P-code
  build process, PT 53

P-code (*continued*)
  compiling
    from PWB, PT 43–44
    from the command line, PT 44
    options, PT 50–52
  debugging, ET 372, 389–393; PT 45
  described, PT 43
  ensures small code size, LR 258
  entry points
    described, PT 47
    removing, PT 51
  entry tables, specifying, ET 521
  entry tables, specifying maximum, PT 51–52
  error messages. *See* “Error Messages,” Part 2 of this book
  fine-tuning, PT 50
  frame sorting, PT 52
  function calls, PT 47
  functions, native entry points
    described, PT 47
    removing, PT 51
  instructions
    data types, PT 48–49
    modes, PT 48
    naming conventions, PT 47–49
    qualifiers, PT 48
    (table), PT 297–299
  linking, PT 53
  mixing with machine code, PT 50
  modifying before compiling, PT 50
  naming conventions, instructions, PT 47–49
  native entry points, LR 212, 260
    described, PT 47
    removing, PT 51
  native entry points, removing, ET 520–521
  opcodes, PT 52, 297–299
  optimization, LR 260–261
  optimizing, CL option, ET 538
  quoting, LR 261
    controlling, PT 52
    described, PT 46
    disabling, PT 46

P-code (*continued*)
  quoting, CL option, ET 533
  registers, displaying, ET 355
  segment, naming, LR 260
  sorting local variables by frequency of use, LR 261
  stack machine, PT 45–46
  stacks, local variable allocation order, PT 52
  transition from machine code, PT 47
__PCODE macro, C++, LR+ 375
/PCODE option
  LINK, PT 53
pcount member function
  ostrstream class, XRF 913
  strstream class, XRF 940
peek member function
  istream class, XRF 882
Pens
  available in device context, enumerating, CDC::EnumObjects, XRF 182–184
  CPen class described, XRF 508
  creating
    constructor, CPen::CPen, XRF 509
    initializing CPen::CreatePen, XRF 510
    initializing with specified structure, CPen::CreatePenIndirect, XRF 510
  handles, CPen::FromHandle, XRF 511
  predefined
    retrieving handle to, CGdiObject::CreateStockObject, XRF 345–346
    selecting, CMetaFileDC::SelectStockObject, XRF 442
  setting drawing mode, CDC::SetROP2, XRF 243–244
Percent sign (%)
  file specifier, NMAKE, ET 653
  Filename-Parts Syntax, PWB, ET 265–266
Period (.)
  Current Location command, CodeView, ET 423, 471
  dot directives, NMAKE, ET 687
  inference rules, NMAKE, ET 681
  line number specifier, CodeView, ET 365
  LINK syntax, ET 565
  match character
    regular expression syntax, ET 779
  wildcard character
    regular expression syntax, ET 778, 781
perror function, LIB 547–548
/PF option
  PWB, ET 141
PFLAGS options macro
  NMAKE, ET 676
_pg_analyzechart function, LIB 549–551; PT 201–202
_pg_analyzechartms function, LIB 549–551; PT 201–202
_pg_analyzepie function, LIB 552; PT 201–202
_pg_analyzescatter function, LIB 553–554; PT 201–202
_pg_analyzescatterms function, LIB 553–554; PT 201–202
_pg_chart function, LIB 555–557; PT 201–202
PGCHART.LIB, PT 201–202
_pg_chartms function, LIB 555–557; PT 201–202
_pg_chartpie function, LIB 558–559; PT 201–202
_pg_chartscatter function, LIB 560–561; PT 201–202
_pg_chartscatterms function, LIB 560–561; PT 201–202
_pg_defaultchart function, LIB 562–563; PT 201–202
_pg_getchardef function, LIB 564; PT 201–202
_pg_getpalette function, LIB 565–567; PT 201–202
_pg_getstyleset function, LIB 568; PT 201–202
_pg_hlabelchart function, LIB 569; PT 201–202
_pg_initchart function, LIB 570; PT 201–202
_pgmptr variable, LIB 67
_pg_resetpalette function, LIB 571; PT 201–202
_pg_resetstyleset function, LIB 572; PT 201–202
_pg_setchardef function, LIB 573; PT 201–202
_pg_setpalette function, LIB 574; PT 201–202
_pg_setstyleset function, LIB 575; PT 201–202
_pg_vlabelchart function, LIB 576; PT 201–202
Phases of translation, LR 190–191
Phases of translation, C++, LR+ 395–396
PHBOOK sample program
  CMainWindow class
    Foundation classes tutorial, XUG 121
  compiling, XUG 243
  data interface, simplifying, XUG 122–134
  database
    ASSERT macro, XUG 127, 138
    ASSERT_VALID macro, XUG 126, 138
    AssertValid member function, XUG 133, 138
    CDataBase class, XUG 123, 134–140
    CDataBase constructor, XUG 135
    CMenu class, XUG 119
    CObList class, XUG 127
    code listings, XUG 119, 140–149
    CPersonList object, XUG 134
    editing, XUG 118
    encapsulation, XUG 120, 134, 140

PHBOOK sample program (*continued*)
  database (*continued*)
    Foundation classes tutorial, XUG 117
    member functions, XUG 135
    overview, XUG 118
    serialization, XUG 131–133
  dialog boxes
    adding, XUG 153–161
    code listings, XUG 151
    COMMDLG.H file, XUG 153, 167
    described, XUG 151, 162
    editing tools, XUG 156
    extern "C" directive, XUG 153, 167
    HELLO, using as a template, XUG 152–153
  file handling member functions, XUG 136
  menu commands, XUG 135
  message handlers
    ASSERT macro, XUG 203
    assertions, XUG 203
    CMainWindow class, XUG 199–202
    code listings, XUG 243
    constructors, adding, XUG 202
    creation and sizing, XUG 199, 224–226
    described, XUG 197
    functions, XUG 204
    keyboard and mouse, XUG 199, 230–235
    menu commands, adding to, XUG 199, 205–223
    message maps, adding, XUG 199–202
    naming conventions, XUG 204–205
    notification messages, XUG 205
    ON_COMMAND macro, XUG 204
    painting, XUG 199
    planning for, XUG 197–199
    scrolling, XUG 199
    scrolling member functions, adding, XUG 227–229
    supporting files, preparing, XUG 242
    utility member functions, adding, XUG 238–241
    VERIFY macro, XUG 203
    WM_COMMAND message, XUG 204
    WM_PAINT, adding, XUG 235–237, 241
  message maps
    Foundation classes tutorial, XUG 204
  overview, XUG 6

PHBOOK sample program (*continued*)
  writing
    Foundation classes tutorial, XUG 120, 122
Phone Book sample program. *See* PHBOOK sample program
Phone support. *See* the Product Assistance Request Form in LIB
Physical coordinates
  described, PT 180–182
  using, functions listed, PT 181
Physical segments
  defined, ET 813
PID defined, ET 813
Pie charts
  analyzing data series for, _pg_analyzepie function, LIB 552
  described, PT 203
  displaying, _pg_chartpie function, LIB 558–559
  sample program, PT 206–208
  styles, PT 204–205
_pie function, LIB 577–579; PT 190
Pie member function
  CDC class, XRF 219–220
PIE.C sample presentation graphics program, PT 206–208
Pies, determining viewpoint coordinate endpoints, _getarcinfo function, LIB 344
Pie-shaped wedges, creating, CDC::Pie, XRF 219–220
_pie_w function, LIB 577–579
_pie_wxy function, LIB 577–579; PT 190
PIF files
  association with Makefiles, ET 67
Pixel cursors, coordinate systems, PT 186
Pixels
  converting coordinates, LIB 23
  getting values, _getpixel functions, LIB 374–375
  retrieving RGB color values, CDC::GetPixel, XRF 202
  setting at specified point, CDC::SetPixel, XRF 242
  setting to current color, _setpixel functions, LIB 676–677

- /PL option
  - PWB, ET 141
- \plain formatting code
  - HELPMAKE, ET 727
- Platforms
  - optimization precautions, PT 28
- Playback macro, PWB, ET 224
- PlayMetaFile member function
  - CDC class, XRF 221
    - CMetaFileDC::Close, XRF 440
- Plines function, PWB, ET 152, 193
- Plus operator (+). *See* Addition operator
- Plus operator, unary, C++
  - unary-operator expressions, LR+ 93
- Plus sign (+)
  - Add command, LIB, ET 703–704
  - concatenating help files, ET 772
  - LINK syntax, ET 565, 567
  - options, NMAKE, ET 688–689
  - searching, PWB, ET 92
- /PM option
  - LINK, ET 562, 590
    - setting application type, EXEHDR, ET 630
- /PMTYPE option
  - EXEHDR, ET 630
  - LINK, ET 562, 590
- /PN option
  - PWB, ET 141
- Point of declaration, C++, LR+ 29–30
- POINT structure, XRF 85
  - CDC::Arc, XRF 166
  - CDC::Chord, XRF 171
  - CDC::DPtoLP, XRF 175
  - CDC::DrawIcon, XRF 176
  - CDC::GetPixel, XRF 202
  - CDC::LineTo, XRF 214
  - CDC::LPtoDP, XRF 215
  - CDC::MoveTo, XRF 215
  - CDC::Pie, XRF 220
  - CDC::Polygon, XRF 221
  - CDC::PolyPolygon, XRF 222
  - CDC::PtVisible, XRF 223
  - CDC::RoundRect, XRF 227
  - CDC::SetBrushOrg, XRF 239
  - CDC::SetPixel, XRF 242
  - CDC::SetViewportOrg, XRF 251
  - CDC::SetWindowOrg, XRF 253
  - CDialog::IsDialogMessage, XRF 269
  - CPen::CreatePenIndirect, XRF 511
  - CPoint::CPoint, XRF 513
  - CPoint::Offset, XRF 514
- POINT structure (*continued*)
  - CPoint::operator, XRF 515–516
  - CRect::BottomRight, XRF 523
  - CRect::OffsetRect, XRF 528
  - CRect::operator+, XRF 534
  - CRect::operator+=, XRF 532
  - CRect::operator–, XRF 534
  - CRect::operator– =, XRF 533
  - CRect::PtInRect, XRF 528
  - CRgn::CreatePolygonRgn, XRF 542
  - CRgn::CreatePolyPolygonRgn, XRF 543
  - CRgn::OffsetRgn, XRF 547
  - CRgn::PtInRegion, XRF 548
  - CSize::CSize, XRF 559
  - CWnd::ClientToScreen, XRF 663
  - CWnd::OnGetMinMaxInfo, XRF 740
  - CWnd::ScreenToClient, XRF 802
  - CWnd::WindowFromPoint, XRF 821
- Pointer arithmetic
  - huge memory model, PT 60–61
  - mixed memory model, PT 64
  - speed, PT 57–58
- Pointer conversions, C++
  - from arrays, LR+ 73–74
  - integral constant expressions to null pointer, LR+ 75
  - keyword-modified pointers, LR+ 74
  - pointers of type void to other types, LR+ 71
  - pointers to base classes to pointers to derived classes, LR+ 76
  - pointers to classes to pointers to base classes, LR+ 72–73
  - pointers to functions to type void, LR+ 71
  - pointers to objects to type void, LR+ 71
  - zero values to null pointer, LR+ 71, 75
- Pointers
  - address storage, PT 55
  - argv parameter, LR 31
  - arithmetic, LR 128–130, 242
  - arrays, LR 185; XRF 458
  - based
    - *See also* Based pointers
    - described, PT 58, 78–79
    - fixed base, PT 79–80
    - __self keyword, PT 85
    - variable base, PT 80–84
    - __void keyword, PT 86
  - based, C++, LR+ 406–411
  - based on constant, LR 80
  - based on pointers, LR 82
  - based on segment variable, LR 81
  - based on self, LR 83–84

Pointers (*continued*)
- based on void, LR 82
- casting, LR 241
- CFile object
  - getting for archive, XRF 97
- checking, LR 210
  - CL option, ET 556–557
- CMDIChildWnd to parent CMDIFrameWnd, CMDIChildWnd::m_pMDIFrameWnd, XRF 400
- CObject
  - lists, CObList class described, XRF 477
  - maps to CString objects, XRF 377
- codes
  - memory model, CL options, ET 489–490
  - sizes, PT 56–57, 64, 66–68
- comparisons, LR 132–133
- const keyword, effect, LR+ 188–190
- converting, LR 86, 146–147, 251
- converting global memory handles, ET 463–464
- converting local memory handles, ET 466–467
- CWnd
  - object when given handle to window, CWnd::FromHandle, XRF 684
  - retrieving to active, CWnd::GetActiveWindow, XRF 685
- data, sizes, PT 56–58, 64–66
- declarations, LR 76–79
- declarators, LR+ 188–190
- defined, ET 813; LR 54
- envp parameter, LR 31
- expanding and contracting
  - CodeView, ET 367–368, 478–479
- far pointers, PT 57
- far, setting offsets and segments, _FP_OFF and _FP_SEG functions, LIB 293–294
- file. *See* File pointers
- functions, LR 185
- get
  - advancing past spaces and tabs, istream::eatwhite, XRF 877
  - changing for stream, istream::seekg, XRF 883
  - getting value, istream::tellg, XRF 884
  - incrementing, streambuf::gbump, XRF 925

Pointers (*continued*)
- huge pointers, PT 57–58
- mixed-language programming, PT 266–267
- mixed memory models, problems caused by, PT 64–65
- near pointers, PT 56–57
- null, LR 31, 133, 147, 149
  - memory models, using with, PT 61–62
  - portability guidelines, PT 282
- portability guidelines, PT 281–283
- put, incrementing, streambuf::pbump, XRF 928
- repositioning external file pointer, streambuf::pbackfail, XRF 927
- returning, TUT 82
- returning, display context for client area, CWnd::GetDC, XRF 688
- size
  - code pointers, custom memory model, PT 70–71
  - converting, PT 68–69
  - data pointers, custom memory model, PT 71
  - defaults, PT 58
  - (table), PT 283
- smart, defined, LR+ 363
- this pointer, LR+ 244–246
  - argument matching, overloaded functions, LR+ 345–346
  - overloading, PT 100–101
- to bit fields, LR 69
- to const objects
  - initializing, LR+ 218
- to functions, C++
  - types, LR+ 52
- to identifiers, LR 86
- to interrupt handler, LR 175
- to members, C++
  - declarators, LR+ 196–198
  - types defined, LR+ 55
- to unspecified type, LR 51
- to void, LR 77, 146–148
- types, conversions, LR 146–147
- values
  - accessing, LR 123
  - converting to integral, LR 146

Pointers (*continued*)
  values (*continued*)
    printing, LR 251
    reading, LR 251
    with indirection operators, LR 122
  void
    16-bit words keyed by, XRF 375
    CPtrArray class described, XRF 517
    CPtrList class described, XRF 519
    keyed by void pointers, XRF 373
    maps keyed by 16-bit words, XRF 393
    maps keyed by CString objects, XRF 387
  volatile keyword, effect, LR+ 188–190
  v-table, described, PT 102–103
Pointer-to-member operators, C++
  expressions with, described, LR+ 124–126
Points
  adding separate values to x and y members,
      CPoint::Offset, XRF 513
  checking
    equality between two, CPoint::operator= =,
      XRF 515
    if within region, CRgn::PtlnRegion, XRF 548
    inequality between two, CPoint::operator!=,
      XRF 515
  converting, logical to device, CDC::LPtoDP,
      XRF 215
  CPoint class described, XRF 512
  defining x- and y- coordinates of, XRF 85
  determining if within rectangles, CRect::PtlnRect,
      XRF 528
  drawing functions (list), PT 189–191
  identifying window containing given,
      CWnd::WindowFromPoint, XRF 821
  mapping coordinates from device to logical system,
      CDC::DPtoLP, XRF 175
  offsetting by a size, CPoint::operator+=, +,
      XRF 515–516
  offsetting negatively by a size, CPoint::operator-,
      XRF 516
  rectangles
    referencing bottom right, CRect::BottomRight,
      XRF 523
    referencing top left, CRect::TopLeft, XRF 529
  specified, determining which child window
      contains, CWnd::ChildWindowFromPoint,
      XRF 662–663
  subtracting a size, CPoint::operator-=, XRF 516
_polygon functions, LIB 580–582
Polygon member function
  CDC class, XRF 221
Polygons
  creating multiple filled, CDC::PolyPolygon,
      XRF 222
  drawing, CDC::Polygon, XRF 221
  filling mode, retrieving current,
      CDC::GetPolyFillMode, XRF 203
  regions
    creating, CRgn::CreatePolygonRgn, XRF 542
    creating series of, CRgn::CreatePolyPolygonRgn,
      XRF 543
  setting, filling mode, CDC::SetPolyFillMode,
      XRF 242
Polygons, drawing, _polygon functions,
    LIB 580–582
_polygon_w function, LIB 580–582
_polygon_wxy function, LIB 580–582
Polyline member function
  CDC class, XRF 222
Polymorphism
  among Foundation classes, XRF 10
  defined, TUT 123
  design issues, TUT 178
  limitations, TUT 195–196
PolyPolygon member function
  CDC class, XRF 222–223
Pools, presentation graphics
  character pool, PT 219
  color pool, PT 215–216
  fill pattern, PT 217–218
  style pool, PT 216
.popup command
  HELPMAKE, ET 723
Pop-up menus
  called when about to become active,
      CWnd::OnInitMenuPopup, XRF 744–745
  defined, ET 813
  determining number of items,
      CMenu::GetMenuItemCount, XRF 424
  displaying floating, with item tracking,
      CMenu::TrackPopupMenu,
        XRF 436–437
  obtaining item identifier, CMenu::GetMenuItemID,
      XRF 425
  retrieving CMenu object, CMenu::GetSubMenu,
      XRF 428
  specifying status of items, CMenu::GetMenuState,
      XRF 425–426
Pop-up windows, determining most recently active,
    CWnd::GetLastActivePopup, XRF 692–693

Port defined, ET 813
Port Input command, CodeView, ET 422, 434–435
Port Output command, CodeView, ET 422, 448
port: option
  CodeView, ET 396
Portability
  ANSI compatibility, LR 233
  disabling /Za command-line option, LR 5
  effect of /Ze command-line option, LR 265
  integers converted to pointer type, LR 147
  keywords, LR 55
  machine-specific features, LR 209
  sizeof operator vs. hard-coded data sizes, LR 99
Portability guidelines
  address space, PT 283–284
  argument lists, variable, PT 289
  bit fields, PT 279–280
  byte order, PT 277–278, 292–293
  case translation, PT 285
  character set, PT 284–285
  compiler assumptions, PT 285
  data files, PT 292
  data types, PT 271–274
  environments, PT 291
  evaluation order, PT 289–290
  function and macro arguments, PT 290–291
  global register allocation, PT 289
  hardware assumptions, PT 271
  I/O, PT 291
  identifiers, PT 288
  memory availability assumptions, PT 283–284
  Microsoft C specific issues, PT 292
  pointers, PT 281–283
  processor arithmetic modes, PT 280–281
  register variables, PT 288–289
  sign extension, PT 285, 287
  storage order and alignment, PT 274–276
  structures
    bit fields, PT 279–280
    order and alignment, PT 275–276
    reading and writing, PT 278–279
  type promotion, PT 285–287
  unions, PT 276–277
Ports, I/O routines, LIB 43–44
Position, getting current and returning as structure, _getcurrentposition functions, LIB 352–353
Position, retrieving current, CDC::GetCurrentPosition, XRF 195
Postfix expressions, C++, LR+ 81
Postfix operators
  CodeView precedence, ET 406
  described, LR 116–121
Postfix operators, C++
  decrement, LR+ 90–91
  function-call, LR+ 83–88
  increment, LR+ 90–91
  member-selection, LR+ 89
  subscript, LR+ 81–83
  (table), LR+ 81
PostMessage member function
  CWnd class, XRF 800–801
Pound sign (#)
  custom builds, ET 59
  HELPMAKE syntax, ET 712–713
  inference rules, NMAKE, ET 681
  makefile comments, NMAKE, ET 654
  Tab Set command, CodeView, ET 423, 470
  TOOLS.INI file syntax, ET 652
  user-defined macros, NMAKE, ET 669
pow function, LIB 583–584; PT 11
Power, regular expression syntax, ET 780, 787
Powers, calculating, pow functions, LIB 583–584
_powl function
  intrinsic form, PT 11
/PP option
  PWB, ET 141
Ppage function, PWB, ET 152, 194
Ppara function, PWB, ET 152, 194
pptr member function
  streambuf class, XRF 928
PQ register
  CodeView, ET 419
  syntax, ET 450
#pragma preprocessor directive, C++, LR+ 385–392

Pragmas
  alloc_text, ET 599, 626; LR 57, 169, 210
  auto_inline, LR 210
  C++, LR+ 386–392
  check_pointer, ET 556–557; LR 210
  check_stack, ET 518–520; LR 210
  comment, LR 210
  consistency rules, precompiled headers, PT 41
  data_seg, LR 211
  described, LR 189–213
  function, LR 211
  hdrstop, LR 211
  inline_depth, LR 211
  inline_recursion, LR 211
  intrinsic, LR 211
  linesize, LR 212
  loop_opt, LR 266
  message, LR 212
  Microsoft C specific, LR 27
  native_caller, ET 520–521; LR 212
  new in version 7.0, LR 258
  obsolete, LR 266
  optimization, PT 6–7
  optimize, LR 212, 266
    subexpression optimization, ET 538
    unsafe optimizations, ET 532
  overview, LR 26
  pack, ET 554–555; LR 71, 212
  page, LR 213
  pagesize, LR 213
  same_seg, LR 213, 266
  skip, LR 213
  subtitle, LR 213
  title, LR 213
  unsupported, LR 213
  warning, LR 213
Precedence
  defined, ET 813
  described, LR 112–114
  inference rules, NMAKE, ET 686–687
  macros, NMAKE, ET 680
  operator overloading, TUT 137
  operators, LR 23, 132
  operators, C++, LR+ 11–14
  prefix increment and decrement operators, LR 122
.PRECIOUS dot directive
  NMAKE, ET 687
precision member function
  ios class, XRF 861
Precompiled headers
  CL options, ET 546–550; LR 261
  consistency rules, PT 39–41
  controlled by #pragma hdrstop, LR 211
  creating, LR 261, 266; PT 34–36
  debugging, LR 261, 266
  debugging information, overriding CodeView, PT 38–39
  described, PT 33
  hdrstop pragma
    placement, PT 37–38
    syntax, PT 36–37
  include path consistency, PT 40
  new feature, LR 257
  options, PT 34–40
  pragma consistency, PT 41
  source file consistency, PT 41
  using, PT 33–36
Predefined collections, using, Foundation classes cookbook, XUG 270
Predefined expressions
  non-UNIX syntax, ET 780
  UNIX syntax, ET 778, 785
Predefined inference rules
  NMAKE, ET 684–685
Predefined macros
  described, LR 198–200; LR+ 7–9
  line control, LR+ 384
  PWB, ET 222–224, 227–263
    key assignments, ET 146–150
    menu command, ET 142, 144–146
  (table), LR+ 374–376
Predefined names, removing, CL option, ET 542–544
Predefined stream objects
  cerr, XRF 900
  cin, XRF 895
  clog, XRF 900
  cout, XRF 900
Prefixes
  context, HELPMAKE, ET 729
  program segments, ET 323
PRELOAD keyword
  module-definition files, ET 621
Preprocessing
  copying output, CL, ET 493–494
  preserving comments, CL, ET 491
  translation phase, LR 191
Preprocessing, C++
  line control, LR+ 384–385
  macros, LR+ 366–368

Preprocessor, C++, LR+ 365
Preprocessor directives
bracketed file names, LR 245
character set, LR 245
define, LR 190–197
described, LR 189–213
example, LR 30
(list), LR 26, 246
NMAKE, ET 688, 690–692
overview, LR 26
quoted file names, LR 246
inline assembly, using in, PT 115–116
Preprocessor directives, C++
conditional compilation control, LR+ 379
#define, LR+ 368–370
defined, LR+ 365
described, LR+ 366
#elif, LR+ 379–383
#else, LR+ 379–383
#endif, LR+ 379–383
#error, LR+ 385
grammar summary, LR+ 434–435
#if, LR+ 379–383
#ifdef, LR+ 383–384
#ifndef, LR+ 383–384
#include, LR+ 376–378
#line, LR+ 384–385
(list), LR+ 366
null, LR+ 384
#pragma, LR+ 386–392
#undef, LR+ 373–374
Preprocessor operators, LR 195
Preprocessor operators, C++
charizing, LR+ 372
defined, LR+ 365, 381
described, LR+ 370
stringizing, LR+ 371–372
token-pasting, LR+ 373
Presentation graphics
See also Charts
analysis functions, PT 202
character pool, PT 219
chart types and features, PT 202–205
Presentation graphics (*continued*)
color pool, PT 215–216
customizing, PT 219–220
defined, PT 201
displaying, LIB 29–30
environment variables
_axistype structures, PT 221–223
_chartenv structures, PT 219–220, 226–227
described, PT 219–220
_legendtype structures, PT 225
_titletype structures, PT 220–221
_windowtype structures, PT 223–224
fill patterns, PT 217–218
functions, LIB xii, 29, 31
graph types and features, PT 202–205
initializing, _pg_initchart function, LIB 570
library, PGCHART.LIB, PT 201–202
palettes
character pool, PT 219
color pool, PT 215–216
described, PT 214–215
fill pattern pool, PT 217–218
style pool, PT 216
pattern pool, PT 217–218
pools
character, PT 219
color, PT 215–216
fill pattern, PT 217–218
style, PT 216
primary functions (list), PT 201–202
programs, writing steps, PT 205–206
sample programs
BAR.C, PT 208–212
PIE.C, PT 206–208
SCATTER.C, PT 212–214
secondary functions (list), PT 201–202
style pool, PT 216
PreTranslateMessage member function
CWinApp class, XRF 638
CWnd class, XRF 801
PrevDlgCtrl member function
CDialog class, XRF 272

.previous command
  HELPMAKE, ET 723
Previous command, PWB
  described, ET 76
  function, ET 145
Previous Error command, PWB
  described, ET 74
  predefined macros, ET 144
Previous match command, PWB
  described, ET 73
  predefined macros, ET 144
  searching, ET 89
Primary expressions, LR 106–107
Primary expressions, C++, LR+ 78–80
Print command
  CodeView, ET 358–359
  Phone Book sample program, XUG 212
  PWB, ET 72
Print dialog box
  CodeView, ET 359
  standard, Foundation classes tutorial, XUG 167
Print function, PWB, ET 152, 194–195
Print Results command, PWB, ET 77
Printcmd switch, PWB, ET 264–266, 291
PrintDlg function
  Foundation classes tutorial, XUG 163
printf function, LIB 585–592
Printfile entry
  TOOLS.INI file
    CodeView, ET 330, 334
Printing
  aborting current job, CDC::AbortDoc, XRF 164
  canceling
    _pwbcancelprint macro, ET 230
  data to stream, fprintf function, LIB 299–300
  ending
    job, CDC::EndDoc, XRF 180
    page, CDC::EndPage, XRF 181
  error information, LIB 63
  error messages
    perror function, LIB 547–548
    strerror and _strerror functions, LIB 755–756
  files
    CodeView, ET 359
    PWB, ET 194–195
  font-based text in graphics mode, _outgtext function, LIB 537–539
  informing device driver of new job, CDC::StartDoc, XRF 254
Printing (*continued*)
  installing abort procedure in job, CDC::SetAbortProc, XRF 235–237
  job, called when adding or deleting from queue, CWnd::OnSpoolerStatus, XRF 786
  memory statistics report, CMemoryState::DumpStatistics, XRF 56
  output to streams, printf function, LIB 585–592
  preparing printer driver to receive data, CDC::StartPage, XRF 254
  specifying program
    PWB, ET 291
  text
    graphics mode, _outtext function, LIB 545–546
    of specified length in graphics mode, _outmem function, LIB 540–541
  to console, _cprintf function, LIB 165–166
private
  base classes, TUT 131
  members, TUT 45, 112
Private assignment operator, CObject::operator =, XRF 475
PRIVATELIB keyword
  LIBRARY statement
    module-definition files, ET 612–613
Privileged mode, ET 813
PRN
  CL options, appending to, ET 497–498
Procedural abstraction, TUT 162–163
Procedural decomposition, TUT 172
Procedure call defined, ET 813
Process control functions, LIB 49–52
Process Descriptor Block
  command field, CodeView, ET 382
Process identification number defined, ET 813
Processes
  child, loading and executing, _exec functions, LIB 246–250
  identification, _getpid function, LIB 373
  terminating calling, exit and _exit functions, LIB 251–252
Processing at exit, atexit and fatexit functions, LIB 96–97
Processor arithmetic modes
  portability guidelines, PT 280–281
Processors
  p-code run-time interpreter, PT 45–46
  selecting, generating instructions, PT 24

Product assistance. *See* the Product Assistance Request Form in LIB
Profiler
  machine code vs. p-code, PT 50
Program Arguments command, PWB, ET 74
Program execution
  outcomes
    Foundation classes cookbook, XUG 297
Program Item
  adding, PWB, ET 66
Program Manager
  Foundation classes tutorial, XUG 9
Program segment prefixes
  debugging considerations, ET 323
Program Step command, CodeView, ET 386, 422, 449, 452
Program step defined, ET 814
Programmer's WorkBench. *See* PWB
Programming, mixed-language. *See* Mixed-language programming
Programs
  aborting, assert function, LIB 92–93
  building, ET 56, 58
  debugging, preparing for, ET 321–325
  efficiency, increasing, PT 24–25
  executing, sending signal to, raise function, LIB 607–608
  execution, LR 30–34
  overlaid
    creating, ET 619–620
    LINK, ET 570, 579, 597–601
    module-definition files, ET 607–608
    MOVE, ET 597–605
  PWB
    building, ET 45–46
    debugging, ET 29
    editing, ET 47–49, 55
    multimodule, ET 41–42
    non-PWB makefiles, ET 61–63
    project dependencies, ET 45, 48
    running, ET 46–47
  saving current state, setjmp function, LIB 666
Programs (*continued*)
  size
    optimizing, PT 9–10
    p-code use, effect on, PT 43
  speed
    optimizing, PT 9–10
    p-code use, effect on, PT 43
  termination, LR 30
Programs, C++
  defined, LR+ 33
  elements (list), LR+ 1
  file translation order, LR+ 1–2
  startup code
    initialization considerations, LR+ 43–44
    main function, LR+ 38–42
  termination
    initialization considerations, LR+ 44–45
    methods, LR+ 42–43
Project dependencies
    PWB, ET 45
Project function, PWB, ET 152, 195–196
Project menu, PWB
  described, ET 74
  predefined macros, ET 144
Project Templates command, PWB, ET 75
Projects
  opening
    automatically, ET 285–286
  PWB
    adding files, ET 44, 48, 50
    closing, ET 234
    contents, ET 43–44
    creating, ET 42
    defined, ET 41
    deleting files, ET 48
    dependencies, ET 45, 48
    editing, ET 47–49
    extending, ET 58–61
    makefiles, ET 56, 58
    menu commands, ET 74
    moving files, ET 49

Projects (*continued*)
  PWB (*continued*)
    opening, ET 195–196
    status files, ET 138–139
  using, ET 49
Promoting
  data types, portability guidelines
  floating-point types, PT 129
Promotions, C++
  integral, described, LR+ 66–67
Prompt function, PWB, ET 116–117, 152, 196–197
Prompts
  Askexit switch, PWB, ET 267
  Askrtn switch, PWB, ET 267
  LIB, ET 698–699
  LINK, ET 573–576
Propagating
  constants, PT 9
Protected members, LR+ 295–296; TUT 130–131, 179
Protected mode
  defined, ET 803, 814
  module-definition files, specifying, ET 617
  optimizing entry/exit codes, ET 515
PROTMODE statement
  module-definition files, ET 609, 617
Prototype scope
  ANSI compliant, LR 263
Prototypes
  See also Function prototypes
  arguments, LR 167
  comparing types, LR 166
  direct-declarator, LR 168
Prototypes, C++
  defined, LR+ 155
  function, TUT 9–10
Psearch function, PWB, ET 87, 107, 153, 197
Pseudofiles
  creating, in PWB, ET 187–188, 245
  Saveall function, PWB, ET 209
Pseudoinstructions
  _emit, PT 115
Pseudotargets, NMAKE, ET 658
_psp variable, LIB 66–67
PtInRect member function
  CRect class, XRF 528
PtInRegion member function
  CRgn class, XRF 548
PTR operator
  debugging assembly language, ET 414–415
ptrdiff_t type
  defining size of integral value, LR 129
PtVisible member function
  CDC class, XRF 223
public
  base classes, TUT 111, 131
  members, TUT 45
Public names
  overlaid DOS programs, ET 604
  restricting length, CL option, ET 525
Public symbols
  searching, CodeView, ET 406–407
Punctuation, C character set, LR 23
Punctuators, C++, LR+ 9–10
Pure virtual functions, TUT 127–129
Pushbutton control,dialog boxes
  changing default, CDialog::SetDfID, XRF 272
  getting default ID, CDialog::GetDefID, XRF 268
put areas
  first byte of, returning, streambuf::pptr, XRF 928
  number of characters available for fetching, returning, streambuf::out_waiting, XRF 926
  pointer to byte after last, returning, streambuf::epptr, XRF 925
  pointer to start of, returning, streambuf::pbase, XRF 928
  setting pointer values, streambuf::setp, XRF 932
  storing character, streambuf::sputc, XRF 934
put member function
  ofstream class, XUG 375
  ostream class, XRF 903
Put pointers
  following stored characters, streambuf::sputn, XRF 935
  incrementing, streambuf::pbump, XRF 928
putback member function
  istream class, XRF 882
putc function, LIB 593–594
_putch function, LIB 595–596
putchar function, LIB 593–594
_putenv function, LIB 597–599
_putimage function, LIB 600–601; PT 192
_putimage_w function, LIB 600–601; PT 193
puts function, LIB 602
Putting strings to the console, _cputs function, LIB 167
_putw function, LIB 603–604

- PWB
  - Browse menu
    - described, ET 76
    - functions, ET 145, 200
  - browser database
    - browser utilities, ET 731
  - building programs, XUG 9
  - command line, ET 141–142
  - commands
    - choosing, ET 78–79
    - cursor movement, ET 154–155
    - executing, ET 78–82, 142, 170, 219
  - configuration
    - autoloading, ET 131
    - environment variables, ET 137
    - overview, ET 130–131
  - customizing colors, ET 124–125
  - debug mode, XUG 12
  - DOS programs
    - running, XUG 13
  - Edit menu
    - described, ET 73
    - predefined macros, ET 143
  - error messages. *See* "Error Messages," Part 2 of this book
  - File menu
    - described, ET 72
    - predefined macros, ET 142
  - files
    - adding, ET 48, 50
    - deleting, ET 48
    - estimating size, ET 103–104
    - moving, ET 49
  - floating-point math packages options, PT 134
  - functions
    - Assign, ET 121–124
    - Backtab, ET 127–128
    - described, ET 150–221
    - executing, ET 106–108
    - modifying, ET 181
    - Prompt, ET 116–117
    - Set Switch, ET 123
    - Tab, ET 127–128

- PWB (*continued*)
  - Help
    - copying and pasting, ET 761, 771
    - getting, ET 756–765
    - global searches, ET 766–767
    - keywords, ET 762
    - managing files, ET 772–773
    - opening files, ET 765–766
    - structure, ET 755
  - Help menu, ET 78, 146, 757
  - HELPMAKE restrictions, ET 709
  - key assignments, ET 146–150
  - Language Options menu, PT 134
  - linking floating-point math libraries, PT 137
  - makefiles
    - DMTEST sample program, XUG 65
    - filenames, XUG 8
    - Foundation classes tutorial, XUG 7
    - HELLO sample program, XUG 109
    - loading, ET 142
    - opening, ET 141
  - macros
    - changing key assignments, ET 119–121, 135
    - executing, ET 106–108, 170
    - flow control statements, ET 112–114
    - overview, ET 109
    - recording, ET 109–112
    - user input statements, ET 114–117
  - memory model, selecting, PT 62
  - multimodule programs, ET 45, 49
  - optimization, controlling, PT 5–6
  - options, ET 141–142
  - Options menu, ET 75
  - p-code compiling, PT 43–44
  - predefined macros, ET 142–146, 222–224, 227–263
  - programs, ET 45–46, 53–56
    - adding files, ET 44, 50
    - build process, ET 56–58
    - creating projects, ET 42
    - editing, ET 47–49, 55
    - extending projects, ET 58–61
    - non-PWB makefiles, ET 61–63
    - overview, ET 41–42

PWB (*continued*)
programs (*continued*)
project contents, ET 43–44
project dependencies, ET 48
running, ET 46–47
project file list, ET 43
Project menu, ET 74
predefined macros, ET 144
prompt
Askexit switch, ET 267
Askrtn switch, ET 267
QuickWin programs, compiling, PT 156
quitting, ET 47, 171, 251
regular expressions
syntax, ET 303–304
Run menu
adding commands, ET 125–127
described, ET 74
predefined macros, ET 144
screen display, ET 67–68
Search menu
described, ET 73
predefined macros, ET 144
searching
find command, ET 87–90
mark function, ET 86
overview, ET 85–86
regular expressions, ET 90–93
single-module programs
debugging, ET 29
source browser, ET 200
browser database, ET 61
building database, ET 101
call tree, showing, ET 99–101
CL options, ET 507–508
combined database, ET 106
creating database, ET 97–98
estimating file size, ET 103–104
finding symbols, ET 98, 101, 103
non-PWB project database, ET 104–106
starting, ET 65–67
status files, ET 138–139
changing, ET 122, 124
Filename-Parts syntax, ET 265–266
help, ET 313–315
library, ET 310–312
source browser, ET 309–310
syntax, ET 141–142
Boolean switches, ET 266
Filename-Parts Syntax, ET 265–266
tabs, ET 127–130, 132–133

PWB (*continued*)
text, replacing, ET 93–96
TOOLS.INI file, ET 113, 115
line continuation, ET 136
switch syntax, ET 135
tags, ET 132–134
undefined macros, ET 224, 226
View menu, ET 770
Window menu
described, ET 77
predefined macros, ET 145
PWB Windows command, PWB, ET 77
_pwbarrange predefined macro, ET 222, 227–228
_pwbboxmode predefined macro, ET 222, 228
_pwbbuild predefined macro, ET 222, 229
PWBC library switches, ET 310–312
_pwbcancelbuild predefined macro, ET 222, 229–230
_pwbcancelprint predefined macro, ET 222, 230
_pwbcancelsearch predefined macro, ET 222, 230–231
_pwbcascade predefined macro, ET 222, 231–232
_pwbclear predefined macro, ET 222, 232
_pwbclose predefined macro, ET 222
_pwbcloseall predefined macro, ET 222, 232–233
_pwbclosefile predefined macro, ET 222, 233
_pwbcloseproject predefined macro, ET 222, 233–234
_pwbcompile predefined macro, ET 222, 234
_pwbfile predefined macro, ET 222
_pwbgotomatch predefined macro, ET 222, 235
Pwbhelp function, ET 153, 198
_pwbhelp_again predefined macro, ET 222, 236
_pwbhelp_back predefined macro, ET 222, 237
_pwbhelp_contents predefined macro, ET 222, 237–238
_pwbhelp_context predefined macro, ET 222, 238
_pwbhelp_general predefined macro, ET 222, 239
_pwbhelp_index predefined macro, ET 222, 239–240
_pwbhelpn1 predefined macro, ET 222
Pwbhelpnext function, ET 153, 198–199
_pwbhelpnl predefined macro, ET 235
Pwbhelpsearch function, ET 153, 199
_pwbhelp_searchres predefined macro, ET 222, 240
_pwblinemode predefined macro, ET 222, 241
_pwblogsearch predefined macro, ET 222, 241–242
_pwbmaximize predefined macro, ET 223, 242
_pwbminimize predefined macro, ET 223, , 243–244
_pwbmove predefined macro, ET 223, 244
_pwbnewfile predefined macro, ET 223, 245

- _pwbnewwindow predefined macro, ET 223, 245–246
- _pwbnextfile predefined macro, ET 223, 246
- _pwbnextlogmatch predefined macro, ET 223, 247
- _pwbnextmatch predefined macro, ET 223, 247–248
- _pwbnextmsg predefined macro, ET 223, 248
- _pwbpreviouslogmatch predefined macro, ET 223, 248–249
- _pwbpreviousmatch predefined macro, ET 223, 249–250
- _pwbprevmsg predefined macro, ET 223, 250
- _pwbprevwindow predefined macro, ET 223, 250
- _pwbquit predefined macro, ET 223, 251
- _pwbrebuild predefined macro, ET 223, 252
- _pwbrecord predefined macro, ET 223, 252–253
- _pwbredo predefined macro, ET 223, 253
- _pwbrepeat predefined macro, ET 223, 253–254
- _pwbresize predefined macro, ET 223, 254–255
- _pwbrestore predefined macro, ET 223, 255
- PWBRMAKE.EXE, ET 731–734
- Pwbrowse1stdef function, ET 153, 200
- Pwbrowse1stref function, ET 153, 200
- Pwbrowsecalltree function, ET 153, 200
- Pwbrowseclhier function, ET 153, 200
- Pwbrowsecltree function, ET 153, 200
- Pwbrowsefuhier function, ET 153, 200
- Pwbrowsegotodef function, ET 153, 200
- Pwbrowsegotoref function, ET 153, 200
- Pwbrowselistref function, ET 153, 200
- Pwbrowsenext function, ET 153, 200
- Pwbrowseoutline function, ET 153, 200
- Pwbrowsepop function, ET 153, 200
- Pwbrowseprev function, ET 153, 200
- Pwbrowseviewrel function, ET 153, 200
- Pwbrowsewhref function, ET 153, 200
- _pwbsaveall predefined macro, ET 223, 255–256
- _pwbsavefile predefined macro, ET 223, 256
- _pwbsetmsg predefined macro, ET 223, 257
- _pwbshell predefined macro, ET 223, 257–258
- _pwbstreammode predefined macro, ET 223, 258
- _pwbtile predefined macro, ET 223, 258–259
- _pwbundo predefined macro, ET 223, 259
- _pwbusern predefined macro, ET 223, 260
- PWBUTILS, PWB Options menu, ET 75
- _pwbviewbuildresults predefined macro, ET 223, 261
- _pwbviewsearchresults predefined macro, ET 223, 261–262
- Pwbwindow function, ET 153, 201
- _pwbwindow predefined macro, ET 223, 262–263
- Pword function, PWB, ET 153, 201
- pword member function
  - ios class, XRF 861
- .PXE files. *See* LINK

Q

- Q command, CodeView, ET 423, 449
- /Q option
  - EXP, ET 750
  - LINK, ET 590–591
  - NMAKE, ET 649–650
- q option, optimize pragma, PT 50
- _QC macro, C++, LR+ 376
- /qc option, CL, ET 540; LR 211
- QH command, MS-DOS, ET 768–769
- QHELLO.C sample QuickWin program, PT 154–155
- .QLB files, ET 814
- Qreplace function, PWB, ET 153, 202
- qsort function, LIB 605–606
- Qualified names, C++
  - described, LR+ 282–283
  - primary expressions, LR+ 80
- Qualifiers, p-code instructions, PT 48
- Question mark (?)
  - call tree, PWB, ET 100
  - decorated names, C++, ET 409
  - Display Expression command, CodeView, ET 424, 477–478
  - escape sequence, LR 18
  - filename macros, NMAKE, ET 672–673
  - Quick Watch command, CodeView, ET 424, 478–479
  - SBRPACK syntax, ET 740
  - usage, LR 31

Question mark (?) (*continued*)
- wildcard character
  - HELPMAKE syntax, ET 711
  - NMAKE, ET 653–654
  - regular expression syntax, ET 780, 786
  - UNDEL, ET 749

Queue collections, creating, Foundation classes cookbook, XUG 276

Quick Compile option, CL, ET 540

Quick Watch command, CodeView, ET 364, 367–368, 424, 478–479

Quick Watch dialog box
- CodeView
  - described, ET 367–368
  - displaying, ET 478–479
  - exploring watch expressions, ET 327

QuickBASIC, mixed-language programming, PT 242

QuickHelp
- BSCMAKE option, ET 736
- CL option, ET 525
- commands, ET 770
- copying text, ET 771
- CVPACK option, ET 745
- EXEHDR option, ET 630
- EXP option, ET 750
- format
  - defining topics, ET 716–717
  - described, HELPMAKE, ET 716
  - dot commands, ET 722–724
  - formatting attributes, ET 718–719, 721
  - global contexts, ET 719–720
  - linking topics, ET 717–719
  - local contexts, ET 720
- help files, opening and closing, ET 769
- /HELP option, ET 768
- HELPMAKE option, ET 715
- IMPLIB option, ET 747
- LIB option, ET 701
- LINK option, ET 581
- NMAKE option, ET 648
- pasting text, ET 771
- QH command, MS-DOS, ET 768–769
- SBRPACK option, ET 740
- specifying format, HELPMAKE, ET 713
- topics
  - displaying, ET 769
  - navigating, ET 770
  - selecting, ET 763
- UNDEL option, ET 749

/QUICKLIBRARY option, LINK, ET 590–591

QuickWin
- About command
  - described, PT 151
  - dialog box, customizing, PT 151, 157
- active window
  - described, PT 152
  - setting, PT 161–162
- Arrange Icons command, PT 150
- buffer size, PT 161
- Cascade command, PT 150
- child windows
  - closing, PT 162
  - displaying, PT 147–148
  - (list), PT 150
  - opening, PT 151, 157–158
  - reading from, PT 159–160
  - sizing, positioning, PT 153, 160
  - writing to, PT 159–160
- CL option, ET 528
- Clear Paste command, PT 150
- closing window's file handle, _wclose function, LIB 865–866
- compiling
  - from PWB, PT 156
  - from the command line, PT 154–155
- Copy command, PT 149
- Copy Tabs command, PT 149
- described, PT 145
- Exit command, PT 148
- exiting
  - closing all windows, PT 162
  - leaving windows open, PT 162–163
- functions, LIB xi, 53
- Help file, PT 165
- icons, customizing, PT 164–165
- Index command, PT 151
- Input command, PT 150
- libraries, PT 145
- limitations, PT 153
- Mark command, PT 148–149
- menu items, choosing, _wmenuclick function, LIB 880–881
- menus
  - controlling, PT 152
  - Edit, PT 148–149
  - File, PT 148
  - Help, PT 151
  - simulating mouse clicks in, PT 163–164
  - State, PT 149
  - Window, PT 149–150, 163–164

QuickWin (*continued*)
mouse clicks, simulating, PT 163–164
Paste command, PT 149
Pause command, PT 149
program exit behavior, _wgetexit function, LIB 871–872
programs
enhanced, creating, PT 146–147, 157–165
exiting, PT 148
running, PT 154
simple, creating, PT 146
Resume command, PT 149
sample programs
QHELLO.C, PT 154–155
QWDEMO.C, PT 157
screen buffer, PT 161
Select All command, PT 149
setting strings for About dialog boxes, _wabout function, LIB 863–864
specifying exit behavior of application, _wsetexit function, LIB 889–891
Status Bar command, PT 150
Tile command, PT 150
user interface described, PT 147–148
Using Help command, PT 151
windows
activating, _wsetfocus function, LIB 892–893
getting current screen-buffer size, _wgetscreenbuf function, LIB 875–876
getting current size, position, _wgetsize function, LIB 877–879
getting file handles, _wgetfocus function, LIB 873–874
opening, _wopen function, LIB 882–884
setting screen buffer size, _wsetscreenbuf function, LIB 894–895
setting size, screen position, _wsetsize function, LIB 896–897
yielding processor control for Windows queue servicing, LIB 898–899
yielding to other applications, PT 164
Quit command, CodeView, ET 423, 449
Quitting
CodeView, ET 360
PWB, ET 47, 171, 251
Quotation marks (")
character strings, ET 805
CodeView syntax, ET 340
escape sequence, LR 18
LINK syntax, ET 565
long filenames, NMAKE, ET 654
module statement syntax, ET 610–611
Pause command, CodeView, ET 423, 470
use of, LIB xv; XUG ix; LR xiv, 18, 21, 32
Quote function, PWB, ET 153, 203
QUOTE p-code instruction, PT 46
Quoted string, predefined expression syntax, ET 778, 780, 786
Quotients, computing, ldiv and ldiv_t functions, LIB 449–450
Quoting, p-code
controlling, PT 52
described, PT 46
disabling, PT 46
QWDEMO.C sample QuickWin program, PT 157
QWIN.HLP file, PT 165

R

R command, CodeView, ET 423
:r command, HELPMAKE, ET 724
\r (escape sequence), carriage return, LR 18
/R option
CodeView, ET 396
EXEHDR, ET 631
EXP, ET 750
NMAKE, ET 650
PWB, ET 142
RM, ET 748
/r option
BSCMAKE, ET 737
CL, ET 562
Radio buttons
check-marking, CWnd::CheckRadioButton, XRF 662

Radio buttons (*continued*)
  CWnd, retrieving ID of check-marked, CWnd::GetCheckedRadioButton, XRF 686
  getting check state, CButton::GetCheck, XRF 132
  setting
    check state, CButton::SetCheck, XRF 134
    highlighting control, CButton::GetState, XRF 132
Radix
  changing in CodeView, ET 444–445
  CodeView expression evaluators, ET 407–408
  command, CodeView, ET 422, 444–445
  defined, ET 814
raise function, LIB 607–608
RAM defined, ET 814
RAMDrive. *See Getting Started*
rand function, LIB 609–610
Random
  number generation, rand function, LIB 609–610
  starting point, setting, srand function, LIB 729–730
Random access memory, ET 814
Ranges
  floating-point types, LR 145
  integers, LR 238
  reading, LR 252
Raster operations, XRF 172
Raster-operation codes (list), XRF 167–168
rate option, CodeView, ET 396
.raw command, HELPMAKE, ET 724
.RC files, ET 814
RC.HLP file, ET 771
RCVCOM option, CodeView, ET 396
RCVCOM.EXE file, remote debugging, ET 395
RCVWCOM option, CodeView, ET 396
rdbuf member function
  fstream class, XRF 843
  ifstream class, XRF 850
  ios class, XRF 862
  istrstream class, XRF 892
  ofstream class, XRF 898
  ostrstream class, XRF 913
  stdiostream class, XRF 918
  strstream class, XRF 940
rdstate member function
  ios class, XRF 862
  ofstream class, iostream classes tutorial, XUG 377
_read function, LIB 611–612
Read member function
  CArchive class, XRF 99
  CFile class, XRF 314
    CStdioFile::ReadString, XRF 570
    Foundation classes cookbook, XUG 278
read member function
  input streams, iostream classes tutorial, XUG 389
  istream class, XRF 882–883
Read Only command, PWB, ET 73
Reading
  archives
    object data, CArchive::ReadObject, XRF 99
    specified number of bytes, CArchive::Read, XRF 99
  characters from streams, getc and getchar functions, LIB 346–347
  console data, _cscanf function, LIB 171–172
  data in CFile object file buffers, CFile::Read, XRF 314
  file data
    _dos_open function, LIB 214–215
    _read function, LIB 611–612
  formatted data
    from input stream, scanf function, LIB 635–639
    from strings, sscanf function, LIB 731–732
  object to archive, CObject::Serialize, XRF 473–474
  stream data
    fread function, LIB 304–305
    fscanf function, LIB 316–317
  text data into buffer from file associated with CStdioFile object, CStdioFile::ReadString, XRF 569–570
README, Foundation classes tutorial, XUG 15
ReadObject member function, CArchive class, XRF 99–100
  CObject::Serialize, XRF 473
READONLY keyword, module-definition files, ET 621
Readonly switch, PWB, ET 122, 264–266, 291–292
ReadString member function, CStdioFile class, XRF 569–570
READVC.C sample graphics program, PT 171–172
READWRITE keyword, module-definition files, ET 621
Real mode
  defined, ET 814
  specifying, module-definition files, ET 617
  Windows, specifying, LR 260
RealizePalette member function, CDC class, XRF 224
  CGdiObject::UnrealizeObject, XRF 350
  CPalette::SetPaletteEntries, XRF 507
realloc functions, LIB 613–615
Reallocating memory blocks, realloc functions, LIB 613–615

- REALMODE statement, module-definition files, ET 609, 617
- Realtabs switch, PWB, ET 127–128, 264, 292
- Rebuild All command, PWB, predefined macros, ET 144
- Rebuilding, _pwbrebuild macro, ET 252
- Record function, PWB, ET 153, 203–204
- Record On command, PWB
  - described, ET 73
  - predefined macros, ET 143
- Record Results, PWB, ET 77
- Recording macros, PWB, ET 109–112, 203–204, 252–253
- Records
  - inline assembly limitations, PT 113
  - mixed-language programming, PT 265
- RECT structure, XRF 86
  - CButton::Create, XRF 128
  - CComboBox::Create, XRF 143
  - CDC::DPtoLP, XRF 175
  - CDC::DrawFocusRect, XRF 175
  - CDC::DrawText, XRF 177
  - CDC::ExtTextOut, XRF 188
  - CDC::FillRect, XRF 189
  - CDC::FrameRect, XRF 191
  - CDC::GetClipBox, XRF 195
  - CDC::IntersectClipRect, XRF 212
  - CDC::InvertRect, XRF 213
  - CDC::LPtoDP, XRF 215
  - CDC::Pie, XRF 220
  - CDC::Rectangle, XRF 225
  - CDC::RectVisible, XRF 225
  - CDC::RoundRect, XRF 227
  - CDC::ScrollDC, XRF 230
  - CDialog::MapDialogRect, XRF 270
  - CEdit::GetRect, XRF 293
  - CEdit::SetRect, XRF 299
  - CEdit::SetRectNP, XRF 300
  - CListBox::Create, XRF 356
  - CListBox::GetItemRect, XRF 363
  - CMenu::TrackPopupMenu, XRF 437
  - CRect::CopyRect, XRF 523
  - CRect::CRect, XRF 524
- RECT structure (*continued*)
  - CRect::EqualRect, XRF 525
  - CRect::IntersectRect, XRF 526
  - CRect::operator&, XRF 535
  - CRect::operator&=, XRF 533
  - CRect::operator|, XRF 535
  - CRect::operator|=, XRF 533
  - CRect::UnionRect, XRF 530
  - CRgn::CreateEllipticRgnIndirect, XRF 541
  - CRgn::CreateRectRgnIndirect, XRF 544
  - CRgn::GetRgnBox, XRF 547
  - CRgn::RectInRegion, XRF 549
  - CRgn::SetRectRgn, XRF 549
  - CScrollBar::Create, XRF 553
  - CStatic::Create, XRF 563
  - CWnd::BeginPaint, XRF 660
  - CWnd::ClientToScreen, XRF 663
  - CWnd::GetClientRect, XRF 686
  - CWnd::GetUpdateRect, XRF 700
  - CWnd::GetWindowRect, XRF 703
  - CWnd::InvalidateRect, XRF 707
  - CWnd::MoveWindow, XRF 714
  - CWnd::OnNcCalcSize, XRF 762
  - CWnd::OnSizeClipboard, XRF 785
  - CWnd::ScreenToClient, XRF 802
  - CWnd::ScrollWindow, XRF 803
  - CWnd::ValidateRect, XRF 820
- _rectangle function, LIB 616–617; PT 169, 190
- Rectangle member function, CDC class, XRF 224–225
- Rectangles
  - bounding
    - coordinate systems, PT 185
    - copying dimensions, CWnd::GetWindowRect, XRF 703–704
    - list boxes, retrieving dimensions, CListBox::GetItemRect, XRF 363
    - of CRgn object, retrieving, CRgn::GetRgnBox, XRF 547
    - retrieving dimensions around clipping boundary, CDC::GetClipBox, XRF 195
  - calculating width of CRect, CRect::Width, XRF 530

Rectangles (*continued*)
- checking if within region, CRgn::RectlnRegion, XRF 549
- converting between CRect and LPRECT, CRect::operator, XRF 531
- copies dimensions of scrRect to CRect, CRect::operator=, XRF 531
- copying to CRect, CRect::CopyRect, XRF 523
- creating
  - CRect object, CRect::CRect, XRF 524
  - new clipping region, CDC::ExcludeClipRect, XRF 185
  - NULL, CRect::SetRectEmpty, XRF 529
- CRect class described, XRF 521
- CWnd, validating client area, CWnd::ValidateRect, XRF 820
- defining upper-left and lower-right corner coordinates, RECT structure, XRF 86
- determining
  - equality between two, CRect::EqualRect, XRF 525
  - equality to CRect, CRect::operator==, XRF 531
  - if empty, CRect::IsRectEmpty, XRF 527
  - if top, left, bottom and right values equal 0, CRect::IsRectNull, XRF 527
  - if within clipping region, CDC::RectVisible, XRF 225
  - inequality, CRect::operator!=, XRF 532
  - size, CRect::Size, XRF 529
  - whether specified point lies within, CRect::PtlnRect, XRF 528
- drawing
  - borders, CDC::FrameRect, XRF 191
  - _rectangle functions, LIB 616–617
  - style indicating focus, CDC::DrawFocusRect, XRF 175
  - text in, CDC::DrawText, XRF 177–179
  - with current pen, CDC::Rectangle, XRF 224
  - with rounded corners, CDC::RoundRect, XRF 226–227
- enclosing update region, retrieving coordinates, CWnd::GetUpdateRect, XRF 700–701
- filling with specified brush, CDC::FillRect, XRF 189–190
- formatting. *See* Formatting rectangle
- gray, creating for system caret, CWnd::CreateSolidCaret, XRF 672
- height, calculating, CRect::Height, XRF 525
- inflating or deflating, CRect::InflateRect, XRF 525

Rectangles (*continued*)
- intersecting CRect with rect2
  - CRect::operator&, XRF 535
  - CRect::operator|, XRF 535
- invalidating client areas within, CWnd::InvalidateRect, XRF 707
- inverting contents, CDC::InvertRect, XRF 213
- making CRect equal to intersection of two rectangles, CRect::IntersectRect, XRF 526
- making dimensions equal to union of two rectangles, CRect::UnionRect, XRF 530
- moving bitmaps from source to destination, CDC::StretchBlt, XRF 255–257
- moving
  - CRect::OffsetRect, XRF 527
  - CRect::operator+=, XRF 532
  - CRect::operator-=, XRF 533
- RECT structure, XRF 86
- referencing
  - bottom-right point, CRect::BottomRight, XRF 523
  - top-left point, CRect::TopLeft, XRF 529
- regions, creating
  - CRgn::CreateRectRgn, XRF 544
  - CRgn::SetRectRgn, XRF 549–550
  - indirect, CRgn::CreateRectRgnIndirect, XRF 545
- returning new rect equal to CRect plus point
  - CRect::operator+, XRF 534
  - CRect::operator-, XRF 534
- scrolling, CDC::ScrollDC, XRF 230
- setting dimensions
  - CRect::SetRect, XRF 528
  - CRect to equal intersection with rect, CRect::operator&=, XRF 533
  - equal to union with rect, CRect::operator|=, XRF 533
  - multiple-line edit control, CEdit::SetRect, XRF 299
- structure, copying client coordinates of CWnd client area into, CWnd::GetClientRect, XRF 686–687

_rectangle_w function, LIB 616–617; PT 190

_rectangle_wxy function, LIB 616–617; PT 191

rectDefault data member, CFrameWnd class, XRF 341

RectInRegion member function, CRgn class, XRF 549

RectVisible member function, CDC class, XRF 225

Recursion macros, NMAKE, ET 674–675

Recursive functions, LR 188, 211
Red, color value, ET 273
_RED constant, PT 180
Redefining
keywords, LR 4
macros, LR 192
manifest constants, LR 192
members of base class, TUT 114–115
Redirect Input command, CodeView, ET 340, 424, 475, 477
Redirect Output command, CodeView, ET 340, 424, 476–477
Redirection, ET 814
Redo command, PWB, ET 73, 143
Redraw command, CodeView, ET 424, 479
Redrawing
allowing or preventing changes, CWnd::SetRedraw, XRF 811
menu bars, CWnd::DrawMenuBar, XRF 681
Reduced programming surface area, XRF 10
Reducing program size
optimization. *See* Optimization
p-code. *See* P-code
.ref command, HELPMAKE, ET 724
References
compared with pointers, TUT 27–29
declarators, LR+ 190–196
guidelines, TUT 31–32
initialization,LR+ 223–224; TUT 27, 77
overview, TUT 25–27
parameters, TUT 29–31, 72, 79, 81
returning, TUT 32, 55–56, 76, 82–83
to classes, conversion to references to base classes, LR+ 75
to objects, types, LR+ 53–54
Referencing
bit fields, LR 69
declarations, LR 45–47
Refresh function, PWB, ET 153, 204
Regions, checking
equivalent, CRgn::EqualRgn, XRF 546
if coordinates are within, CRgn::PtlnRegion, XRF 548

Regions, checking (*continued*)
if rectangle within CRgn object, CRgn::RectlnRegion, XRF 549
clipping. *See* Clipping region
combining, CRgn::CombineRgn, XRF 540
copying, CRgn::CopyRgn, XRF 540
creating
by combination, CRgn::CombineRgn, XRF 539–540
constructor, CRgn::CRgn, XRF 545
rectangular, CRgn::CreateRectRgn, XRF 544
rectangular, indirect, CRgn::CreateRectRgnIndirect, XRF 545
series of polygonal, CRgn::CreatePolyPolygonRgn, XRF 543
CRgn class described, XRF 537
drawing borders around, CDC::FrameRgn, XRF 192
elliptical, creating
CRgn::CreateEllipticRgn, XRF 541
CRgn::CreateEllipticRgnIndirect, XRF 541
filling
with brush, CDC::PaintRgn, XRF 218
with specified brush, CDC::FillRgn, XRF 190
handles, CRgn::FromHandle, XRF 546
invalidating client areas within, CWnd::InvalidateRgn, XRF 708
moving, CRgn::OffsetRgn, XRF 547
polygonal, creating, CRgn::CreatePolygonRgn, XRF 542
preventing drawing within areas, CDC::ExcludeUpdateRgn, XRF 186
rectangular, creating, CRgn::SetRectRgn, XRF 549–550
retrieving bounding rectangle coordinates, CRgn::GetRgnBox, XRF 547
update
retrieving coordinates of smallest rectangle that encloses, CWnd::GetUpdateRect, XRF 700–701
retrieving into specified region, CWnd::GetUpdateRgn, XRF 701
writing character strings within, CDC::ExtTextOut, XRF 188–189

Register allocation
  optimization, PT 21–22
  portability guidelines, PT 289
Register calling convention, PT 30
Register command, CodeView, ET 373–374, 391–392, 423
Register indirection, debugging assembly language, ET 414
register keyword, PT 124–125, 288–289
register keyword, C++
  declaration statements, LR+ 149–151
  declarations, use in, LR+ 157–158
  described, LR+ 46–47
Register names, CodeView recognition, ET 400–401, 419
register storage-class specifier
  described, LR 48
  external level, invalid, LR 45
  in parameters, LR 86
  lifetime, LR 37, 44
  nonterminal, LR 43
  visibility, LR 37
Register values, getting, _dosexterr function, LIB 234–235
Register variables
  portability guidelines, PT 288–289
  storage, __asm block effect on, PT 124–125
Register variables, C++, LR+ 46–47
Register window
  CodeView
    described, ET 327
    function, ET 354–355
    opening, ET 374
    overview, ET 348
    debugging p-code, ET 391–392
  defined, ET 814
_registerfonts function, LIB 618; PT 195–196
Registering fonts, PT 195–196
Registering routine to be called on exit, _fonexit, and_onexit functions, LIB 531–532
Registering windows classes, AfxRegisterWndClass, XRF 37
Registers
  AL
    fastcall functions, LR 170
    for return values, LR 171
  allocating, CL options, ET 533
  __asm blocks, PT 120–121
  availability, LR 242

Registers (*continued*)
  AX
    16-bit compiler, LR 58
    fastcall functions, LR 170
    for return values, LR 171
  BL, fastcall functions, LR 170
  BX
    16-bit compiler, LR 58
    fastcall functions, LR 170
  calling conventions, CL options, ET 520
  changing values, CodeView, ET 450–452
  CodeView expressions, ET 400–401, 419
  CPU, saving, LR 176
  defined, ET 814
  DI, variables for version 7.0, LR 48
  display radix, ET 444–445
  displaying value, CodeView, ET 354–355
  DL, fastcall functions, LR 170
  DS
    loading values, LR 175–176
    with based pointers, LR 81
  DX
    16-bit compiler, LR 58
    fastcall functions, LR 170
  DX:AX
    fastcall functions, LR 170
    for 4-byte return values, LR 171
  EBX, variables for version 7.0, LR 48
  ECX
    32-bit compiler, LR 58
    argument passing, LR 171
  EDI, variables for version 7.0, LR 48
  EDX
    32-bit compiler, LR 58
    argument passing, LR 171
  ESI, variables for version 7.0, LR 48
  fastcall functions, LR 170–171
  flags defined, ET 809
  math coprocessors, dumping registers, ET 473–474
  p-code, PT 45–46
  SI, variables for version 7.0, LR 48
  storage, LR 48
Registration, Windows classes, XRF 12
  attributes, changing, XUG 325
  attributes, passing on, XUG 326
  described, XUG 325
  Foundation classes tutorial, XUG 112
  key attributes, XUG 325

Regular expressions
  defined, ET 814
  finding, CodeView, ET 361
  global searches, in Microsoft Advisor, ET 766
  matching
    non-UNIX, ET 788
    PWB, ET 307–308
  predefined. *See* Predefined expressions
  replacing text, PWB, ET 93–96
  searching for, CodeView, ET 472–473
  searching, PWB, ET 85–86, 90–93
  syntax
    CodeView, ET 779
    non-UNIX, ET 780, 786
    PWB, ET 92
    UNIX, ET 777–781, 785
  tagged. *See* Tagged expressions
Relational operators, LR 132–133
Relational operators, C++
  binary-operator expressions, LR+ 107–110
  overloading, LR+ 359
Relationships between classes, TUT 176
Release mode
  debug mode, switching to, XUG 12
  makefile defaults, XUG 9
ReleaseBuffer member function, CString class, XRF 588–589
  Foundation classes cookbook, XUG 260
ReleaseDC member function, CWnd class, XRF 801–802
  CDC::DeleteDC, XRF 174
Releasing, device contexts, CWnd::ReleaseDC, XRF 801–802
Releasing memory block, _dos_freemem function, LIB 194–195
Relocatable files, LINK, ET 563
Relocatable Object-Module Format, LINK, ET 563
Relocation table, optimizing, LINK, ET 580
Relocations, EXEHDR, ET 639
Remainders, LR 239
_remapallpalette function, LIB 619–623; PT 188
_remappalette function, LIB 619–623; PT 188
Remapping palette colors, _remapallpalette and _remappalette functions, LIB 619–623
Remote debugging
  bit rate, ET 396
  options, ET 396
  overview, ET 393
  requirements, ET 393–395
  starting a session, ET 397–398
  syntax, ET 396
Remove Custom Project Templates command, PWB, ET 75
remove function, LIB 624
Remove member function, CFile class, XRF 315
RemoveAll member function
  CMapStringToOb class, XRF 383
  CObArray class, XRF 458
  CObList class, XRF 493–494
RemoveAt member function
  CObArray class, XRF 458–459
  CObList class, XRF 494–495
RemoveHead member function, CObList class, XRF 495
RemoveKey member function, CMapStringToOb class, XRF 383–384
RemoveMenu member function, CMenu class, XRF 434
RemoveTail member function, CObList class, XRF 496
Removing
  breakpoints, CodeView, ET 367
  directories, _rmdir function, LIB 629–630
  elements from arrays, CObArray::RemoveAt, XRF 458–459
  files
    remove function, LIB 624
    temporary, _rmtmp function, LIB 631–632
  invariant code, ET 535–536, PT 18–19
  items from list boxes, CListBox::ResetContent, XRF 367
  library name, CL option, ET 553–554
  macro names, LR 198
  macros, LR 192
  manifest constants, LR 192

Removing (*continued*)
  menu items, CMenu::RemoveMenu, XRF 434
  native entry points, p-code, PT 51
  optimizations, PT 9
  p-code entry points, ET 520–521; LR 260
  pointers from arrays, CObArray::RemoveAll, XRF 458
  predefined names, CL option, ET 542–544
  stack probes, PT 21
  status bar, CodeView, ET 347
  unreferenced functions, PT 28
rename function, LIB 625–626
Rename member function
  CFile class, XRF 315–316
Renaming
  directories, rename function, LIB 625–626
  files
    CFile::Rename, XRF 315
    executable, CL, ET 499
    object, CL, ET 498
    rename function, LIB 625–626
Repeat command, PWB, ET 73, 143
Repeat function, PWB, ET 153, 205
Repeat, regular expression syntax, ET 778–787
Repeat Last Find command, CodeView, ET 361–362
Repeating
  function actions, in PWB, ET 281–282
  statements, LR 164
Replace command
  LIB, ET 704–705
  PWB, ET 73
Replace function, PWB, ET 153, 205–207
ReplaceSel member function, CEdit class, XRF 297
Replacing
  object modules, ET 704–705
  text
    Mreplace function, PWB, ET 185
    Mreplaceall function, PWB, ET 185–186
    Qreplace function, PWB, ET 202
    Replace function, PWB, ET 205–207
  text in edit control, CEdit::ReplaceSel, XRF 297
Repositioning file pointers
  CFile::Seek, XRF 316–317
  rewind function, LIB 627–628
.RES files, ET 814
/RESERERROR option, EXEHDR, ET 631
Reserve areas
  allocating when needed, streambuf::doallocate, XRF 923
  attaching to stream's filebuf object, ifstream::setbuf, XRF 850
Reserve areas (*continued*)
  attaching to streambuf object, streambuf::setbuf, XRF 931
  returning pointer to byte after last, streambuf::ebuf, XRF 924
  returning pointer to, streambuf::base, XRF 922
  returning size in bytes, streambuf::blen, XRF 922
  setting position values with, streambuf::setb, XRF 931
  setting up streambuf::allocate, XRF 922
Reserved words, module statements, ET 611
ResetContent member function
  CComboBox class, XRF 153
    CWnd::OnDeleteItem, XRF 729
  CListBox class, XRF 367
    CWnd::OnDeleteItem, XRF 729
Resetting
  chart environment variables, PT 219
  CodeView command, ET 436–437
  edit control undo flag, CEdit::EmptyUndoBuffer, XRF 290
  floating-point packages, _fpreset function, LIB 295–298
  palette values, _pg_resetpalette function, LIB 571
  PWB, ET 141
  stream error indicator, clearerr function, LIB 150–151
  styleset to default, _pg_resetstyleset function, LIB 572
Resident option, CodeView, ET 396
Resize function, PWB, ET 153
ResizePalette member function, CPalette class, XRF 506
Resizing logical palettes, CPalette::ResizePalette, XRF 506
Resizing windows, PWB, ET 254–255
Resolution, graphics, maximizing, PT 172
Resolving ambiguities, conversions, TUT 152–155
Resource Compiler, NMAKE macros, ET 676
Resource include files
  creating, Foundation classes tutorial, XUG 242
  HELLO sample program, XUG 107, 109
Resource script (.RC) files
  completing, Foundation classes tutorial, XUG 242
  Foundation classes tutorial, XUG 156
  HELLO sample program, XUG 107–109
  QuickWin icons, PT 164–165
  Windows applications, Foundation classes cookbook, XUG 310
Resource-compiler source file, PWB, ET 43

- Resources
  - CResourceException class described, XRF 536
- Response files
  - BSCMAKE, ET 738
  - defined, ET 814
  - LIB, ET 699
  - LINK, ET 573–575
- Responsibilities
  - distributing among classes, TUT 175
  - of a class, TUT 174
- Restart command, CodeView, ET 362, 422, 436–437
- Restart macro, PWB, ET 224, 227
- Restcur function, PWB, ET 153, 208
- Restore command
  - CodeView, ET 373–374
  - PWB
    - described, ET 77
    - predefined macros, ET 145
- RestoreDC member function, CDC class, XRF 226
- Restorelayout switch, PWB, ET 264, 293
- Restoring
  - files, UNDEL, ET 749
  - MDI child window, CMDIFrameWnd::MDIRestore, XRF 408
  - optimization state, PT 9
  - stack environment and execution locale, longjmp function, LIB 466–467
  - status bar, CodeView, ET 347
  - Windows device context to previous state, CDC::RestoreDC, XRF 226
  - windows, PWB, ET 255–256
- Resume command, QuickWin, PT 149
- Retrieving
  - character line index, CEdit::LineIndex, XRF 295
  - clipboard owner, CWnd::GetClipboardOwner, XRF 687
  - scroll-bar thumb current position, XRF 555
- Return codes
  - CVPACK, ET 745
  - defined, ET 807
  - LINK, ET 596
- Return codes (*continued*)
  - NMAKE, ET 696
    - from commands, ET 662–663
    - ignoring, ET 649, 687
  - SBRPACK, ET 741
  - Windows applications, optimizing, ET 515
  - Windows functions
    - customizing, ET 515
    - generating, ET 522–523
- Return instructions, overlaid DOS programs, ET 603–604
- Return objects, addressing modes, specifying, PT 101–102
- return statements
  - containing expressions, LR 262
  - controlling execution, LR 30, 152, 166
  - described, LR 160–161
  - preferred over goto statements, LR 158
- return statements, C++
  - jump statements, LR+ 148–149
  - terminating programs
    - described, LR+ 43
    - initialization considerations, LR+ 44–45
- Return types, LR 87, 177–178
- Return values
  - floating-point types, functions, declaring, PT 130
  - inline assembly, registers, PT 120–121
  - references, TUT 32
  - successful termination, LR 30
- Reuse
  - code inheritance, TUT 170–171
  - interface inheritance, TUT 171
- ReverseFind member function, CString class, XRF 589
- Reversing characters in strings, _strrev and _fstrrev functions, LIB 780–781
- rewind function, LIB 627–628; PT 159–160
- RFLAGS options macro, NMAKE, ET 676
- Rich text format, HELPMAKE
  - described, ET 716, 725–726
  - encoding, ET 727
  - formatting codes, ET 726
  - specifying, ET 713

Key

ET	Environment and Tools	PT	Programming Techniques
LIB	Run-Time Library Reference	TUT	C++ Tutorial
LR	C Language Reference	XRF	Class Libraries Reference
LR+	C++ Language Reference	XUG	Class Libraries User's Guide

Right function, PWB, ET 153, 208
Right member function, CString class, XRF 589–590
Right shifts, LR 240
Right-shift assignment operator (>> =), LR 138
Right-shift operator, C++
  binary-operator expressions, LR+ 106–107
  overloading, LR+ 359
RM
  command line, ET 748–749
  options, ET 748
  overview, ET 743, 747–748
  syntax, ET 748–749
Rmargin switch, PWB, ET 264, 293–294
_rmdir function, LIB 629–630
_rmtmp function, LIB 631–632
Root defined, ET 815
Rotating bits
  _lrotl and _lrotr functions, LIB 468
  _rotl and _rotr functions, LIB 633–634
_rotl function, LIB 633–634
_rotr function, LIB 633–634
RoundRect member function, CDC class, XRF 226–227
Routines
  choosing functions or macros, LIB 9–11
  defined, ET 815
  described by category, LIB 17–60
  listing in CodeView, ET 372–373
  registering to be called on exit, _onexit function, LIB 531–532
.RSP files, ET 815
RTF. *See* Rich text format
Rules
  arguments, interpreting, LR 32
  assignment conversions, LR 139
  command-line arguments, LR 32–33
  complex declarators, LR 60
  conversions, LR 15, 115, 141–150
  declaring parameters, LR 166
  division operations, LR 127
  enumeration sets, LR 63
  function scope, LR 35–36
  functions, LR 34
  initializing variables, LR 92
  static, LR 45–47
  storage duration, LR 34
  type casts, LR 126
  variables, LR 34
  visibility, LR 35, 50
Rules, inference
  commands, NMAKE, ET 660
  inferred dependents, NMAKE, ET 685–686
  precedence, NMAKE, ET 686–687
  search paths, NMAKE, ET 682
  syntax, NMAKE, ET 680–682
  user-defined, NMAKE, ET 682–685
Run command, Windows Program Manager, XUG 14
Run DOS Command command, PWB, ET 74
Run member function
  CWinApp class, XRF 638
Run menu
  CodeView, ET 362–363
  PWB
    adding menu items, ET 125, 127, 304–306
    custom items, ET 260
    described, ET 74
    predefined macros, ET 144
Run OS/2 Command command, PWB, ET 74
Running
  LINK, ET 572
  NMAKE, ET 647
  programs, PWB, ET 46–47
  QuickWin programs, PT 154
  Windows programs, XUG 14
Run-time
  class structure, returning for specified class, RUNTIME_CLASS macro, XRF 474
  error messages. *See* “Error Messages,” Part 2 of this book
  returning file pointer associated with stdiobuf object, XRF 916
  stream file, CStdioFile class described, XRF 567
  structures, getting for CObject-derived class, CObject::GetRunTimeClass, XRF 470
Run-time class information, accessing, Foundation classes cookbook, XUG 265–267
Run-time error defined, ET 815
Run-time information
  supplying, DECLARE_DYNAMIC macro, XRF 38–39
  supplying dynamic, IMPLEMENT_DYNAMIC macro, XRF 40
Run-time interpreter, p-code, PT 43, 45–46
Run-time startup code, CL linking options, ET 528
RUNTIME_CLASS macro, XRF 474; XUG 265–266
R-value expressions, LR 107
R-values, C++, LR+ 60–61

S

- /S option
  - BSCMAKE, ET 737
  - CodeView, ET 338, 341, 343
  - EXEHDR, ET 631
  - NMAKE, ET 650
- same_seg pragma, precompiled header compilation, effect on, PT 41
- same_seg pragma, C++, LR+ 391
- Sample programs
  - abstract declarators, LR 88
  - addition and subtraction operators, LR 128–129
  - allocating bit fields, LR 69–70
  - ANNUITY1.C, ET 29
  - array declarations, LR 75
  - based pointer declarations, LR 79
  - blocks, LR 38
  - break statement, LR 152
  - calling variable number of arguments, LR 187
  - __cdecl calling convention, LR 170
  - CMDBOOK
    - Foundation classes tutorial, XUG 140
  - complex declarations, LR 89–91
  - compound statement, LR 153–154
  - continue statement, LR 154
  - database
    - creating, XUG 58
    - destroying, XUG 58
  - declarations and definitions, LR 28–30
  - #define preprocessor directive, LR 194–195
  - diagnostics
    - defined, XUG 21
  - distribution disks
    - Foundation classes tutorial, XUG 7
  - DMTEST
    - building, XUG 65
    - CArchive class, XUG 20
    - CFile class, XUG 20
    - CObject class, XUG 19
    - CObList class, XUG 20
    - code listings, XUG 66–79
    - CPerson class, XUG 22, 28
    - CPersonList object, XUG 37

- Sample programs (*continued*)
  - DMTEST (*continued*)
    - CString class, XUG 20, 31
    - CTime class, XUG 20
    - data object, designing, XUG 22
    - program capabilities, XUG 19
    - summary, XUG 65
    - testing, XUG 49–64
    - writing, overview, XUG 21
  - enumeration declarations, LR 63–64
  - equality operators, LR 133
  - exceptions, XUG 20
  - expression statements, LR 155–156
  - external declarations, LR 46–47
  - __far keyword, LR 169
  - for statement, LR 157
  - function called from switch statement, LR 184
  - function return values, LR 178
  - goto statement, LR 158
  - graphics
    - *See also* presentation graphics
    - ERESBOX.C, PT 168–169
    - READVC.C, PT 171–172
    - SAMPLER.C, PT 198–199
    - YELLOW.C, PT 178
  - HELLO
    - code listings, XUG 83, 107, 113–116
    - compiling, XUG 109
    - dialog boxes, adding, XUG 105–107
    - execution, sequence of, XUG 110–111
    - Foundation classes cookbook, XUG 307, 310, 312
    - overview, XUG 81–82
    - Windows, communication with, XUG 95–101
    - windows, creating, XUG 90–93
    - windows, painting text in, XUG 101–102
    - writing, overview, XUG 84
  - if statement, LR 159
  - illegal bit fields, LR 69–70
  - incomplete types, LR 101
  - initializations, LR 93
  - initializers for array, LR 95–97
  - internal- and external-level declarations, LR 49
  - logical operators, LR 136

Sample programs (*continued*)
  main function invoking macro, LR 196
  makefiles, PHBOOK, XUG 9–11
  nameless structure, LR 68
  nested structure declarations, LR 66–67
  nested unions, LR 73
  nesting, LR 38
  null statement, LR 160
  Phone Book
    ASSERT macro, XUG 127, 138
    ASSERT_VALID macro, XUG 126, 138
    code listings, XUG 141–149, 168–194, 243–246
    data interface, simplifying, XUG 122–134
    database, Windows, XUG 117
    described, XUG 118
    dialog boxes, XUG 151–167
    message handlers, XUG 197
    writing, overview, XUG 120, 122
  pointer declarations, LR 77–78
  pointers as arguments, LR 186
  presentation graphics
    BAR.C, PT 208–212
    PIE.C, PT 206–208
    SCATTER.C, PT 212–214
  PWB
    COUNT, ET 41–63, 97–103
  QuickWin
    QHELLO.C, PT 154–155
    QWDEMO.C, PT 157
  recursive calls, LR 188
  relational operators, LR 133
  return statement, LR 160–161
  return types in function declarations, LR 87
  running
    described, XUG 13
    DOS, using, XUG 13
    Windows, using, XUG 14
  __segment keyword, LR 83
  serialization
    defined, XUG 20
  simple forms of declarators, LR 55
  sizeof operator, LR 126
  stream derivation
    iostream classes tutorial, XUG 400–405
  stringizing operator, LR 196–197
  switch statements, LR 161–164
  token-pasting operators, LR 197
  two-dimentional array of structures, LR 69
  typedef & local scope identifier, same, LR 102–103

Sample programs (*continued*)
  typedef declarations, LR 102–104
  unions, LR 72
  visibility of variables, LR 38
  while statement, LR 164
SAMPLER.C sample fonts program, PT 198–199
Save All command, PWB
  described, ET 72
  predefined macros, ET 142–143
Save As command
  Phone Book sample program, XUG 207, 215
  PWB, ET 72
Save command
  Phone Book sample program, XUG 207, 215
  PWB
    described, ET 72
    predefined macros, ET 142–143
Save Custom Project Template command, PWB, ET 75
Save dialog box, standard, Foundation classes tutorial, XUG 167
Save member function, CMainWindow class
  adding, XUG 240
  described, XUG 215
Saveall function, PWB, ET 153, 209
Savecur function, PWB, ET 153, 209
SaveDC member function
  CDC class, XRF 227–228
Savescreen switch, PWB, ET 264, 294
Saving
  CodeView environment, ET 360
  device context current state, CDC::SaveDC, XRF 227
  files
    Autosave switch, PWB, ET 269
    PWB, ET 72, 209, 255–256, 300
  macros
    PWB, ET 112
  marks
    PWB, ET 287
  registers, LR 176
  space
    by suppressing _setargv, LR 33–34
__saveregs keyword
  invalid with fastcall, LR 171
  saves all CPU registers, LR 176
  using, LR 175
__saveregs keyword, C++, LR+ 422
Saving current state of program, setjmp function, LIB 666

- .SBR files
  - building browser database, ET 97–98, 105
  - defined, ET 815
  - estimating size, ET 103–104
- sbrfiles field
  - BSCMAKE, ET 735
  - SBRPACK, ET 740
- SBRPACK
  - command line, ET 740–741
  - error messages. *See* “Error Messages,” Part 2 of this book
  - exit codes, ET 741
  - options, ET 740
  - overview, ET 731, 739–740
  - suppressing
    - CL options, ET 507–508
  - syntax, ET 740
  - turning off, LR 266
    - CL option, ET 555–556
- sbumpc member function
  - streambuf class, XRF 929
- Scalar initialization, LR 91–93
- Scalar types
  - postfix operators, LR 121
- ScaleViewportExt member function
  - CDC class, XRF 228
- ScaleWindowExt member function
  - CDC class, XRF 229
- scanf function, LIB 635–639
- Scanning strings
  - for characters in specified character sets, strpbrk and _fstrpbrk functions, LIB 776–777
  - for last occurrence of characters, strrchr and _fstrrchr functions, LIB 778–779
- Scatter charts
  - analyzing data series, _pg_analyzescatter functions, LIB 553–554
  - displaying, _pg_chartscatter functions, LIB 560–561
- Scatter diagrams
  - described, PT 203–204
  - sample program, PT 212, 214
  - styles, PT 204–205
- SCATTER.C sample presentation graphics program, PT 212–214
- SCHAR_MAX constant, C++
  - integral limits, LR+ 62
- SCHAR_MIN constant, C++
  - integral limits, LR+ 62
- Scope
  - defined, ET 815
  - identifiers, LR 7, 153
  - labels in __asm blocks, PT 121–122
  - overview, LR 35–36
  - specifying
    - searching for symbols, ET 467–468
  - structures, LR 66
- Scope, C++
  - block
    - linkage rules, LR+ 34
  - class
    - described, LR+ 29
    - linkage rules, LR+ 34
    - type names in, LR+ 257
  - classes, LR+ 282–284
  - default arguments, LR+ 212, 283
  - defined, LR+ 25–26
  - described, LR+ 28–29
  - file
    - described, LR+ 29
    - linkage rules, LR+ 34
  - formal arguments, LR+ 33
  - function
    - described, LR+ 29
  - hiding names, LR+ 30–32
  - local
    - described, LR+ 28
  - overloading, LR+ 342–343
- Scope operator (::)
  - CodeView precedence, ET 406
- Scope resolution operator, TUT 15
- Screen
  - coordinates, PT 185
  - pixel cursor, PT 186
  - PWB display, ET 67–68

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User’s Guide

Screen area, clearing, _clearscreen function, LIB 152–153
Screen buffer
  QuickWin windows, PT 161
Screen exchange
  CodeView options, ET 341, 343, 371, 445–447
  defined, ET 815
Screen Exchange command, CodeView, ET 424, 479
Screen Swap command
  CodeView, ET 368, 371
ScreenToClient member function
  CWnd class, XRF 802
Scroll bars
  CodeView
    options, ET 445–447
    toggling options, ET 370
  copying
    current minimum and maximum positions, CWnd::GetScrollRange, XRF 697
    position to specified locations, CScrollBar::GetScrollRange, XRF 556
  creating
    constructor, CScrollBar::CScrollBar, XRF 553
    initializing, CScrollBar::Create, XRF 553–555
  CScrollBar class described, XRF 551
  displaying, CWnd::ShowScrollBar, XRF 817–818
  hiding, CWnd::ShowScrollBar, XRF 817–818
  horizontal
    called when event occurs in clipboard viewer's, CWnd::OnHScrollClipboard, XRF 742–743
    called when user clicks, CWnd::OnHScroll, XRF 741–742
  in frame windows, XRF 17
  PWB
    screen display, ET 68
    window styles, ET 221, 275
  recalibrating
    Foundation classes tutorial, XUG 220, 226
  setting position range
    CScrollBar::SetScrollRange, XRF 556–557
    CWnd::SetScrollRange, XRF 812–813
  thumb, retrieving current postion, CScrollBar::GetScrollPos, XRF 555
  vertical
    called when clicked, CWnd::OnVScroll, XRF 796–797
    called with event in, CWnd::OnVScrollClipboard, XRF 797–798
Scroll boxes
  retrieving current position, CWnd::GetScrollPos, XRF 696–697
  setting to specified position, CWnd::SetScrollPos, XRF 811–812
ScrollDC member function
  CDC class, XRF 230
Scrolling
  CWnd, CWnd::ScrollWindow, XRF 803–804
  defined, ET 815
  horizontally, called when user clicks on bar, CWnd::OnHScroll, XRF 741–742
  keyboard commands for
    Foundation classes tutorial, XUG 233
  list boxes
    retrieving event, CListBox::GetHorizontalExtent, XRF 362
    setting width, CListBox::SetHorizonalExtent, XRF 370
  member functions, adding
    Foundation classes tutorial, XUG 227–229
  message handlers
    described, XUG 229
    Phone Book sample program, XUG 199
  message maps, using
    Foundation classes cookbook, XUG 322
  Mlines function, PWB, ET 183
  Plines function, ET 193
  switches, PWB, ET 284
  text, CEdit::LineScroll, XRF 296
  Vscroll switch, PWB, ET 306
  Window messages
    Foundation classes cookbook, XUG 322
  WM_HSCROLL message
    Foundation classes cookbook, XUG 322
  WM_VSCROLL message
    Foundation classes cookbook, XUG 322
_scrolltextwindow function, LIB 640–642
ScrollWindow member function
  CWnd class, XRF 803–804
Sdelete function, PWB, ET 153, 209–210
Search command
  CodeView, ET 423, 472–473
  QuickHelp, ET 769
Search logging, PWB, ET 178, 241–242
Search Memory command, CodeView, ET 443–444
Search menu
  CodeView, ET 361–362
  PWB
    described, ET 73
    predefined macros, ET 144

- Search paths
  - inference rules, NMAKE, ET 682
  - specifying, NMAKE, ET 660
- Search Results command, PWB
  - described, ET 77–78
  - predefined macros, ET 146
- Search Results dialog box, PWB, ET 89
- Search Results window
  - PWB
    - clearing, ET 162
    - described, ET 261–262
    - Mgrep function, ET 181–182
    - Nextsearch function, ET 189–190
- Searchall function, PWB, ET 153, 210
- Searchdialog switch, PWB, ET 264, 294
- _searchenv function, LIB 643–644
- Searchflush switch, PWB, ET 264, 295
- Searching
  - arrays
    - for keys, _lfind function, LIB 451–452
    - for values, _lsearch function, LIB 469–470
    - with binary search, bsearch function, LIB 127–128
  - backwards, PWB, ET 186
  - canceling
    - PWB, ET 161
    - pwbcascade macro, ET 230–231
  - dialog box controls, previous or next control, CWnd::GetNextDlgGroupItem, XRF 693–694
  - directories, CL option, ET 525–526
  - Find command, PWB, ET 87–90
  - for files using environment paths, _searchenv function, LIB 643–644
  - for first matching CObject pointer, COb List::Find, XRF 483
  - for specified window, CWnd::FindWindow, XRF 683
  - for strings
    - list box of combo box, XRF 148, 153
    - first character match, CString::FindOneOf, XRF 581
    - first substring match, CString::Find, XRF 580–581
  - global, Microsoft Advisor, ET 766–767
  - help system, in PWB, ET 199
- Searching (*continued*)
  - highlighting search strings, PWB, ET 210
  - in CodeView, overview, ET 361–362
  - INCLUDE environment variables, LR 201
  - library files, LINK, ET 570, 583–584
  - logging searches in PWB, ET 178, 241–242, 278
  - mark function, PWB, ET 86
  - memory, CodeView, ET 443–444
  - Mgrep function, PWB, ET 181–182
  - Mgreplist macro, PWB, ET 226
  - module-definition files, LINK, ET 571
  - object files, LINK, ET 565
  - overview, PWB, ET 85–86
  - procedure
    - Foundation classes tutorial, XUG 45–46
  - regular expressions
    - CodeView, ET 472–473
    - PWB, ET 90–93
  - sorting routines (list), LIB 54
  - symbol definitions
    - PWB, ET 98–99, 101–103
  - symbols
    - CodeView, ET 406–407, 421–422, 467–468
  - text
    - PWB, ET 197
  - window-manager's list for next or previous window, CWnd::GetNextWindow, XRF 695
- Searchwrap switch, PWB, ET 264, 295–296
- Seconds
  - getting, CTime::GetSecond, XRF 614
  - in current hour, getting, CTimeSpan::GetSeconds, XRF 623
- Section tags
  - TOOLS.INI file
    - PWB, ET 132
- Seek member function
  - CFile class, XRF 316–317; XUG 278
- seekg member function
  - istream class, XRF 883; XUG 390–391
- seekoff member function, XRF 929–930
- seekp member function, ofstream class
  - iostream classes tutorial, XUG 376
  - ostream class, XRF 904

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

seekpos member function
  streambuf class, XRF 930
SeekToBegin member function
  CFile class, XRF 317
SeekToEnd member function
  CFile class, XRF 317
/SEG option
  LINK, ET 591–592
__segment keyword
  base operators, LR 121
  based addressing, LR 121
  based functions, LR 173
  based pointers, LR 81
  described, LR 80
  restrictions, LR 80
__segment keyword, C++, LR+ 406, 412–414
segment pragma
  precompiled header compilation, effect on, PT 41
Segment registers, getting current values, _segread function, LIB 645–646
Segment tables
  EXEHDR output, ET 635–638
__segment type, C++, LR+ 50
Segmented architecture
  effects on addresses in arrays, LR 130
Segmented executable files
  defined, ET 815
  EXEHDR output, ET 634–636
  format, ET 631
  heap allocation, setting in EXEHDR, ET 630
  information, providing in EXEHDR, ET 631
Segmented Executable Linker
  LINK, ET 561
Segmented files
  adding to
    module-definition files, ET 614
  creating, LINK, ET 564
  packing code segments, LINK, ET 587–588
Segments
  code segments
    naming, custom memory models, PT 76–77
    packing, PT 26–27
    pointers, PT 56
    specifying, custom memory models, PT 77–78
  data segments
    naming, custom memory models, PT 76–77
    packing, PT 27
    stack segments, equality with, PT 71–74
  defined, ET 815
  defining attributes
    module-definition files, ET 618–620
Segments (*continued*)
  memory models, CL options, ET 488–490
  naming, CL option, ET 528–530
  ordering
    debugging considerations, ET 322
    LINK, ET 578, 585
  overlaid DOS programs, ET 598, 603–604
  packing code, LINK, ET 587–588
  setting number, LINK, ET 591–592
  relocating in memory, LR 59
  naming, custom memory models, PT 76–77
  references to, inline assembly, PT 114
  stack segments
    data segments, equality with, PT 71–74
/SEGMENTS option
  LINK, ET 591–592
SEGMENTS statement
  module-definition files, ET 609, 619–620
  overlaid DOS programs, ET 600–601
  overlay number, LINK, ET 564, 598
__segname function, C++, LR+ 414–415
__segname keyword, PT 79–80
  base operators, LR 121
  cast to segment values, LR 83
  described, LR 80, 173
  restrictions, LR 80
  storing data, LR 59
__segname keyword, C++, LR+ 406
_segread function, LIB 645–646
Selcur function, PWB, ET 153, 211
Select All command
  QuickWin, PT 149
Select function, PWB, ET 154, 211
Select To Anchor command, PWB
  described, ET 73
  predefined macros, ET 143
SelectClipRgn member function
  CDC class, XRF 231
Selected text command, CodeView, ET 361
SelectGdiObject member function, CDC class
  CBitmap::CreateBitmap, XRF 110
  CBitmap::CreateBitmapIndirect, XRF 110
  CBitmap::CreateDiscardableBitmap, XRF 112
  CFont::CreateFontIndirect, XRF 334
Selecting
  colors, graphics, PT 175–176
  consecutive items in list box, CListBox::SelItemRange, XRF 368
  floating-point libraries, PT 132

Selecting (*continued*)
  in PWB
    selection mode, ET 211–212, 241, 258, 278
    text, ET 211
    windows, ET 212
  memory models, PT 58, 62–63
  object into CMetaFileDC,
      CMetaFileDC::SelectObject, XRF 441
  predefined stock pens, brushes, fonts,
      CMetaFileDC::SelectStockObject, XRF 442
  video configuration, PT 173
Selection modes
  PWB
    changing, ET 211–212, 241, 258, 278
    setting, ET 73
Selection statements, C++, LR+ 138–142
SelectObject member function
  CDC class, XRF 232–233
    CBitmap::CreateBitmap, XRF 110
    CBitmap::CreateBitmapIndirect, XRF 110
    CBitmap::CreateDiscardableBitmap, XRF 112
    CFont::CreateFontIndirect, XRF 334
    Foundation classes cookbook, XUG 348
  CMetaFileDC class, XRF 441–442
    CBitmap::CreateBitmap, XRF 110
    CBitmap::CreateBitmapIndirect, XRF 110
    CBitmap::CreateDiscardableBitmap, XRF 112
    CFont::CreateFontIndirect, XRF 334
_selectpalette function, LIB 647–649; PT 188
SelectPalette member function
  CDC class, XRF 233–234
SelectStockObjec member function
  CDC class
    CDC::SelectStockObject, XRF 235
SelectStockObject member function
  CDC class, XRF 234–235
  CMetaFileDC class, XRF 442
SelectString member function
  CComboBox class, XRF 153
  CListBox class, XRF 367–368
__self function
  ensuring location, LR 174
__self keyword, PT 85
__self keyword, C++, LR+ 406
SelItemRange member function
  CListBox class, XRF 368
Selmode function, PWB, ET 154, 211–212
Selwindow function, PWB, ET 154, 212
Semaphores
  defined, ET 815
Semicolon (;)
  command separator, CodeView, ET 340, 352–353
  comments, PWB, ET 136
  LINK syntax, ET 565, 571
  null statement, LR 159–160
  prompt defaults
    LIB, ET 699
  statement terminator, LR 151
  terminating commands
    LIB, ET 698
  TOOLS.INI file syntax, ET 329–330
Semicolon (;;)
  TOOLS.INI file syntax, ET 652
SendDlgItemMessage member function
  CWnd class, XRF 804
Sending
  messages to windows, CWnd::SendMessage,
      XRF 805
  signal to executing programs, raise function,
      LIB 607–608
SendMessage member function
  CWnd class, XRF 805
Separator
  custom builds in PWB, ET 59
Sequence, NMAKE operations, ET 692–694
Sequence points
  after first operand in logical expressions, LR 136
  after logical-OR expression, LR 136
  described, LR 110–111
  evaluating expressions, LR 109
  expression evaluation, PT 289–290
  expressions, C++, LR+ 129–130
  guarantee order of evaluation, LR 113
  sequential operations, LR 140
  side effects, LR 183

Sequential-evaluation operator
  described, LR 140
  sequence points, LR 113
Serialization
  CArchive object, creating
    Foundation classes cookbook, XUG 284
  CDataBase member functions
    Foundation classes tutorial, XUG 136
  classes, of
    requirements, XUG 281–283
  classesDECLARE_SERIAL macro, XRF 40
  CObject class
    Foundation classes tutorial, XUG 29
  collection, XRF 23
  constructors
    defining, XUG 281
    tutorial, XUG 30
  CPerson object
    DMTEST sample program, XUG 33
    overview, XUG 34
    Foundation classes tutorial, XUG 25
  CPersonList object
    Foundation classes tutorial, XUG 25
  CString class
    Foundation classes tutorial, XUG 31
  DECLARE_SERIAL macro, XRF 39; XUG 24, 38, 44
  default behavior
    Foundation classes tutorial, XUG 43
  defined
    Foundation classes tutorial, XUG 20
  described, XRF 23
    Foundation classes cookbook, XUG 279
  exceptions
    constructing objects,
      CArchiveException::CArchiveException, XRF 105
    specifying cause, CArchiveException::m_cause,
      XRF 106
  Foundation classes tutorial, XUG 50
  IMPLEMENT_SERIAL macro
    Foundation classes tutorial, XUG 38, 44
  IsStoring member function
    Foundation classes tutorial, XUG 34
  objects, of
    Foundation classes cookbook, XUG 283–285
  Phone Book sample program, XUG 131–133
  procedure
    Foundation classes tutorial, XUG 42, 60
  Serialize member function
    Dump member function, differences, XUG 35
    Foundation classes cookbook, XUG 284
Serialization (*continued*)
  Serialize member function (*continued*)
    Foundation classes tutorial, XUG 24, 33
    overriding, XUG 282–283
  support, adding
    Foundation classes cookbook, XUG 265
  TRACE macro
    Foundation classes tutorial, XUG 50
  type-safety
    described, XUG 44
  testing objects for eligibility,
      CObject::IsSerializable, XRF 473
Serialize member function
  CObject class, XRF 473
Services
  CObject class, XRF 23–26
Sessions
  called to inform CWnd of end,
      CWmd::OnEndSession, XRF 736
  called when ending, CWnd::OnQueryEndSession,
      XRF 776
  defined, ET 806
  remote debugging, starting, ET 397–398
Set Anchor command, PWB
  described, ET 73
  predefined macros, ET 143
Set Breakpoint command, CodeView
  described, ET 364–366
  dialog box, CodeView, ET 365–366
  line numbers, ET 400
SET command
  environment variables in NMAKE, ET 678
Set Line-Display Mode option
  CodeView, ET 339
Set Mark File command, PWB, ET 73
Set Project Templates command, PWB, ET 75
Set Record command, PWB, ET 73
Set Runtime Arguments command, CodeView,
      ET 362–363
Set Screen-Exchange Method option
  CodeView, ET 341, 445–447
Set Screen Swapping option
  CodeView, ET 343
Set Switch function, PWB
  changing settings, ET 123
_set_new_handler, TUT 99–100
SetAbortProc member function
  CDC class, XRF 235–237
_setactivepage function, LIB 650–651
SetActiveWindow member function
  CWnd class, XRF 805–806

_setargv function, LR 31
_setargv function, C++, LR+ 39–41
SETARGV.OBJ file
  adding
    PWB, ET 50
  linking from within PWB, LR 31
SetAt member function
  CMapStringToOb class, XRF 384–385
  CObArray class, XRF 459–460
  CObList class, XRF 496–497
  CString class, XRF 590
SetAtGrow member function
  CObArray class, XRF 460–461
setb member function
  streambuf class, XRF 931
SetBitmapBits member function
  CBitmap class, XRF 116
SetBitmapDimension member function
  CBitmap class, XRF 116–117
_setbkcolor function, LIB 652–653; PT 189
_set_bnew_handler function, LIB 672–675
SetBkColor member function
  CDC class, XRF 238
SetBkMode member function
  CDC class, XRF 238–239
SetBrushOrg member function
  CDC class, XRF 239, 350
setbuf function, LIB 654–655
setbuf member function
  fstream class, XRF 843
  ifstream class, XRF 850
  ofstream class, XRF 899
  streambuf class, XRF 931–932
SetButtonStyle member function
  CButton class, XRF 133
SetCapture member function
  CWnd class, XRF 806
SetCaretPos member function
  CWnd class, XRF 806–807
SetCheck member function
  CButton class, XRF 134
SetClipboardViewer member function
  CWnd class, XRF 807
_setcliprgn function, LIB 656–657; PT 181, 187
_setcolor function, LIB 658–659; PT 189
SetColumnWidth member function
  CListBox class, XRF 369
SetCurSel member function
  CComboBox class, XRF 154
  CListBox class, XRF 369
SetDefID member function
  CDialog class, XRF 272
SetDepth member function
  CDumpContext class, XRF 277; XUG 36
SetDlgItemInt member function
  CWnd class, XRF 808
SetDlgItemText member function
  CWnd class, XRF 808
SetEditSel member function
  CComboBox class, XRF 154–155
_setenv function, C++
  suppressing library routine use, LR+ 41–42
setf member function
  ios class, XRF 863
Setfile function, PWB, ET 154, 212–213, 289–290
_setfillmask function, LIB 660–661; PT 191
_set_fnew_handler function, LIB 672–675
SetFocus member function
  CWnd class, XRF 809
_setfont function, LIB 662–664; PT 196–197
SetFont member function
  CWnd class, XRF 809
setg member function
  streambuf class, XRF 932
_setgtextvector function, LIB 665
SetHandle member function
  CEdit class, XRF 297–298
Sethelp function, PWB, ET 154, 213–214, 765–766
SetHorizontalExtent member function
  CListBox class, XRF 370
SetItemData member function
  CComboBox class, XRF 155
  CListBox class, XRF 370
setjmp function, ET 604; LIB 666
SetLength member function
  CFile class, XRF 318

_setlinestyle function, LIB 667; PT 192
_set_new_handler function, PT 106–107
setlocale function, LIB 668–669
SetMapMode member function
  CDC class, XRF 240–241
SetMapperFlags member function
  CDC class, XRF 241
SetMenu member function
  CMainWindow class
    adding, XUG 240
  CWnd class, XRF 810
SetMenuItemBitmaps member function
  CMenu class, XRF 435–436
_setmode function, LIB 670–671
_set_new_handler function, LIB 672–675
_set_nnew_handler function, LIB 672–675
setmode member function
  filebuf class, XRF 835
  fstream class, XRF 844
  ifstream class, XRF 851
  ofstream class, XRF 899
SetModify member function
  CEdit class, XRF 298
_set_new_handler function, C++, LR+ 321–323
setp member function
  streambuf class, XRF 932–933
SetPaletteEntries member function
  CPalette class, XRF 506–507
SetParent member function
  CWnd class, XRF 810
SetPasswordChar member function
  CEdit class, XRF 299
_setpixel function, LIB 676–677; PT 191
SetPixel member function
  CDC class, XRF 242
_setpixel_w function, LIB 676–677; PT 191
SetPolyFillMode member function
  CDC class, XRF 242–243
SetRect member function
  CEdit class, XRF 299–300
  CRect class, XRF 528–529
SetRectEmpty member function
  CRect class, XRF 529
SetRectNP member function
  CEdit class, XRF 300
SetRectRgn member function
  CRgn class, XRF 549–550
SetRedraw member function
  CWnd class, XRF 811
SetROP2 member function
  CDC class, XRF 243–244
SetScrollPos member function
  CScrollBar class, XRF 556–557
  CWnd class, XRF 811–812
SetScrollRange member function
  CScrollBar class, XRF 557
  CWnd class, XRF 812–813
SetSel member function
  CEdit class, XRF 301
  CListBox class, XRF 371
SetSize member function
  CObArray class, XRF 461
SetState member function
  CButton class, XRF 134
SetStatus member function
  CFile class, XRF 318–319
SetStretchBltMode member function
  CDC class, XRF 245
SetSysModalWindow member function
  CWnd class, XRF 813
SetTabStops member function
  CEdit class, XRF 301–302
  CListBox class, XRF 371–372
SetTextAlign member function
  CDC class, XRF 246–247
SetTextCharacterExtra member function
  CDC class, XRF 247
_settextcolor function, LIB 678–680
SetTextColor member function
  CDC class, XRF 248
_settextcursor function, LIB 681–682
SetTextJustification member function
  CDC class, XRF 248–249
_settextposition function, LIB 683–684
_settextrows function, LIB 685–686
_settextwindow function, LIB 687
SetTimer member function
  CWnd class, XRF 813–814
Setting
  active page, _setactivepage function, LIB 650–651
  active window, QuickWin, PT 161–162
  attributes of files, directories, _dos_setfileattr function, LIB 222–223
  background mode, CDC::SetBkMode, XRF 238
  binary/text mode
    filebuf objects, filebuf::setmode, XRF 835
    stream's filebuf object, fstream::setmode, XRF 844
    stream's filebuf object, ifstream::setmode, XRF 851
    streams, ios& binary, XRF 868
    streams, ofstream::setmode, XRF 899
  bitmap bits to values, CBitmap::SetBitmapBits, XRF 116

Setting (*continued*)
- bitmap-stretching mode, CDC::SetStretchBltMode, XRF 245
- buffers to specified character, memset and _fmemset functions, LIB 513–514
- characters of strings to character, _strset and _fstrset functions, LIB 782–783
- characters' range in edit control, CEdit::SetSel, XRF 301
- clipping region for graphics, _setcliprgn function, LIB 656–657
- colors
  - background, _setbkcolor function, LIB 652–653
  - background, current, CDC::SetBkColor, XRF 238
  - current, _setcolor function, LIB 658–659
  - text, _settextcolor function, LIB 678–680
  - text, CDC::SetTextColor, XRF 248
- cursor
  - attributes, _settextcursor function, LIB 681–682
  - toggle for graphics, _displaycursor function, LIB 179–180
- CWnd control caption or text, CWnd::SetDlgItemText, XRF 808
- date and time for files, _dos_setftime function, LIB 224–226
- default drive, _dos_setdrive function, LIB 220–221
- device contexts, x- and y-extents for associated windows, CDC::SetWindowExt, XRF 252
- drawing mode, CDC::SetROP2, XRF 243–244
- dump depth, CDumpContext::SetDepth, XRF 276–277
- error-bits, ios::clear, XRF 856
- far-pointer offsets and segments, _FP_OFF and _FP_SEG functions, LIB 293–294
- file default permission mask, _umask function, LIB 825–826
- file translation mode, _setmode function, LIB 670–671
- files' status, CFile::SetStatus, XRF 318
- fill masks, _setfillmask function, LIB 660–661
- floating point control word, _control87 function, LIB 160–162
- fonts, PT 196–197

Setting (*continued*)
- fonts, CWnd, CWnd::SetFont, XRF 809
- format flags in streams, setiosflags, XRF 870
- formatting rectangle of multiple-line edit control, CEdit::SetRect, XRF 299–300
- graphics modes, PT 168–171
- intercharacter spacing, CDC::SetTextCharacterExtra, XRF 247
- interrupt
  - signal handling, signal function, LIB 707–711
  - vector, _dos_setvect function, LIB 229–231
- line drawing logical mode, _setwritemode function, LIB 706
- line styles, _setlinestyle function, LIB 667
- locales, setlocale function, LIB 668–669
- mapping mode, CDC::SetMapMode, XRF 240–241
- menus, current to specified, CWnd::SetMenu, XRF 810
- palette values, _pg_setpalette function, LIB 574
- passwords, CEdit::SetPasswordChar, XRF 299
- pixel bitmaps for specified characters, _pg_setchardef function, LIB 573
- pixels at specified point, CDC::SetPixel, XRF 242
- pixels to current color, _setpixel functions, LIB 676–677
- polygon-filling mode, CDC::SetPolyFillMode, XRF 242
- screen rows for text, _settextrows function, LIB 685–686
- scroll bar position range
  - CScrollBar::SetScrollRange, XRF 557
  - CWnd::SetScrollRange, XRF 812–813
- scroll-bar thumb position, CScrollBar::SetScrollPos, XRF 556
- stream position indicators, fsetpos function, LIB 321–322
- streambuf object's buffering state, streambuf::unbuffered, XRF 937
- stream's
  - fill character, setfill, XRF 869
  - format conversion base to 10, ios& dec, XRF 868
  - format conversion base to 16, ios& hex, XRF 868
  - format conversion base to 8, ios& oct, XRF 869

Setting (*continued*)
  stream's (*continued*)
    internal field width parameter, setw, XRF 871
    internal field width variable, ios::width, XRF 865
    internal flags, ios::flags, XRF 858–859
    internal floating-point precision variable, setprecision, XRF 870
  styleset, _pg_setstyleset function, LIB 575
  system
    date, _dos_setdate function, LIB 218–219
    time, _dos_settime function, LIB 227–228
  system timer, CWnd::SetTimer, XRF 813–814
  text position, _settextposition function, LIB 683–684
  video mode, _setvideomode function, LIB 690–694
  video modes and rows in text modes, _setvideomoderows function, LIB 695–696
  visual pages, _setvisualpage function, LIB 701
  windows
    size, position, ordering, CWnd::SetWindowPos, XRF 814–816
Setting default arguments, TUT 11–12
SetTopIndex member function
  CListBox class, XRF 372
Setup
  Foundation classes tutorial, XUG 8
SETUP program
  See also Getting Started
  CodeView, installing, ET 327–329
  floating-point math library, PT 130–131
  help files, installing, ET 771
  memory model support, PT 59
_setvbuf function, LIB 688–689
_setvideomode function, LIB 690–694; PT 169–171, 187
_setvideomoderows function, LIB 695–696; PT 187
_setvieworg function, LIB 697–698; PT 181, 188
_setviewport function, LIB 699–700; PT 181–183, 188
SetViewportExt member function
  CDC class, XRF 250
SetViewportOrg member function
  CDC class, XRF 251
_setvisualpage function, LIB 701
_setwindow function, PT 184, 188
Setwindow function, PWB, ET 154, 214
SetWindowExt member function
  CDC class, XRF 252
SetWindowOrg member function
  CDC class, XRF 253
SetWindowPos member function
  CWnd class, XRF 814–816
SetWindowText member function
  CWnd class, XRF 297, 816
_setwritemode function, LIB 706
sgetc member function
  streambuf class, XRF 933
sgetn member function
  streambuf class, XRF 933
Shapes
  drawing functions (list), PT 189–191
SHARED keyword
  module-definition files, ET 622
Shell Escape command, CodeView, ET 423, 468–469
Shell function, PWB, ET 154, 214–215
Shells
  defined, ET 815
  DOS Shell command, ET 359
Shift operators, LR 130
Shift operators, bitwise, C++
  binary-operator expressions, LR+ 106–107
  overloading, LR+ 358
Shifting
  values, LR 131
short int type
  portability guidelines, PT 272
short int type, C++
  size, LR+ 51–52
short keyword, LR 51
short type
  byte ordering, PT 292–293
  conversion, LR 142
short type, C++, LR+ 50–52
Short-circuit evaluation, LR 114, 136
Shortcut keys
  CodeView, ET 346–347
  Foundation classes tutorial, XUG 93
  PWB, ET 79
Shortnames switch, PWB, ET 264, 296
ShowCaret member function
  CWnd class, XRF 816–817
ShowDropDown member function
  CComboBox class, XRF 155
Showing
  call tree, PWB, ET 99–101
  list box of combo box, CComboBox::ShowDropDown, XRF 155
ShowOwnedPopups member function
  CWnd class, XRF 817
ShowScrollBar member function
  CWnd class, XRF 817–818

- ShowWindow member function
  - CWnd class, XRF 818–819
    - CWinApp::m_nCmdShow, XRF 640
- SHRT_MAX constant, C++
  - integral limits, LR+ 62
- SHRT_MIN constant, C++
  - integral limits, LR+ 62
- SI register
  - CodeView syntax, ET 419, 450
  - variables, LR 48
- Side effects
  - caused by evaluation of expression, LR 110
  - completed, LR 155
  - in function calls, LR 110
  - inline functions, LR 192–193
  - macros, LR 192–193
  - order of evaluation, LR 110
  - sequence points, LR 183
  - unexpected results, LR 194
  - void expressions, LR 51
- SIGILL signal, LR 249
- Sign extension
  - portability guidelines, PT 285, 287
- signal function, LIB 707–711; LR 248
- Signaling executing programs, raise function, LIB 607–608
- Signals
  - defaults, LR 249
- Signed integers
  - conversions, LR 141
  - described, LR 99
  - results of bitwise operation, LR 135
- signed char type, C++
  - size, LR+ 51–52
- signed int type, C++
  - size, LR+ 51–52
- signed keyword
  - required with /J option, LR 52
  - with integral types, LR 51
- signed long type, C++
  - size, LR+ 51–52
- signed short type, C++
  - size, LR+ 51–52
- signed type, C++, conversion
  - from unsigned, LR+ 67–68
  - to unsigned, LR+ 67
- Significance
  - floating-point types, PT 127–129
- .SILENT dot directive
  - NMAKE, ET 687–688
- Simple assignment operator (=), LR 139
- Simple classes
  - Foundation classes cookbook, XUG 333
- Simple variable declarations, LR 54, 61–62
- sin function, LIB 712–713; PT 11
- Sines, calculating, sin functions, LIB 712–713
- Single inheritance, LR+ 259–264
- SINGLE keyword
  - module-definition files, ET 621
- Single precision, ET 815
- Single quotation mark (')
  - escape sequence, LR 18
  - forming character constants, LR 16
  - restrictions, LR 197
- sinh function, LIB 712–713; PT 11
- _sinhl function, LIB 712–713; PT 11
- _sinl function, LIB 712–713; PT 11
- Sinsert function, PWB, ET 154, 215–216
- Size
  - adding to CSize, CSize::operator+=, XRF 560
  - arrays, LR 94
    - establishing, CObArray::SetSize, XRF 461
    - returning, CObArray::GetSize, XRF 456
  - checking
    - equality between sizes, CSize::operator==, XRF 560
    - inequality between sizes, CSize::operator!=, XRF 560
  - creating CSize object, CSize::CSize, XRF 559
  - CSize class described, XRF 558
  - determining, LR 125
  - heaps, specifying, ET 617–618
  - pointers
    - code, custom memory model, PT 70–71
    - converting, PT 68–69
    - data, custom memory model, PT 71

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

- Size (*continued*)
  - pointers (*continued*)
    - defaults, PT 58
    - segments, PT 56
    - (table), PT 283
  - program
    - optimizing, PT 9–10
    - p-code use, effect on, PT 43
  - returning difference between two sizes, CSize::operator-, XRF 561
  - returning sum of two sizes, CSize::operator-, XRF 561
  - signed int, LR 99
  - stacks, specifying, ET 617
  - subtracting, CSize::operator-=, XRF 561
  - types, LR 98
  - types, C++, LR+ 51–52
  - unsigned int, LR 99
- Size command
  - CodeView, ET 373–374
  - PWB
    - described, ET 77
    - predefined macros, ET 145
- Size member function
  - CRect class, XRF 529
- SIZE operator
  - inline assembler use, PT 114
- sizeof operator
  - described, LR 125–126
  - in unary expressions, LR 111
  - in unsized arrays, LR 68
  - placement and association, LR 122
- sizeof operator, C++
  - unary-operator expressions, LR+ 95–96
- Sizing message handlers
  - adding
    - Foundation classes tutorial, XUG 224–226
  - Phone Book sample program, XUG 199
- skip pragma
  - precompiled header compilation, effect on, PT 41
- skip pragma, C++, LR+ 391
- /Sl option
  - CL, ET 541; LR 212
- Slash (/)
  - CL syntax, ET 488
  - command line, NMAKE, ET 647
  - HELPMAKE options, ET 711
  - LINK syntax, ET 575
  - Search command, CodeView, ET 361, 423, 472–473
- Slow motion
  - CodeView execution, ET 363, 369
- Small memory models
  - CL, PT 63
  - defined, ET 815
- Smart pointers, C++, LR+ 363
- SMARTDrive. *See Getting Started*
- SMARTDRV.SYS, ET 815
- /Sn option
  - HELPMAKE, ET 713
- Snapshots, memory, taking, CMemoryState::Checkpoint, XRF 54
- snextc member function
  - streambuf class, XRF 934
- Snow
  - suppressing, CodeView option, ET 341
- _snprintf function, LIB 725–726; LR 260
- Softcr switch, PWB, ET 264, 296–297
- _sopen function, LIB 714–716
- Sorting
  - frames, CL option, ET 539
  - qsort function, LIB 605–606
- Source 1 command, CodeView, ET 373–374
- Source 2 command, CodeView, ET 373–374
- Source browser
  - browser database, PWB, ET 731
    - building, ET 101–102, 106
    - case sensitivity, ET 309
    - combined, ET 106
    - creating, ET 97–98
    - estimating file size, ET 103–104
    - finding symbols, ET 103
    - makefiles, ET 61
    - non-PWB projects, ET 104–105
      - specifying, ET 310
  - CL options, ET 507–508
  - makefiles, PWB, ET 61
  - menu commands, PWB, ET 76
  - Pwbrowse functions, ET 200
  - searching, PWB, ET 86
  - switches, ET 309–310
- Source character set
  - compared to execution character set, LR 18
  - defined, LR 7
- Source code
  - displaying, CodeView, ET 350, 457–460
- .source command
  - HELPMAKE, ET 724
- Source files
  - C++
    - specifying, LR 261
  - consistency rules, precompiled headers, PT 41

- Source files (*continued*)
  - creating listing, CL, ET 501
  - decoding, HELPMAKE, ET 713–714
  - defined, ET 815
  - format options, CL, ET 541
  - HELPMAKE formats
    - minimally formatted ASCII, ET 728
    - QuickHelp, ET 716–724
    - rich-text format, ET 725–726
  - loading, CodeView, ET 359
  - opening, CodeView, ET 358
  - overview, LR 25–30
  - PWB project file list, ET 43
  - referencing variables at external level, LR 49
  - specifying type, HELPMAKE, ET 713
- Source files, suffix, TUT 60
- Source listing
  - form feeds, LR 213
  - number of lines per page, LR 213
  - skipping lines, LR 213
  - specifying, LR 212
  - subtitle, LR 213
  - title, LR 213
- Source mode defined, ET 815
- Source programs
  - *See also* Source files
  - defined, LR 26
- Source window
  - CodeView
    - arranging display, ET 327
    - displaying, ET 457–460
    - function, ET 350
    - getting help, ET 756
    - opening, ET 374
    - overview, ET 347–348
    - setting mode, ET 453–454
- Source1 Window command, CodeView, ET 368
- Source2 Window command, CodeView, ET 368
- /Sp option
  - CL, ET 541
- SP register
  - CodeView syntax, ET 419, 450
- Space
  - allocation
    - LINK, ET 577–578
  - inserting
    - PWB, ET 215–216
  - optimizing
    - PWB, ET 52–54
    - SBRPACK, ET 739–740
- Spaces
  - CodeView expression evaluators, ET 405
  - trailing, display mode, ET 301–302
- Spacing, intercharacter, retrieving setting, CDC::GetTextCharacterExtra, XRF 206
- SpanExcluding member function
  - CString class, XRF 591
- SpanIncluding member function
  - CString class, XRF 591
- _spawn functions, LIB 717–722; LR 34
- Special characters
  - C character set, LR 23
  - NMAKE
    - makefiles, ET 653
    - user-defined macros, ET 668
- Special macros, NMAKE, ET 671–672
- Special member functions. *See* Member functions, special
- Special-purpose words table, indexing
  - ios::iword, XRF 860
  - ios::pword, XRF 861
- Specifications, linkage. *See* Linkage Specifiers
  - CodeView Options command, ET 445–447
  - displaying source code, ET 457–460
  - memory format
    - dumping memory, ET 438–439
    - entering data, ET 440–441
    - viewing memory, ET 455–457
  - scope
    - searching for symbols, ET 467–468
- Specifiers, C++
  - access
    - base classes, LR+ 287–290
    - described, LR+ 286–287
  - described, LR+ 156

Specifiers, C++ (*continued*)
friend, LR+ 167
function, LR+ 159–163
memory-model, LR+ 401
storage-class, LR+ 157–158
type, LR+ 168–173
typedef, LR+ 163–167
Specify Interrupt Trapping option
CodeView, ET 341–342
Specifying
argument-type list, LR 51
calling conventions, LR 169–170
color, PWB display, ET 271–273
decorated names, ET 790–791
entry tables, CL option, ET 521
environment variables, CL, ET 557–559
execution model, CodeView, ET 333
expression evaluators, CodeView, ET 331
file type, HELPMAKE, ET 713
filename
HELPMAKE, ET 712, 714
LINK, ET 566–567
floating-point math package, CL options, ET 508
function addressing, LR 169
function return type, LR 51
identifiers, LR 5
fast compile, LR 260
inline files, NMAKE, ET 664
interrupt trapping, CodeView, ET 341–342
LIB fields, ET 699–705
libraries, LINK, ET 566–570
maximum number of entry tables, LR 260
module-definition files, LINK, ET 570
object files, LINK, ET 565
options, LINK, ET 575–576
page size, with LIB, ET 701–702
paths, CL, ET 496
pointer to unspecified type, LR 51
precompiled header filename, LR 260
preprocessor instructions, LR 260
real mode Windows, LR 260
response files, LINK, ET 573–575
search path, NMAKE, ET 660
source listing, LR 212
storage classes, LR 176
structure declarations, LR 68
symbol handlers
CodeView, ET 334–335
Speed
compile
increasing using precompiled headers, PT 33
p-code use, effect on, PT 43
execution
CodeView, ET 453
PWB switches, ET 280, 282
optimizing
LINK, ET 580–581
PWB, ET 54
pointer arithmetic, PT 57–58
program
optimizing, PT 9–10
_splitpath function, LIB 723–724
Splitting
help files, ET 773
floating point values into mantissa and exponent, modf and _modfl functions, LIB 523–524
sprintf function, LIB 725–726
sputbackc member function
streambuf class, XRF 934
sputc member function
streambuf class, XRF 934–935
sputn member function
streambuf class, XRF 935
sqrt function, LIB 727–728; PT 11
_sqrtl function, LIB 727–728; PT 11
Square brackets ([ ])
inline assembly, using in, PT 116
Square roots, calculating, sqrt and _sqrtl functions, LIB 727–728
srand function, LIB 729–730
_SRES16COLOR constant, PT 170
_SRES256COLOR constant, PT 170
/Ss option
CL, ET 541
SS register
CodeView syntax, ET 419, 450
SS register, equal to DS, PT 71–74
sscanf function, LIB 731–732
/ST option
LINK, ET 592
/St option
CL, ET 541
Stack allocation
setting, in EXEHDR, ET 631
Stack checking
disabling, PT 21
Stack collections, Foundation classes cookbook, XUG 275

- _STACK constant, C++, LR+ 415
- Stack frame defined, ET 815
- Stack machine
  - debugging p-code, ET 389
- /STACK option
  - EXEHDR, ET 631
  - LINK, ET 592
- Stack probes
  - defined, ET 518–520
- Stack segments
  - data segments, equality with, PT 71–74
- Stack trace
  - defined, ET 816
- Stack Trace command, CodeView, ET 422, 435–436
- _stackavail function, LIB 733
- Stacks
  - allocating memory on, _alloca function, LIB 84–85
  - checking on entry, LIB 11–12
  - defined, ET 815
  - getting available size, _stackavail function, LIB 733
  - numeric data processor
    - floating-point values, PT 130
  - p-code
    - local variables allocation order, PT 52
    - uses, PT 45–46
  - restoring environment, longjmp function, LIB 466–467
  - size, specifying, ET 494, 617
  - stack-probe routines
    - CL options, ET 518–520
- STACKSIZE statement
  - module-definition files, ET 609, 617
- Standard error defined, ET 816
- Standard input defined, ET 816
- Standard integral conversion, C++, LR+ 68
- Standard library defined, ET 816
- Standard memory models. *See* Memory models
- Standard mode defined, ET 816
- Standard output defined, ET 816
- Standard types
  - (list), LIB 67–69
  - using, LIB 61, 69
- StartDoc member function
  - CDC class, XRF 254
- Starting Microsoft C/C++. *See Getting Started*
- Starting PWB
  - command line, ET 65–66
  - Windows Program Manager, ET 66
- Starting point, setting random, srand function, LIB 729–730
- StartPage member function
  - CDC class, XRF 254
- Startup code
  - CL linking options, ET 528
  - defined, ET 816
- Startup code, C++
  - command-line arguments, parsing, LR+ 40–42
  - initialization considerations, LR+ 43–44
  - main function, LR+ 38–42
- Startup files
  - PWB configuration, ET 137
- STARTUP.CMD
  - PWB configuration, ET 137
- Startup, modifying CSTARTUP.BAT, LIB 42
- _stat function, LIB 734–735
- State file, CodeView
  - overview, ET 344
  - toggling status, ET 343
- State menu
  - QuickWin, PT 149
- State of an object, TUT 174
- Statefileread entry
  - TOOLS.INI file
    - CodeView, ET 330, 334, 343
- Statement body, LR 151
- Statement labels
  - *See also* goto statements
  - overview, LR 39
  - used in goto statements, LR 6
- Statements
  - described, LR 152–164
  - flow control, PWB, ET 112–114
  - module-definition files, ET 608–627
  - multiple, debugging, ET 322

- Statements (*continued*)
  - overlaid DOS programs, ET 600–601
  - overview, LR 151
  - repeating, LR 156
  - restrictions, LR 245
  - specifying file type, LINK, ET 564
  - TOOLS.INI syntax, PWB, ET 134–135
  - user input, PWB, ET 114–117
- Statements, C++
  - categories, LR+ 133
  - compound
    - described, LR+ 137
  - declaration
    - described, LR+ 134, 149–154
  - described, LR+ 134
  - #endif, Foundation classes tutorial, XUG 34
  - expression
    - described, LR+ 136–137
  - grammar summary, LR+ 433–434
  - #ifdef, Foundation classes tutorial, XUG 34
  - iteration
    - described, LR+ 142
  - jump
    - described, LR+ 147–149
  - labeled
    - described, LR+ 134–136
  - null
    - described, LR+ 136–137
  - selection
    - described, LR+ 138–142
  - syntax, LR+ 134
- Static binding, TUT 124
- Static control
  - creating
    - attaching, CStatic::Create, XRF 563–564
    - constructor, CStatic::CStatic, XRF 563
  - CStatic class described, XRF 562
- static keyword
  - applying to functions, LR 46
  - described, LR 48
  - external level, LR 45
  - in function declarations, LR 50
  - internal level, LR 36
  - lifetime, LR 37, 44
  - nonterminal, LR 43
  - overriding external linkage, LR 44
  - rules, LR 45–47
  - specifying, LR 35
  - visibility, LR 37
- static keyword, C++
  - declaration statements, LR+ 152–154
  - declarations, use in, LR+ 158
  - described, LR+ 46
  - linkage specification, LR+ 181
- Static library defined, ET 816
- Static linking, ET 697–698, 816
- Static members
  - data members, TUT 86–88
  - described, LR+ 243–244
  - initializing, LR+ 219
  - member functions, TUT 88
  - overview, TUT 85–86, 88–89
- Static objects
  - Foundation classes tutorial, XUG 111
  - initializing, LR+ 152–154, 329
- Static Overlay Manager, ET 604–605
  - LINK option, ET 586
  - overlaid DOS programs, ET 600
- Static storage class, C++
  - declaration statements, LR+ 152–154
  - specifiers, LR+ 158
- Static variables, C++
  - described, LR+ 46
  - initialization, LR+ 47–49
- Status
  - files
    - getting, CFile::GetStatus, XRF 310–311
    - setting, CFile::SetStatus, XRF 318
  - menu items, specifying, CMenu::GetMenuState, XRF 425–426
- Status Bar
  - defined, ET 816
  - overview, CodeView, ET 347
  - showing
    - CodeView option, ET 445–447
- Status Bar command
  - CodeView, ET 368, 370
  - QuickWin, PT 150
- Status files
  - PWB, ET 138–139
- Status information
  - getting on files, _stat function, LIB 734–735
  - returning graphics function call, _grstatus function, LIB 396–399
- _status87 function, LIB 736–737
- __STDC__ macro, LR 198
- __STDC macro, C++, LR+ 376

- __stdcall keyword
  - calling conventions, LR 55
    - 32-bit compilations, LR 170
    - unsupported for 16-bit targets, LR 266
  - described, LR 58
  - enabling
    - CL options, ET 550
  - specifying, LR 170
- __stdcall keyword, C++
  - calling convention, LR+ 419
- stdio member function, ios class
  - ios::sync_with_stdio, XRF 863
  - ostream::osfx, XRF 902
- stdiobuf class
  - described, XRF 915
  - member functions, XRF 916
- stdiobuf constructor, XRF 916
- stdiobuf destructor, XRF 916
- stdiobuf objects
  - creating, stdiobuf::stdiobuf, XRF 916
  - destroying, stdiobuf::~stdiobuf, XRF 916
  - returning C run-time file pointer,
- stdiobuf::stdiofile, XRF 916, 918
- stdiofile member function
  - stdiobuf class, XRF 916
- stdiostream class
  - described, XRF 917
  - member functions
    - rdbuf, XRF 918
    - stdiostream, XRF 918
    - ~stdiostream, XRF 918
- stdiostream constructor, XRF 918
- stdiostream destructor, XRF 918
- stdiostream objects
  - creating, stdiostream::stdiostream, XRF 918
  - destroying, stdiostream::~stdiostream, XRF 918
- Stock objects, retrieving handle to, CGdiObject::CreateStockObject, XRF 345–346
- Storage
  - arrays, LR 76
  - enumeration variables, LR 62
  - floating-point type requirements, PT 127–129
  - integers, LR 70, 98

Storage (*continued*)

  - portability guidelines, PT 274–276
  - register variables
    - __asm block effect on, PT 124–125
  - registers, LR 48
  - string literals, LR 21
  - structures, LR 70–71
  - types, LR 98–100
  - unions, LR 73
  - unsigned int, LR 99
- Storage classes, LR+ 25–26, 46–47
  - described, LR 43–50
  - determining meaning, LR 44
  - extern, LR 37, 44–47
  - functions, LR 168
  - in function declarations, LR 50
  - in parameters, LR 86
  - in variable declarations, LR 43
  - internal level, LR 47–50
  - nonterminal, LR 43
  - of identifiers, LR 34
  - of local variables, LR 181
  - register, LR 168
  - required on variable declarations, LR 61
  - restrictions, LR 43, 66
  - rules, LR 35
  - specifiers listed, LR 44
  - specifying, LR 176
  - static, LR 22, 35–36
- Storage-class specifiers, LR+ 157–158
  - restrictions, LR 44
- Storing
  - archives
    - object or primitive type, CArchive::operator, XRF 103
    - specified object to, CArchive::WriteObject, XRF 101
  - segments, LR 59
  - string pointers, LR 22
  - unions, LR 72
- Storing images in buffers, _getimage functions, LIB 367–369

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

- stossc member function
  - streambuf class, XRF 935
- str member function
  - istrstream class, XRF 892
  - ostrstream class, XRF 914
  - strstream class, XRF 940
  - strstreambuf class, XRF 944
- strcat function, LIB 738–739
- strchr function, LIB 740–742
- strcmp function, LIB 743–745; PT 11
- _strcmpi function, LIB 743–745, 759–760
- strcoll function, LIB 746
- strcpy function, LIB 747–748; PT 11
- strcspn function, LIB 749–750
- _strdate function, LIB 751–752
- _strdup functions, LIB 753–754
- Stream derivation sample program
  - iostream classes tutorial, XUG 400–405
- Stream I/O
  - buffering, LIB 36
  - controlling, setbuf function, LIB 654–655
  - error handling, LIB 13
  - error testing, LIB 38
  - predefined pointers, LIB 35–36
  - routines, LIB 33–35
  - transferring data, LIB 37–38
- Stream Mode command, PWB, ET 73, 143
- Stream objects, predefined
  - cerr, XRF 900
  - cin, XRF 875
  - clog, XRF 900
  - cout, XRF 900
- Stream pointers
  - defined, LIB 33
  - predefined, LIB 35–36
- Stream selection mode
  - setting, in PWB, ET 258
- streambuf class
  - consume defined, XRF 927
  - defining characteristics of derived class
    - streambuf::overflow, XRF 926
    - streambuf::sync, XRF 937
    - streambuf::underflow, XRF 938
  - described, XRF 919–921
  - get area
    - returning lower bound, streambuf::eback, XRF 924
    - returning number of character available for fetching, streambuf::in_avail, XRF 926
    - returning pointer to byte after last, streambuf::epptr, XRF 925
    - setting pointer values, streambuf::setg, XRF 932
- streambuf class (*continued*)
  - get pointers
    - following fetched characters, streambuf::sgetn, XRF 933
    - incrementing, streambuf::gbump, XRF 925
    - moving back, streambuf::sputbackc, XRF 934
    - moving forward one character, streambuf::snextc, XRF 934
    - moving forward one character, streambuf::stossc, XRF 935
    - returning character at, streambuf::sgetc, XRF 933
    - returning to next character to be fetched, streambuf::gptr, XRF 925
    - testing, streambuf::snextc, XRF 934
  - member functions
    - allocate, XRF 922
    - base, XRF 922
    - blen, XRF 922
    - dbp, XRF 923
    - doallocate, XRF 923
    - eback, XRF 924
    - ebuf, XRF 924
    - egptr, XRF 924
    - epptr, XRF 925
    - gbump, XRF 925
    - gptr, XRF 925
    - in_avail, XRF 926
    - out_waiting, XRF 926
    - overflow, XRF 926–927
    - pbackfail, XRF 927
    - pbase, XRF 928
    - pbump, XRF 928
    - pptr, XRF 928
    - sbumpc, XRF 929
    - seekoff, XRF 929–930
    - seekpos, XRF 930
    - setb, XRF 931
    - setbuf, XRF 931–932
    - setg, XRF 932
    - setp, XRF 932–933
    - sgetc, XRF 933
    - sgetn, XRF 933
    - snextc, XRF 934
    - sputbackc, XRF 934
    - sputc, XRF 934–935
    - sputn, XRF 935
    - stossc, XRF 935
    - streambuf, XRF 936
    - ~streambuf, XRF 936
    - sync, XRF 884, 907, 937

- streambuf class (*continued*)
  - member functions (*continued*)
    - unbuffered, XRF 937
    - underflow, XRF 938
  - output streams, deriving
    - iostream classes tutorial, XUG 399–405
  - put area
    - returning first byte of, streambuf::pptr, XRF 928
    - returning pointer to start of, streambuf::pbase, XRF 928
    - setting pointer values, streambuf::setp, XRF 932
    - storing character, streambuf::sputc, XRF 934
  - put pointer
    - following stored characters, streambuf::sputn, XRF 935
    - incrementing, streambuf::pbump, XRF 928
  - repositioning external file pointer, streambuf::pbackfail, XRF 927
    - attaching to object, streambuf::setbuf, XRF 931
  - reserve area
    - returning pointer to byte after last, streambuf::ebuf, XRF 924
    - returning pointer to, streambuf::base, XRF 922
    - returning size in bytes, streambuf::blen, XRF 922
    - setting position values, streambuf::setb, XRF 931
    - setting up, streambuf::allocate, XRF 922
  - returning current character and advancing get pointer, streambuf::sbumpc, XRF 929
  - returning number of characters available for fetching, streambuf::out_waiting, XRF 926
  - returning pointer to byte after last, streambuf::egptr, XRF 924
  - virtual overflow function, streambuf::overflow, XRF 926–927
  - virtual sync function, streambuf::sync, XRF 937
  - virtual underflow function, streambuf::underflow, XRF 938
  - writing ASCII debugging information on stdout, streambuf::dbp, XRF 923
- streambuf constructor, XRF 936
- streambuf destructor, XRF 936
- streambuf objects
  - associated with stream, returning pointer to, ios::rdbuf, XRF 862
  - changing position for, streambuf::seekoff, XRF 929–930
  - changing position relative to stream beginning, streambuf::seekpos, XRF 930
  - creating, streambuf::streambuf, XRF 936
  - reserve area, allocating when needed, streambuf::doallocate, XRF 923
  - setting buffering state, streambuf::unbuffered, XRF 937
  - virtual destructor, streambuf::~streambuf, XRF 936
- Streams
  - *See also* iostreams
  - assigning istream object to istream_withassign object, istream_withassign::operator =, XRF 889
  - associating with files, _fdopen function, LIB 264–266
  - attaching
    - to already open file, ofstream::attach, XRF 894
    - to specified open file, ifstream::attach, XRF 846
  - buffer control
    - setbuf function, LIB 654–655
    - setvbuf function, LIB 688–689
  - buffer-deletion flag, assigning value to, ios::delbuf, XRF 856
  - buffers
    - flushing, ostream::flush, XRF 902
    - returning number of bytes stored in, ostrstream::pcount, XRF 913
    - returning pointer to strstreambuf buffer object, XRF 913
  - C++, synchronizing with standard C stdio streams, ios::sync_with_stdio, XRF 863
  - changing, position value, ostream::seekp, XRF 904
  - characters
    - inserting into output, ostream::put, XRF 903
    - returning next without extracting, istream::peek, XRF 882
    - returning number extracted by last unformatted input function, istream::gcount, XRF 877

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

Streams (*continued*)
- characters (*continued*)
  - synchronizing internal buffer with external character source, istream::sync, XRF 884
- clearing format flags, ios::unsetf, XRF 864
- closing
  - fclose and _fcloseall functions, LIB 260–261
  - functions, LIB 37
- description
  - iostream classes tutorial, XUG 363
- determining if error bits are set, ios::operator !(), XRF 866
- diagnostic output, human-readable text, CDumpContext class described, XRF 273
- end-of-file testing, feof function, LIB 267–268
- errors
  - determining if error bits are clear, ios::good, XRF 859
  - determining if error bits are set, ios::operator !(), XRF 866
  - returning current specified error state, ios::rdstate, XRF 862
- extracting
  - characters and discarding, istream::ignore, XRF 880
  - data, istream::get, XRF 878, 880
  - white space, istream& ws, XRF 886
  - white space, istream::eatwhite, XRF 877
- extraction operations
  - called after, istream::isfx, XRF 881
  - called prior to, istream::ipfx, XRF 880
  - operators, istream::operator>>, XRF 885
  - specified number of bytes, istream::read, XRF 883
- file descriptor, returning, ofstream::fd, XRF 895
- filebuf class described, XRF 831
- filebuf objects
  - attaching specified reserve area, fstream::setbuf, XRF 843
  - attaching specified reserve area, ifstream::setbuf, XRF 850
  - attaching specified reserve area, ofstream::setbuf, XRF 899
  - closing, ofstream::close, XRF 894
  - opening file and attaching, fstream::open, XRF 842
  - opening file for attachment, ofstream::open, XRF 898
  - returning pointer to associated, ofstream::rdbuf, XRF 898
  - returning pointer to, ifstream::rdbuf, XRF 850

Streams (*continued*)
- filebuf objects (*continued*)
  - setting binary/text mode, ofstream::setmode, XRF 899
- flushing
  - fflush function, LIB 271–272
  - _flushall function, LIB 287
- flushing output buffer, ostream& flush, XRF 907
- fstream class described, XRF 836
- get pointers
  - changing, istream::seekg, XRF 883
  - getting value, istream::tellg, XRF 884
- getting
  - file handles, _fileno function, LIB 282
  - file-position indicator, fgetpos function, LIB 275–276
  - integers, _getw function, LIB 389–390
  - lines from, gets function, LIB 376–377
  - strings from, fgets function, LIB 277–278
- getting position value, ostream::tellp, XRF 904
- ifstream class described, XRF 845
- increasing maximum number, LIB 40–42
- input, putting character back into, istream::putback, XRF 882
- insert operations
  - called after, ostream::osfx, XRF 902
  - called before, ostream::opfx, XRF 902
- inserting
  - arguments into, ostream::operator<<, XRF 906
  - bytes, ostream::write, XRF 905
  - newline character and flushing buffer, ostream& endl, XRF 907
  - null-terminator character, ostream& ends, XRF 907
- internal flags variable, setting, ios::flags, XRF 858–859
- ios class described, XRF 852
- Iostream_init class described, XRF 874
- iostream class described, XRF 872
- istream class described, XRF 875
- istream objects, XRF 881–882
- istream_withassign class described, XRF 887
- istrstream class described, XRF 890
- masks
  - current radix flag bits, ios::basefield, XRF 867
  - floating-point format flag bits, ios::floatfield, XRF 867
- object state variables, providing without class derivation, ios::xalloc, XRF 865
- ofstream class described, XRF 893

- Streams (*continued*)
  - opening
    - functions, LIB 35
    - with file sharing, _fsopen function, LIB 323–325
  - opening file and attaching to filebuf object, ifstream::open, XRF 849
  - ostream class described, XRF 900
  - ostream_withassign class described, XRF 908
  - ostrstream class described, XRF 911
  - padding flag bits, obtaining, ios::adjustfield, XRF 867
  - pointers. *See* Stream pointers
  - printing
    - data to, fprintf function, LIB 299–300
    - formatted output to, printf function, LIB 585–592
  - pushing characters back onto, ungetc function, LIB 827–828
  - reading characters from
    - fgetc and _fgetchar functions, LIB 273–274
    - getc and getchar functions, LIB 346–347
  - reading data from
    - fread function, LIB 304–305
    - fscanf function, LIB 316–317
  - resetting error indicator, clearerr function, LIB 150–151
  - returning associated file descriptor
    - fstream::fd, XRF 839
    - ifstream::fd, XRF 847
  - returning pointer to associated filebuf buffer object, fstream::rdbuf, XRF 843
  - setting
    - binary/text mode, ifstream::setmode, XRF 851
    - fill character, setfill, XRF 869
    - floating-point precision variable, ios::precision, XRF 861
    - format conversion base to 8, ios& oct, XRF 869
    - format conversion base to 10, ios& dec, XRF 868
    - format conversion base to 16, ios& hex, XRF 868
    - internal field width parameter, setw, XRF 871
    - internal field width variable, ios::width, XRF 865
    - internal fill character variable, ios::fill, XRF 857
    - internal floating-point precision variable, setprecision, XRF 870
- Streams (*continued*)
  - setting (*continued*)
    - mode to text, ios& text, XRF 871
    - position indicator, fsetpos function, LIB 321–322
    - specified format bits, ios::setf, XRF 863
    - text to binary mode, ios& binary, XRF 868
  - special-purpose words table, indexing
    - ios::iword, XRF 860
    - ios::pword, XRF 861
  - stdiobuf class described, XRF 915
  - stdiostr class described, XRF 917
  - streambuf class described, XRF 919
  - streambuf objects, returning pointer to, ios::rdbuf, XRF 862
  - strstream class described, XRF 939
  - strstreambuf buffer object, returning pointer to, istrstream::rdbuf, XRF 892
  - strstreambuf class described, XRF 943
  - synchronizing internal buffer with external character source, istream::sync, XRF 884
  - testing
    - end-of-file, ios::eof, XRF 857
    - for attachment to open file, ifstream::is_open, XRF 849
    - for attachment to specified open disk file, fstream::is_open, XRF 842
    - for attachment to specified open file, ofstream::is_open, XRF 895
    - for errors, ferror function, LIB 269–270
    - for serious I/O errors, ios::bad, XRF 855
  - tying to ostream, ios::tie, XRF 864
  - virtual overflow function, streambuf::overflow, XRF 926–927
  - writing
    - characters to, fputc and _fputchar functions, LIB 301–302
    - characters to, putc and putchar functions, LIB 593–594
    - data from, fwrite function, LIB 338–339
    - integers to, _putw function, LIB 603–604
    - strings to, fputs function, LIB 303
- strerror function, LIB 755–756; LR 254
- _strerror function, LIB 755–756

- StretchBlt member function, CDC class, XRF 255–257
- strftime function, LIB 757–758
- _stricmp function, LIB 743–745, 759–760
- String concatenation
  - multiple strings, LR 19
  - translation phase, LR 191
- String functions
  - standard C library, Foundation classes cookbook, XUG 260
- String literals
  - CodeView expression evaluators, ET 408
  - defined, LR 107
  - described, LR 20–22
  - escape sequences, LR 196
  - sending to standard output, LR 212
  - types, LR 21
- String literals, C++, LR+ 20–23
- String manipulation routines, LIB 54–55
- String pointers
  - storing, LR 22
- Stringizing operator (#), LR 195–196
- Stringizing operator, C++
  - described, LR+ 371–372
- Strings
  - adding
    - list boxes, CListBox::AddString, XRF 355
    - to list boxes, CListBox::InsertString, XRF 366
  - appending
    - characters of, strncat and _fstrncat functions, LIB 765–766
    - strcat and _fstrcat functions, LIB 738–739
  - basic operations
    - Foundation classes cookbook, XUG 257–258
  - character
    - to list box of combo box, CComboBox::AddString, XRF 142
    - retrieving width, height, CDC::GetTabbedTextExtent, XRF 204–205
  - CodeView expression evaluators, ET 408
  - comparing
    - characters from two, strncmp and _fstrncmp functions, LIB 767–769
    - characters of two strings, strnicmp and _fstrnicmp functions, LIB 772–773
    - lowercase, _stricmp and _fstricmp functions, LIB 759–760
    - strcmp and _fstrcmp functions, LIB 743–745
    - strcoll function, LIB 746

- Strings (*continued*)
  - comparing two
    - CString::Collate, XRF 576
    - CString::Compare, XRF 577
    - CString::CompareNoCase, XRF 577–578
  - converting
    - characters from ANSI to OEM character set, CString::AnsiToOem, XRF 576
    - characters from OEM to ANSI character set, CString::OemToAnsi, XRF 588
    - CString object to lowercase, CString::MakeLower, XRF 586
    - CString object to uppercase, CString::MakeUpper, XRF 587
  - converting to values
    - double, atof function, LIB 98–100
    - integer, _atold function, LIB 98–100
    - long double, atoi function, LIB 98–100
    - long, atol function, LIB 98–100
  - copying, strcpy and _fstrcpy functions, LIB 747–748
  - corresponding to CTime object
    - converted, CTime::Format, XRF 609
    - unconverted, CTime::FormatGmt, XRF 610
  - corresponding to CTimeSpan, generating, CTimeSpan::Format, XRF 621
  - CString class described, XRF 572
  - debugging assembly language, ET 415
  - defined, ET 816
  - deleting
    - from list boxes, CListBox::DeleteString, XRF 359
    - from list box in combo box, CComboBox::DeleteString, XRF 147
  - duplicating, _strdup functions, LIB 753–754
  - embedding, CL option, ET 544
  - extracting first characters from CString object and returning copy, CString::Left, XRF 585
  - extracting from CString object the largest substring
    - excluding specified characters CString::SpanExcluding, XRF 591
  - extracting last characters from CString object and returning copy, CString::Right, XRF 589–590
  - extracting substring of specified length and returning copy, CString::Mid, XRF 587
  - finding
    - characters in, strchr and _fstrchr functions, LIB 740–742
    - next token in, strtok and _fstrtok functions, LIB 794–796
    - substrings first, strspn and _fstrspn functions, LIB 784–785

Strings (*continued*)
finding (*continued*)
substrings in, strcspn and _fstrcspn functions, LIB 749–750
substrings, strstr and _fstrstr functions, LIB 786–787
finding in list boxes, CListBox::FindString, XRF 361
getting
character strings from console, _cgets function, LIB 136–137
from list box of combo box, CComboBox::GetLBText, XRF 150
from streams, fgets function, LIB 277–278
length, strlen and _fstrlen functions, LIB 761–762
initializing, LR 97–98
inserting
list box of combo box, CComboBox::InsertString, XRF 151
justifying, CDC::SetTextJustification, XRF 248–249
list boxes
getting length, CListBox::GetTextLen, XRF 366
getting, CListBox::GetText, XRF 365
scrolling selected, CListBox::SelCurSel, XRF 369
searching for matching, CListBox::SelectString, XRF 367
selecting, CListBox::SelSel, XRF 371
lists of objects, CStringList class described, XRF 603
making CString object an empty string, CString::Empty, XRF 580
manipulation of
Foundation classes cookbook, XUG 256–261
menu items, copying, CMenu::GetMenuString, XRF 427
mixed-language programming, PT 259–262
null-terminated
converting to C style, XUG 259
objects arrays, CStringArray class described, XRF 601
overwriting specified character, CString::SetAt, XRF 590
putting to console, _cputs function, LIB 167
reading formatted data from, sscanf function, LIB 731–732

Strings (*continued*)
reading specified Windows string resource, CString::LoadString, XRF 586
returning
character specified by index, CString::GetAt, XRF 581
count of characters in CString object, CString::GetLength, XRF 584
pointer to internal character buffer and matching length, CString::GetBufferSetLength, XRF 583–584
pointer to internal character buffer for CString object, CString::GetBuffer, XRF 582–583
reversing character order in CString object, CString::MakeReverse, XRF 586
searching
CString object for last substring match, CString::ReverseFind, XRF 589
for in list box of combo box, XRF 148, 153
first character match, CString::FindOneOf, XRF 581
first substring match, CString::Find, XRF 580–581
setting to specified integer value, CWnd::SetDlgItemInt, XRF 808
static storage duration, LR 22
streams, returning pointer to character array, istrstream::str, XRF 892
terminating use of buffer, CString::ReleaseBuffer, XRF 588
testing CString object for empty condition, CString::IsEmpty, XRF 584
time, formatting, strftime function, LIB 757–758
transforming based on locale-specific information, strxfrm function, LIB 799–800
user-defined macros, NMAKE, ET 668
writing
at specified location, CDC::TextOut, XRF 259
double numbers to, _ecvt function, LIB 258–259
formatted data to, sprintf function, LIB 725–726
long integers to, _ltoa function, LIB 474–475
tabbed text, CDC::TabbedTextOut, XRF 258–259
to lowercase, _strlwr and _fstrlwr functions, LIB 763–764

Strings (*continued*)
  writing (*continued*)
    to output, puts function, LIB 602
    to regions, CDC::ExtTextOut, XRF 188–189
    to streams, fputs function, LIB 303
strlen function, LIB 761–762
    to uppercase, _strupr and _fstrupr functions, LIB 797–798; PT 11
_strlwr function, LIB 763–764
strncat function, LIB 765–766
strncmp function, LIB 767–769
strncpy function, LIB 770–771
_strnicmp function, LIB 772–773
_strnset function, LIB 774–775
strpbrk function, LIB 776–777
strrchr function, LIB 778–779
_strset function, LIB 782–783
  intrinsic form, PT 11
strspn function, LIB 784–785
strstr function, LIB 786–787
strstream buffer objects, returning pointer to,
istrstream::rdbuf, XRF 892
strstream class
  buffer, returning nunber of bytes in,
strstream::pcount, XRF 940
  described, XRF 939
  iostream classes tutorial, XUG 392–393
  member functions
    pcount, XRF 940
    rdbuf, XRF 940
    str, XRF 940
    strstream, XRF 941
    ~strstream, XRF 942
  returning
    number of bytes in buffer, strstream::pcount, XRF 940
    pointer to internal character array, strstream::str, XRF 940
    pointer to strstreambuf object, strstream::rdbuf, XRF 940
strstream constructor, XRF 941
strstream destructor, XRF 942
strstream objects
  creating, strstream::strstream, XRF 941
  destroying, strstream::~strstream, XRF 942
  returning pointer to, strstream::rdbuf, XRF 940
strstreambuf class
  described, XRF 943
  member functions
    freeze, XRF 913, 944
    str, XRF 944
strstreambuf class (*continued*)
  member functions (*continued*)
    strstreambuf, XRF 945–946
    ~strstreambuf, XRF 946
  preventing automatic memory deletion, strstreambuf::freeze, XRF 944
  returning pointer to internal character array, strstreambuf::str, XRF 944
strstreambuf constructor, XRF 945–946
strstreambuf objects
  creating, strstreambuf::strstreambuf, XRF 945
  destroying, strstreambuf::~strstreambuf, XRF 946
  returning pointer
    from associated stream, ostrstream::rbuf, XRF 913
    to internal character array, strstreambuf::str, XRF 944
_strtime function, LIB 788–789
strtod function, LIB 790–793
strtok function, LIB 794–796
strtol function, LIB 790–793
_strtold function, LIB 790–793
strtoul function, LIB 790–793
struct keyword, C++
  class type declaration, LR+ 228
struct type names, C++
  introduction by declaration statements, LR+ 149
Structure declarations, LR 65–70
Structure members
  defined, ET 816
  described, LR 119–120
Structure tags
  declaring, LR 263
  overview, LR 39
Structure types
  incomplete, LR 100
  inline assembly, PT 117–118
  presentation graphics
    _axistype, PT 221–223
    _chartenv, PT 219–220, 226–227
    described, PT 219–220
    _legendtype, PT 225
    _titletype, PT 220–221
    _windowtype, PT 223–224
Structures
  aggregate types, LR 93–97
  alignment, LR 70–71
  anonymous, LR 67
  debugging assembly language, ET 415
  defined, ET 816; LR 54
  embedded, LR 67

Structures (*continued*)
expanding and contracting, CodeView, ET 367–368, 478–479
inline assembly limitations, PT 113
mixed-language programming, PT 265
packing, LR 212
portability guidelines
bit fields, PT 279–280
order and alignment, PT 275–276
reading and writing, PT 278–279
CL options, ET 554–555
Structures, C++
as user-defined types, TUT 39–41
COMPAREITEMSTRUCT, XRF 77–78
CREATESTRUCT, XRF 78–79
declaring, LR+ 228
DELETEITEMSTRUCT, XRF 80
derivative types, LR+ 58
DRAWITEMSTRUCT, XRF 81–83
MEASUREITEMSTRUCT, XRF 83–84
PAINTSTRUCT, XRF 84–85
POINT, XRF 85
RECT, XRF 86
storage allocation
_strupr function, LIB 797–798
strxfrm function, LIB 799–800
Stub file, defined, ET 816
STUB statement
module-definition files, ET 609, 614
Styles
button
getting, CButton::GetButtonStyle, XRF 131
changing, CButton::SetButtonStyle, XRF 133
windows, retrieving, CWnd::GetStyle, XRF 698
presentation graphics
described, PT 204–205
style pool, PT 216
Styleset
getting current array, _pg_getstyleset function, LIB 568
resetting to default, _pg_resetstyleset function, LIB 572
setting current, _pg_setstyleset function, LIB 575
Subdirectories
copying files to
PWB, ET 95
Subexpressions
elimination, PT 8, 23
optimizing
CL option, ET 533
Subroutine
defined, ET 816
Subscript expressions
applying to pointers, LR 117
evaluating, LR 117–118
l-values, LR 108
referring to array values, LR 116
with multiple subscripts, LR 117
Subscript operator
CObArray::operator [], XRF 462
CString class, XRF 597
overloading, LR+ 362–363
postfix expressions, LR+ 81–83
subtitle pragma
precompiled header compilation, effect on, PT 41
subtitle pragma, C++, LR+ 391
Subtracting
rectangles, CRect::operator –, XRF 533–534
sizes, CSize::operator-=, XRF 561
time spans
CTimeSpan::operator+,-, XRF 625
CTimeSpan::operator+=,-=, XRF 626
Subtraction assignment operator (–=)
table listing, LR 138
Subtraction operator (–)
binary-operator expressions, LR+ 104–106
CRect class, XRF 533–534
described, LR 129
overloading, LR+ 359
.SUFFIXES dot directive
NMAKE, ET 688
clearing, ET 650
inference rules, ET 680–682, 685
Support, product. *See* the Product Assistance Request Form in LIB
Supporting files, XUG 242

Suppress Snow option
  CodeView, ET 341
_swab function, LIB 801–802
Swapping
  bytes, _swab function, LIB 801–802
  defined, ET 816
  screen exchange
    CodeView, ET 341, 343, 371, 445–447
switch statements
  See also if statements
  described, LR 161–164
  terminating, LR 152
switch statements, C++
  drawbacks, TUT 109
  labels, use restrictions, LR+ 135–136
  replacing with polymorphism, TUT 177
  selection statements, LR+ 139–142
Switches
  PWB, ET 263–308
    Boolean switch syntax, ET 266
    changing, ET 122, 124
    Filename-Parts syntax, ET 265–266
    help, ET 313–315
    library, ET 310–312
    source browser, ET 309–310
    tabs, ET 127–129
    unixre, ET 91
  syntax
    TOOLS.INI file, PWB, ET 135
Switching
  Window function, PWB, ET 220
Symbol handler
  specifying
    CodeView, ET 334–335
Symbolhandler entry
  TOOLS.INI file
    CodeView, ET 328, 330, 335
    remote debugging, ET 393–395
Symbolic constants
  described, LR 192
  graphics, PT 180
  inline assembly, using in, PT 115–116
Symbolic debugger
  optimizations, removing, PT 9
Symbolic Debugging Information
  compressing, ET 324–325
  defined, ET 323, 806
  LINK, ET 577
Symbolic Debugging Information (*continued*)
  loading, ET 342
  memory requirements, ET 324
  preserving, with CVPACK, ET 745
  searching, ET 362
Symbols
  C++, name spaces, LR+ 61
  case sensitivity, LR 7, 235
  defined, ET 805–806, 811
  described, LR 5
  exporting, LR 58
  exporting from DLLs, LR 172
  format, CodeView, ET 408–409
  inline assembly, using in, PT 115–117
  local, building a database, ET 732, 736
  PWB, ET 98–99, 101–103
  .SBR files, PWB, ET 104
  searching for, CodeView, ET 421–422, 467–468
  unreferenced, packing files, ET 739
sync member function
  istream class, XRF 884
  streambuf class, XRF 937
  used by ostream flush operator, XRF 907
Synchronizing
  C++ streams with standard C stdio streams,
    ios::sync_with_stdio, XRF 863
sync_with_stdio member function
  ios class, XRF 863
Syntax
  See also specific command or utility
  summary, C++, LR+ 423–436
.SYS files
  defined, ET 817
sys_errlist variable, LIB 63–64
sys_nerr variable, LIB 63–64
__syscall keyword
  calling conventions, LR 55, 266
  enabling
    CL options, ET 550
System call routines, LIB 55
System date, getting, _dos_getdate function,
  LIB 196–197
SYSTEM environment variable
  defined, ET 817
System files system function, LIB 803–804; LR 254
System include files
  finding symbols, PWB, ET 101
System requirements. *See Getting Started*

System time
  called after change, CWnd::OnTimeChange, XRF 794
  getting
    _dos_gettime function, LIB 207–208
    time function, LIB 812–814
SYSTEM.INI. *See Getting Started*

T

T command, CodeView, ET 423, 433, 452–453
/T option
  HELPMAKE, ET 713–714, 722
  LINK, ET 592–593
  NMAKE, ET 650
  PWB, ET 142
t option, optimize pragma, PT 10
/Ta option
  CL, ET 541–542
Tab escape sequences (\t), LR 18
\tab formatting code
  HELPMAKE, ET 727
Tab function, PWB, ET 127–128, 154, 216
Tab Set command, CodeView, ET 423, 470
Tab stops
  setting in edit control, CEdit::SetTabStops, XRF 301
  setting in list boxes, CListBox::SetTabStops, XRF 371
Tabalign switch, PWB, ET 127–129, 264, 297
TabbedTextOut member function
  CDC class, XRF 258–259
Tabdisp switch, PWB, ET 127–128, 264, 297–298
    ASCII code, ET 297
Tables
  accelerator
    Foundation classes tutorial, XUG 109
  EXEHDR output, ET 635–636, 638
  specifying number, LR 260
Tabs
  HELPMAKE syntax, ET 711
  hyperlinks, navigating with, ET 759–761
  module statement syntax, ET 610

Tabs (*continued*)
  PWB
    aligning switches, ET 297
    handling, ET 127–129
    line continuation, ET 136
    preserving, ET 292
    previous, ET 159
    setting, ET 298–299
    width, ET 282
  regular expressions, PWB, ET 93
  setting
    CodeView, ET 470
Tabstops switch, PWB, ET 127, 264, 298–299
  changing, ET 122
Tagged expressions
  Build:message switch, ET 784
  justifying, ET 785
  overview, ET 782–784
  regular expression syntax, ET 778–781, 787
  replacing text
    PWB, ET 93–96
Tags
  enumeration, LR 62
  overview, LR 39
  structure declarations, LR 65
  TOOLS.INI file, ET 329–330
    PWB, ET 132–134
tan function, LIB 805–806; PT 11
Tangents, calculating, tan functions, LIB 805–806
tanh function, LIB 805–806
  intrinsic form, PT 11
_tanhl function, LIB 805–806
  intrinsic form, PT 11
_tanl function, LIB 805–806
  intrinsic form, PT 11
Target files
  defined, NMAKE, ET 646
  dependency lines, NMAKE, ET 655–658
Targets
  building, NMAKE, ET 648
  compiling, PWB, ET 163–164
  function, PWB, ET 58–61
  makefiles, PWB, ET 62–63

targets field
  NMAKE, ET 647
/Tc option
  CL, ET 541–542
Technical support. *See* the Product Assistance Request Form in LIB
Tell dialog box
  PWB, ET 216–218
_tell function, LIB 807–808
Tell function, PWB, ET 154, 216–218
  changing key assignment, ET 122
  executing, ET 108
Telephone support. *See* the Product Assistance Request Form in LIB
tellg member function
  input streams
    iostream classes tutorial, XUG 390–391
  istream class, XRF 884
tellp member function
  ofstream class
    iostream classes tutorial, XUG 376
  ostream class, XRF 904–905
TEMP environment variable
  defined, ET 817
Template program
  Phone Book sample program, XUG 152–153
Templates
  accelerator table resource
    dialog boxes, adding, XUG 156
    Phone Book sample program, XUG 198
  collection classes, creating
    Foundation classes cookbook, XUG 271
  dialog boxes, adding
    Foundation classes tutorial, XUG 156
  dialog resource
    Foundation classes tutorial, XUG 166, 224
  menu resource
    dialog boxes, adding, XUG 156
  resource
    (list), XUG 242
_tempnam function, LIB 809–811
Temporary files
  defined, ET 817
  LINK, ET 595
Temporary objects, LR+ 311–312; TUT 81
Terminate-and-stay-resident programs
  defined, ET 817
  DOS Shell command, ET 359
  graphics adapter requirements, PT 174
  installing, _dos_keep function, LIB 210–211
  Shell function, PWB, ET 214–215
Terminating
  atexit function, LIB 96–97
  calling processes, exit and _exit functions, LIB 251–252
  CodeView execution, ET 387–388
  for statements, LR 156
  programs, LR 30
  QuickWin programs, PT 148
  statements, LR 152
  _vheapterm function, LIB 847
  virtual memory manager, PT 91
Terminating, C++
  abort function
    described, LR+ 43
    immediate termination, LR+ 45
  assert macro
    described, LR+ 43
  atexit function
    described, LR+ 45
  default function, AfxAbort, XRF 61
  dialog boxes, modal, CDialog::EndDialog, XRF 267
  exit function
    described, LR+ 42
    initialization considerations, LR+ 44–45
  initialization considerations, LR+ 44–45
  linking to specified function, AfxSetTerminate, XRF 61
  methods, LR+ 42–43
  on fatal errrors, AfxTerminate, XRF 62
  return statement
    described, LR+ 43
    initialization considerations, LR+ 44–45
Ternary operator
  C++. *See* Conditional operator, C++
  defined, ET 817
Testing
  assumptions, validity of, XUG 33
  code, LR 3
  data models
    Foundation classes tutorial, XUG 58
  DMTEST sample program, XUG 58
  end-of-file
    _eof function, LIB 244–245
    on given stream, LIB 13
    on streams, feof function, LIB 267–268
  for extraction operators
    iostream classes tutorial, XUG 384
  object validity
    Foundation classes tutorial, XUG 32
  objects for class derivation, CObject::IsKindOf, XRF 472

Testing (*continued*)
- program validity
  - Foundation classes tutorial, XUG 137–138
- sample programs, XUG 49–64
- streams for errors, ferror function, LIB 269–270
- validity of object's internal state, XRF 53

Text
- alignment flags, retrieving status, CDC::GetTextAlign, XRF 205
- Arg function, PWB, ET 106–108
- caption titles, returning length, CWnd::GetWindowTextLength, XRF 704
- changing orientation
  - of font text output, _getgtextvector function, LIB 366
  - of output, _setgtextvector function, LIB 665
- colors
  - retrieving current, CDC::GetTextColor, XRF 207
  - setting, LIB 678–680; XRF 248
- computing line dimensions, CDC::GetTextExtent, XRF 207
- copying
  - CodeView commands, ET 353
  - Microsoft Advisor, ET 761
  - QuickHelp, ET 771
  - QuickWin, PT 148–149
- creating output, LIB 27–28
- current cursor attribute in text video mode, _gettextcursor function, LIB 379
- current position, _gettextposition function, LIB 380–381
- CWnd, setting, CWnd::SetDlgItemText, XRF 808
- deleting
  - PWB, ET 166, 232
- dialog boxes, retrieving, CWnd::GetDlgItemText, XRF 691
- drawing dimmed, CDC::GrayString, XRF 210–212
- editing
  - menu commands, PWB, ET 73
- finding
  - PWB, ET 91–93
- font-based, getting width in pixels, _getgtextextent function, LIB 365

Text (*continued*)
- fonts. *See* Fonts
- formatted
  - drawing in rectangle, CDC::DrawText, XRF 177, 179
  - HELPMAKE topics, ET 721
- getting from list boxes, CListBox::GetText, XRF 365
- indenting
  - PWB, ET 296
- lines, retrieving number of, CEdit::GetLineCount, XRF 292
- modes, LIB 695–696; PT 167–168
- pasting
  - Microsoft Advisor, ET 761
  - QuickHelp, ET 771
- printing
  - font-based in graphics mode, _outgtext function, LIB 537–539
  - graphics mode, _outtext function, LIB 545–546
  - specified length in graphics mode, _outmem function, LIB 540–541
- replacing
  - current selection in edit control, CEdit::ReplaceSel, XRF 297
  - PWB, ET 93–96
  - Qreplace function, PWB, ET 202
  - Replace function, PWB, ET 205–207
- scrolling
  - in multiple-line edit control, CEdit::LineScroll, XRF 296
  - in text window, _scrolltextwindow function, LIB 640–642
- searching
  - PWB, ET 85–90, 197
- selecting
  - PWB, ET 211
- setting
  - alignment flags, CDC::SetTextAlign, XRF 246–247
  - caption title to specified, CWnd::SetWindowText, XRF 816
  - justification, CDC::SetTextJustification, XRF 248–249

Key

ET	Environment and Tools	PT	Programming Techniques
LIB	Run-Time Library Reference	TUT	C++ Tutorial
LR	C Language Reference	XRF	Class Libraries Reference
LR+	C++ Language Reference	XUG	Class Libraries User's Guide

- Text (*continued*)
  - setting (*continued*)
    - position, _settextposition function, LIB 683–684
    - screen rows, _settextrows function, LIB 685–686
    - to specified integer value, CWnd::SetDlgItemInt, XRF 808
  - specifying length in an edit control, CEdit::LimitText, XRF 294
  - streams, setting mode to, ios& text, XRF 871
  - windows
    - captions, copying into specified buffer, CWnd::GetWindowText, XRF 704
    - creating, _settextwindow function, LIB 687
    - getting boundaries, _gettextwindow function, LIB 382
  - writing
    - horizontally on screen, _pg_hlabelchart function, LIB 569
    - string at specified location, XRF 258–259
    - vertically on screen, _pg_vlabelchart function, LIB 576
- _TEXT... constants, PT 169
- Text argument, Arg function, PWB, ET 106–108
- Text Argument dialog box, PWB
  - default key assignments, ET 150
  - Lasttext function, ET 175–176
  - Prompt function, ET 196–197
- Text box, PWB, ET 81
- Text files
  - *See also* Files
  - defined, ET 817
- Text segments in version 7.0, LR 259
- Text strings
  - embedding
    - CL option, ET 544
  - searching
    - PWB, ET 85–90
- Text switches
  - PWB, ET 122
- _TEXTMONO constant, PT 170
- TextOut member function
  - CDC class, XRF 259–260
- 32-Bit Specific margin notation described, LR+ 21
- 32-bit targeting
  - DOS Extender described, LIB xi
- 32-bit values
  - setting, combo-box item, CComboBox::SetItemData, XRF 155
- TH register
  - CodeView syntax, ET 419, 450
- this keyword
  - Foundation classes tutorial, XUG 104, 106
- this pointer
  - argument matching, overloaded functions, LR+ 345–346
  - assignment operator, TUT 75
  - described, LR+ 244–246
  - disambiguation, LR+ 401–402
  - modifying, TUT 76
  - overloading, PT 100–101
  - overview, TUT 74
  - returning *this, TUT 75
  - static member functions, TUT 88
- Thread defined, ET 817
- Thread ID defined, ET 817
- Thread of execution defined, ET 817
- Threshold
  - data, setting with CL option, ET 522
- THROW macro, XRF 66
  - Foundation classes tutorial, XUG 64
- THROW_LAST macro, XRF 67
  - Foundation classes tutorial, XUG 64
- ThrowErrno data member
  - CFileException class, XRF 325–326
- ThrowOsError data member
  - CFileException class, XRF 326
- Thunk
  - defined, ET 817
- tie member function
  - ios class, XRF 864
- Tilde (~)
  - menu command, PWB, ET 126
- Tile command
  - CodeView, ET 373–374
  - PWB
    - described, ET 77
    - predefined macros, ET 145
  - QuickWin, PT 150
- Tilemode switch, PWB, ET 264, 299–300
- Tiling windows
  - PWB, ET 258–259, 299–300
- Time
  - absolute, representing, CTime:operator+,-, XRF 615
  - adding and subtracting CTimeSpan object, CTime::operator+=, XRF 616
  - ANSI compatibility, LR 264
  - calculating calling process, clock function, LIB 154–155
  - calling BIOS time and date services, _bios_timeofday function, LIB 125–126

Time (*continued*)
comparing absolute, CTime comparison operators, XRF 616
converting
local to calendar, mktime function, LIB 521–522
to character strings, ctime function, LIB 173–174
values and correcting for zone, localtime function, LIB 458–459
values to structures, gmtime function, LIB 394–395
copying to buffers, _strtime function, LIB 788–789
creating CTime object, CTime::CTime, XRF 608–609
CTime class described, XRF 606
current, getting
CTime::GetCurrentTime, XRF 610
_ftime function, LIB 331–332
PWB, ET 166
current, setting
Foundation classes cookbook, XUG 255
day
of month, CTime::GetDay, XRF 611
of week, CTime::GetDayOfWeek, XRF 611
diagnostic dumping and storing to archive, CTime::operators <<,>>, XRF 617
elapsed
calculating, XUG 256
string representation, formatting, XUG 256
environment variables, setting, _tzset function, LIB 820–822
finding difference between two times, difftime function, LIB 176–177
formatting strings, strftime function, LIB 757–758
functions
described, LIB 58–59
(list), LIB 58
generated formatted string
converted, CTime::Format, XRF 609
unconverted, CTime::FormatGmt, XRF 610
getting struct tm with local time decomposition, CTime::GetLocalTm, XRF 612
getting struct tm with UCT decomposition, CTime::GetGmtTm, XRF 611

Time (*continued*)
getting time file written, _dos_getftime function, LIB 204–206
getting time_t value for CTime object, CTime::GetTime, XRF 614
hours, getting, CTime::GetHour, XRF 613
management, described
Foundation classes cookbook, XUG 255–256
minutes, getting, CTime::GetMinute, XRF 613
months, getting, CTime::GetMonth, XRF 613
optimizing, PWB, ET 54–55
seconds, getting, CTime::GetSecond, XRF 614
setting
file modification, _utime function, LIB 834–835
for files, _dos_setftime function, LIB 224–226
source, copying into CTime object, CTime::operator=, XRF 615
span
adding and subtracting, CTimeSpan::operator+,-, XRF 625
comparing two relative time values, CTimeSpan comparison operators, XRF 626
copying source object, CTimeSpan::operator=, XRF 625
creating CTimeSpan objects, CTimeSpan::CTimeSpan, XRF 620–621
CTimeSpan class described, XRF 618
days, getting, CTimeSpan::GetDays, XRF 622
diagnostic dumping and storing to archive, CTimeSpan::operators<<,>>, XRF 627
generating formatted string corresponding to CTimeSpan, CTimeSpan::Format, XRF 621
hours in current day, CTimeSpan::GetHours, XRF 622
hours, total, CTimeSpan::GetTotalHours, XRF 623
minutes in current hour, CTimeSpan::GetMinutes, XRF 623
minutes, total, CTimeSpan::GetTotalMinutes, XRF 624
seconds in current minute, CTimeSpan::GetSeconds, XRF 623

Time (*continued*)
  span (*continued*)
    seconds, total, CTimeSpan::GetTotalMinutes, XRF 624
  structures, converting to character strings, asctime function, LIB 88–89
  system
    called after change, CWnd::OnTimeChange, XRF 794
    getting, _dos_gettime function, LIB 207–208
    getting, time function, LIB 812–814
    setting, _dos_settime function, LIB 227–228
  years, getting, CTime::GetYear, XRF 614
time function, LIB 812–814
__TIME__ macro
  default, LR 247
  described, LR 198–199
__TIME__ macro, C++, LR+ 376
Time stamps
  changing, NMAKE, ET 650
  defined, ET 646, 817
  displaying, NMAKE, ET 648
Time zone, LR 255
Timers
  called at specified intervals, CWnd::OnTimer, XRF 795
  killing specified event, CWnd::KillTimer, XRF 711
  system, installing, CWnd::SetTimer, XRF 813–814
Timersave switch, PWB, ET 265, 300
__TIMESTAMP__ macro, C++, LR+ 376
Timezone variables, LIB 62
Tiny memory models
  defined, ET 817
  described, PT 60
  option, CL, PT 63
/TINY option
  LINK, ET 592–593
title pragma
  precompiled header compilation, effect on, PT 41
title pragma, C++, LR+ 391
Titles, presentation graphics, PT 220–221
_titletype structures, presentation graphics, PT 220–221
TL register
  CodeView syntax, ET 419, 450
TMP environment variable
  defined, ET 817
  starting PWB, ET 67
.TMP files
  defined, ET 817
tmpfile function, LIB 815–816
tmpnam function, LIB 809–811
Tmpsav switch, PWB, ET 265, 300–301
__toascii function, LIB 817–819
Toggle State-File Reading option
  CodeView, ET 343
Toggling
  defined, ET 817
Tokenization
  translation phase, LR 191
Token-pasting operator (##), LR 195, 197
Token-pasting operator, C++, LR+ 373
Tokens
  described, LR 1–4
  finding next in string, strtok and _fstrtok functions, LIB 794–796
Tokens, C++, LR+ 2–3
tolower function, LIB 817–819
.topic command
  HELPMAKE, ET 724
TOOLS.INI file
  See also Getting Started
  CodeView
    configuring, ET 329–330
    entries, ET 330–336
    installing, ET 328–329
    remote debugging, ET 393–395
    setting options, ET 344
  defined, ET 817
  NMAKE, ET 652
  PWB
    autoloading extensions, ET 131
    comments, ET 136
    extension switches, ET 265
    filename-extension tags, ET 132–133
    Initialize function, ET 172–173
    line continuation, ET 136
    macros, ET 113, 115
    named tags, ET 133–134
    operating-system tags, ET 132
    sections tags, ET 132
    switch syntax, ET 135
TOOLS.PRE file
  CodeView, installing, ET 328
Topic command
  CodeView, ET 374–375
  HELPMAKE, ET 724
  PWB, ET 78
    predefined macros, ET 146
Topic lists
  Microsoft Advisor, ET 765

Topic: command, PWB
  described, ET 757
Topics
  help files, linking, ET 716–719
TopLeft member function
  CRect class, XRF 529
_toupper function, LIB 817–819
/Tp option
  CL, LR 261
Trace command, CodeView, ET 423, 452
  controlling execution, ET 386
TRACE macro, XRF 56
  diagnostic output
    described, XUG 21
  DMTEST sample program, XUG 34
  exceptions, defined, XUG 20
  Foundation classes
    cookbook, XUG 288
    tutorial, XUG 50
Trace speed command, CodeView, ET 368–369, 423, 433, 453
Tracepoint defined, ET 817
Tracing
  defined, ET 817
  functions, CodeView, ET 452–453
TrackPopupMenu member function
  CMenu class, XRF 436
Traildisp switch, PWB, ET 265, 301
Trailing
  lines, display mode, in PWB, ET 301–302
  spaces, display mode, in PWB, ET 301–302
Traillines switch, PWB, ET 265, 301–302
Traillinesdisp switch, PWB, ET 265, 302
Trailspace switch, PWB, ET 265, 302
Transferring control
  out of nested structure, LR 152
  to error handler, _set_new_handler functions, LIB 672–675
Transforming strings based on locale-specific information, strxfrm function, LIB 799–800
Transition from machine code to p-code, PT 47
Translating
  CWinApp messages before dispatched to DispatchMessage function, CWnd::PreTranslateMessage, XRF 801
  far calls to near calls, PT 25–26
  source code
    to assembly code, CL, ET 501–503, 505
    to machine-code, CL, ET 502–503, 505
  text of specified dialog box control into integer value, CWnd::GetDlgItemInt, XRF 690–691
  white space, PWB, ET 128
Translation
  See also Phases of translation
  hiding identifier names, LR 45
  phases, LR 190–191; LR+ 1–2, 395–396
  tokens converted into object code, LR 191
  trigraphs, LR 9
  units
    compiling, LR 26
    defined, LR 25, 190
    warnings, LR 266
  units, C++
    defined, LR+ 2–3, 395–396
    linkage. *See* Linkage, C++
TranslatorAccelerator Windows function, XRF 787
Transport
  entry, TOOLS.INI file
    CodeView, ET 328, 330, 335–336
    remote debugging, ET 393–395
  layer, specifying, CodeView, ET 335–336
Trapping
  interrupting
    CodeView, ET 341–342
Triangles, calculating hypotenuse, _hypot and _hypotl functions, LIB 424–425
Trigraphs
  confusion with question mark, LR 18
  converting to single characters, LR 191
  defined, LR 8
Truncated files
  building a database, ET 733, 735

Truncation
  division operations, LR 127
  text files, LR 250
TRY macro, XRF 67
  Foundation classes cookbook, XUG 298–302
  Foundation classes tutorial, XUG 51, 63
TSF option
  CodeView, ET 338, 343
TSR
  See also Terminate-and-stay-resident programs
  defined, ET 817
  installing, _dos_keep function, LIB 210–211
Tutorial
  Foundation classes. *See* Foundation class tutorial
  iostream classes. *See* iostream classes
  PWB, conventions, ET 7
Twips defined, ET 726
Two's-complement arithmetic
  portability guidelines, PT 280–281
/Tx option
  CL, ET 541–542
.TXT files
  defined, ET 817
Type casting defined, ET 817
Type casts
  conversions, LR 147–150
  described, LR 108, 126
  objects, LR 148
  pointers, LR 149, 241
  rounding of numbers, LR 240
  table of legal, LR 148
  unsigned integers, LR 143–144
Type checking
  ANSI compliant, LR 263
  performed by compiler, LR 149
Type conversions, C++
  See also Conversions, C++
  described, LR+ 312–313
TYPE operator
  inline assembler use, PT 114
Type names, C++
  class declarations, using in, LR+ 238
  class scope, effect, LR+ 257
  declarators, use in, LR+ 185–187
  defining, LR+ 59–60
Type qualifiers
  described, LR 52–53
  nonterminal, LR 43
Type size, graphics, PT 193
Type specifiers
  overview, LR 51–52
  required in declarators, LR 55
  return types, LR 177–178
Type specifiers, C++, LR+ 168–173
typedef keyword
  described, LR 101–104
  improving code readability, LR 103
  names of identifiers in same scope, LR 39
  simplifying declarators, LR 89
  storage-class specifier nonterminal, LR 44
typedef keyword, C++, LR+ 59–60
typedef names
  inline assembly using in, PT 115–116
typedef names, C++
  introduction by declaration statements, LR+ 149
  linkage, LR+ 35–36
typedef specifier, C++, LR+ 163–167
typedef statements, C++
  class naming, LR+ 234
Typefaces, graphics, PT 193–195
Types
  aggregate, LR 93–97
  assigning to constants, LR 15
  cast to assign integer value, LR 64
  characters, LR 17
  conditional operations, LR 136–138
  conversions, LR 126
  converting, LR 126, 149
  creating, LR 102
  double, LR 11, 99
  enumeration, LR 51, 244
  float, LR 11, 99
  incomplete, LR 100–101
  inline assembly, PT 114
  integer constants, LR 14, 17
  integral conversions, LR 141–146
  long, LR 11, 14
  long double, LR 11, 100, 267
  mixed-language programming, PT 258–259, 265
  modifying, LR 55
  names, inline assembly, using in, PT 115–
  pointers conversions, LR 146–147
  portability guidelines, PT 271–274
  promoting, portability guidelines, PT 285–287
  scalar, LR 121
  size_t, LR 125
  standard. *See* Standard types
  storage, LR 98–100

Types (*continued*)
string literals, LR 21
unsigned, LR 14
user-defined, mixed-language programming, PT 265
Types, C++
aggregate
initializing, LR+ 219–222
class. *See* Class types
conversions. *See* Conversions
defined, LR+ 25–26
derived
composed, LR+ 58–59
described, LR+ 52
directly derived, LR+ 52–57
described, LR+ 49
floating
described, LR+ 50–51
fundamental
conversions. *See* Conversions
described, LR+ 50–52
integral
described, LR+ 50–51
__segment
described, LR+ 50
void
described, LR+ 50
Type-safe member functions, defining, Foundation classes cookbook, XUG 333
tzname variable, LIB 62
_tzset function, LIB 820–822

U

U command, CodeView, ET 423, 453–454
:u command, HELPMAKE, ET 724
\u formatting attribute
HELPMAKE, ET 721
/U option, CL, ET 542–544
/u option
CL, ET 542–544; LR 267
Unary operators
defined, LR 111
described, LR 122–125

/u option (*continued*)
on pointers
illegal, LR 265
UCHAR_MAX constant, C++
integral limits, LR+ 62
UINT_MAX constant, C++
integral limits, LR+ 62
\ul formatting code
HELPMAKE, ET 727
ULONG_MAX constant, C++
integral limits, LR+ 62
_ultoa function, LIB 823–824
_umask function, LIB 21, 825–826
Unary negation operator, C++
overloading, LR+ 355
unary-operator expressions, LR+ 93–94
Unary operators
CodeView precedence, ET 406
defined, ET 818
preprocessing directives, NMAKE, ET 690–691
Unary operators, C++
address-of, LR+ 92–93
associativity, LR+ 91
decrement, LR+ 94–95
delete, LR+ 101–102
increment, LR+ 94–95
indirection, LR+ 92
(list), LR+ 91
logical NOT, LR+ 94
new, LR+ 97–101
one's complement, LR+ 94
overloading, LR+ 355–358
sizeof, LR+ 95–96
unary negation, LR+ 93–94
unary plus, LR+ 93
Unary plus operator, C++
overloading, LR+ 355
unary-operator expressions, LR+ 93
Unassemble command, CodeView, ET 423, 453–454
Unassembling
defined, ET 806, 818
p-code, ET 392
Unassigned function, PWB, ET 154, 218

unbuffered member function
  streambuf class, XRF 937
!UNDEF preprocessing directive
  NMAKE, ET 680, 690
#undef preprocessor directive, LR 190, 192, 194, 198
#undef preprocessor directive, C++, LR+ 373–374
Undefined macros
  NMAKE, ET 670
  PWB, ET 224
UNDEL
  command line, ET 749
  options, ET 749
  overview, ET 743, 747–748
  syntax, ET 749
Undelcount switch, PWB, ET 265, 303
Underflow conditions
  bitwise shift operators may cause, LR 131
  cause rounding of values, LR 241
  value set to zero, LR 248
underflow member function
  streambuf class, XRF 938
Underlining
  HELPMAKE code, ET 726
Underscore (_)
  document conventions, LIB xiv
  macros, NMAKE, ET 668
  regular expressions, PWB, ET 93
  symbol format, CodeView, ET 409
  usage in macro and keyword names, LR 58
Undo command
  CodeView, ET 360
  PWB
    described, ET 73
    predefined macros, ET 143
Undo flags
  clearing, resetting, CEdit::EmptyUndoBuffer, XRF 290
  returning edit operations status, CEdit::CanUndo, XRF 285
Undo function, PWB, ET 154, 218
Undo member function
  CEdit class, XRF 302
Undocount switch, PWB, ET 122, 265, 303
Undoing
  last operation in edit control, CEdit::Undo, XRF 302
ungetc function, LIB 827–828
_ungetch function, LIB 829–830
Unicode specification
  wide characters, LR 8
Union declarations
  accessing, LR 242
  defined, LR 54
  described, LR 71–73
  incomplete type, LR 100
  initializing aggregate types, LR 93–97
  members, LR 119–120
  tags, LR 39, 263
union keyword, C++
  class type declaration, LR+ 228
Union operator
  CRect class, XRF 531, 535
union type names, C++
  introduction by declaration statements, LR+ 149
UnionRect member function
  CRect class, XRF 530
Unions
  declaring, LR+ 228
  derivative types, LR+ 58–59
  described, LR+ 249–252
  portability guidelines, PT 276–277
UNIX
  case sensitivity, LIB 9
  compatibility, LIB ix
  naming conventions, LIB 8
  path-name delimiters, LIB 9
  predefined expression syntax, ET 778, 785
  programming, LIB xi
  regular expression syntax, ET 777–778, 781
    setting, in PWB, ET 303–304
  subdirectory conventions, LIB 9
Unixre switch, ET 303–304
  PWB, ET 264
  regular expression syntax, ET 91, 777
_unlink function, LIB 831–832
Unlocking
  range of bytes in file, CFile::UnLockRange, XRF 319
  virtual memory blocks, LIB 862; PT 93
UnlockRange member function
  CFile class, XRF 319
UnrealizeObject member function
  CBrush class
    CWnd::OnEraseBkgnd, XRF 738
  CGdiObject class, XRF 350
_unregisterfonts function, LIB 833
Unresolved external defined, ET 818
unsetf member function
  ios class, XRF 864

unsigned char type
  conversion, LR 144
  range, LR 98
unsigned char type, C++
  size, LR+ 51–52
unsigned int type, C++
  size, LR+ 51–52
Unsigned integers
  converting, LR 143–144
  forcing type, LR 14
  shifting, LR 131
  size, LR 99
Unsigned integral types
  table of conversions, LR 144
unsigned keyword
  with integral types, LR 51
unsigned long type, C++
  size, LR+ 51–52
Unsigned numbers
  predefined expression syntax, ET 778, 780, 786
unsigned short type, C++
  size, LR+ 51–52
unsigned type, C++, conversion
  from signed, LR+ 67
  to signed, LR+ 67–68
Up function, PWB, ET 154, 219
Update region
  retrieving coordinates of smallest rectangle that encloses, CWnd::GetUpdateRect, XRF 700–701
  retrieving into specified region, CWnd::GetUpdateRgn, XRF 701
UpdateColors member function
  CDC class, XRF 260
UpdateWindow member function
  CWnd class, XRF 819
Updating
  client areas, CWnd::UpdateWindows, XRF 819
Uppercase
  converting strings to, _strupr and _fstrupr functions, LIB 797–798
Use 8514 Displays option
  CodeView, ET 339
Use Black-and-White Display option
  CodeView, ET 339
USE command, CodeView, ET 423, 454–455
Use Language command, CodeView, ET 423, 454–455
Use Two Displays option
  CodeView, ET 338
Use VGA Displays option
  CodeView, ET 339
User input
  Windows
    Foundation classes cookbook, XUG 351–357
User input statements, ET 114–117
User interface, QuickWin, PT 147–148
User switch, PWB, ET 265–266, 304–306
Usercmd function, PWB, ET 154, 219
User-defined inference rules, writing, NMAKE, ET 682–684
User-defined macros, NMAKE
  creating, ET 668
  special characters, ET 668
  where to define, ET 669–670
User-defined type, defined, ET 818
USHRT_MAX constant, C++
  integral limits, LR+ 62
Using Help command
  QuickWin, PT 151
Usual arithmetic conversions
  bitwise shift operators, LR 131
  bitwise-NOT operator, LR 124
  described, LR 115–116
  function prototype, LR 150
  logical operators don't perform, LR 135
  multiplicative operators, LR 126–128
  performed on parameters, LR 180
  relational operators, LR 132–133
UTILERR.HLP file, ET 771
Utilities
  Make P-Code (MPC), PT 44, 51–53
  message handler support
    Foundation classes tutorial, XUG 199
Utilities extension
  PWB Options menu, ET 75

Utility member functions
  Phone Book sample program, XUG 238–241
UTILS.HLP file, ET 771, 773
_utime function, LIB 834–835

V

\v (escape sequence), vertical tab, LR 18
\v formatting attribute
  HELPMAKE, ET 718, 721
\v formatting code
  HELPMAKE, ET 727
/V option
  CL, ET 544
  EXEHDR, ET 631, 635–639
  HELPMAKE, ET 715
  NMAKE, ET 650, 679
/v option
  BSCMAKE, ET 737
va_arg function, LIB 836–839
va_end function, LIB 836–839
ValidateRect member function
  CWnd class, XRF 820
ValidateRgn member function
  CWnd class, XRF 820
Validating client area
  within given rectangle, CWnd::ValidateRect, XRF 820
  within given region, CWnd::ValidateRgn, XRF 820
Validity checking of objects, CObject::AssertValid, XRF 466
Values
  calculating
    ceilings, ceil and _ceill functions, LIB 133–134
    floors, floor and _floorl functions, LIB 285–286
  characters, LR 17
  constants, LR 13
  converting to void, LR 147
  decimal, LR 10
  entering, CodeView, ET 440–441
  enumeration, LR 63
  floating-point, LR 10–13, 145
  fractions, LR 10
  function return, LR 160–161
  getting
    environment table, getenv function, LIB 360–361
    register, _dosexterr function, LIB 234–235
  hexadecimal, LR 135
  integers, LR 10, 17, 64, 70
  left, LR 107
  locator, LR 107
Values (*continued*)
  passing, LR 190
  pointers, LR 146
  presentation graphics, PT 203
  range of signed integers, LR 99
  referring to with identifiers, LR 5
  returning
    floating-point types, functions, declaring, PT 130
    inline assembly, registers, PT 120–121
    maximum, __max function, LIB 486
    smallest of two, __min function, LIB 515
  searching for, _lsearch function, LIB 469–470
  shifting, LR 131
  unsigned, LR 143
  variables, LR 47, 91–97
Variable declarations
  external level, LR 45
  lifetime, LR 43
  placement, LR 44; TUT 13–14
Variable-length argument lists, LIB 59; LR 171
Variables
  accessing, LR 27
  addresses
    debugging assembly code, ET 414
  aggregate type
    initializing, LR 93–97
  allocating storage for, LR 27
  arrays, addresses, mixed-language programming, PT 266–267
  automatic
    storage-class specifier, LR 47
    values affected by execution, LR 44
    with storage class extern, LR 153
  based
    declaring, PT 65–66
  changing to constants, PT 9
  common subexpression elimination, PT 8, 23
  dead-store elimination, PT 8
  declared with register keyword, LR 153
  declaring
    floating-point types, PT 127–129
    mixed-language programming, PT 258–259
    near, far, huge and based, PT 65–66
  defined, ET 818
  editing, CodeView, ET 351
  environment
    See also Environment variables
    presentation graphics, PT 219–227
  far
    declaring, PT 65–66
  floating-point, PT 128

- Variables (*continued*)
  - global. *See* Global variables huge
    - declaring, PT 65–66
  - inline assembly, PT 114, 117–118
  - lifetime, LR 34
  - local
    - allocating, p-code, PT 52
    - CodeView, ET 354
  - multiple, LR 42, 61
  - near
    - declaring, PT 65–66
  - register
    - declaring as, portability guidelines, PT 288–289
    - storage, __asm block effect on, PT 124–125
  - .SBR files, PWB, ET 104
  - scalar type, LR 91, 93
  - scope, CodeView, ET 445–447
  - simple declarations, LR 61–62
  - storage classes
    - described, LR 43–50
  - strings, LR 97–98
  - type specifiers
    - overview, LR 51–52
  - values
    - setting, LR 91–97
    - undefined, LR 47
  - visibility, LR 45, 48
- Variables, C++
  - automatic
    - described, LR+ 46
    - initialization, LR+ 47–49
  - defined, LR+ 21, 25–26
  - external
    - described, LR+ 47
  - floating-point, precision, setting, ios::precision, XRF 861
  - internal field width, setting, ios::width, XRF 865
  - internal fill character, setting, ios::fill, XRF 857
  - local
    - initialization, LR+ 47–49
  - numerical limits
    - floating, LR+ 63–64
    - integral, LR+ 62
- Variables, C++ (*continued*)
  - object state, providing without class derivation, ios::xalloc, XRF 865
  - objects, compared to, LR+ 25–26
  - reference-type
    - initializing, LR+ 223–224
  - register
    - described, LR+ 46–47
  - static
    - described, LR+ 46
    - initialization, LR+ 47–49
- va_start function, LIB 836–839
- Variant records. *See* Union declarations
- VCPI server. *See* Virtual Control Program Interface server
- /VERBOSE option
  - EXEHDR, ET 631, 635–639
- Verbose output
  - BSCMAKE, ET 737
  - DOS header information, ET 637
  - HELPMAKE option, ET 715
  - OS/2 header information, ET 637–638
- VERIFY macro, XRF 57; XUG 203, 289
- Vertical Scrollbars command, CodeView, ET 368, 370
- Vertical tab (escape sequence), LR 18
- VESA, PT 173–175
- vfprintf function, LIB 840–843
- _vfree function, LIB 844
- VGA
  - defined, ET 818
  - display, specifying, CodeView, ET 339
  - screen modes, LR 261
- _vheapinit function, LIB 845–846; PT 90–91
- _vheapterm function, LIB 847; PT 91
- Video
  - configuration
    - checking, PT 171–172
    - selecting, PT 173
  - getting graphics configuration information, _getvideoconfig function, LIB 383–385
  - graphics adapter defined, ET 818

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

Video Electronics Standards Association, PT 173–175
Video modes
  controlling functions (list), PT 186–188
  described, PT 167–168
  selecting, PT 173
  setting, LIB 690–696; PT 168–169, 171
  Super VGA support, PT 173–175
View Back command, QuickHelp, ET 770
View coordinates
  moving origins, _setvieworg function, LIB 697–698
  translating to window coordinates, _getwindowcoord function, LIB 391
  translating to, _getviewcoord functions, LIB 386–387
View History command, QuickHelp, ET 770
View Last command, QuickHelp, ET 770
View Memory command, CodeView, ET 423, 436–437, 455–457
View menu
  PWB, ET 770
View Next command, QuickHelp, ET 770
View Output command, CodeView, ET 373–374
View References command, PWB
  searching, ET 86
View Relationship command, PWB, ET 76
  function, ET 145
View Source Command, CodeView, ET 423, 457–460
  line numbers, ET 400
Viewports
  coordinates, PT 182–183
  modifying extents, CDC::ScaleViewportExt, XRF 228
  modifying origin, CDC::OffsetViewportOrg, XRF 217
  retrieving device contexts' extents, CDC::GetViewportExt, XRF 209
  retrieving origin coordinates associated with device context, CDC::GetViewOrg, XRF 209
  setting
    origin of device context, CDC::SetViewportOrg, XRF 251
    x- and y-extents, CDC::SetViewportExt, XRF 250
Viewports, creating, _setviewport function, LIB 699–700
Virtual base classes
  described, LR+ 268–271
  design issues, TUT 180
  overview, TUT 133–134
Virtual Control Program Interface server
  defined, ET 818
  memory management, CodeView, ET 336
Virtual destructors, TUT 129
Virtual function table pointers. *See* V-table pointers
Virtual functions, C++
  abstract classes, LR+ 265–266
  accessing, LR+ 296–297
  described, LR+ 265, 275–279
  Foundation classes tutorial, XUG 203
  overhead, TUT 125, 127
  overview, TUT 120–123
  pure, TUT 127–129
virtual keyword
  Foundation classes tutorial, XUG 90
Virtual memory
  allocation functions (list), LIB 60
  blocks
    dirty vs. clean, specifying, PT 92
    loading, PT 92
    locking and unlocking, PT 93
  browser database, ET 733–734
  defined, ET 818
  handles, PT 91–92
  using, techniques, PT 93–96
Virtual memory manager
  described, PT 90
  initializing, PT 90–91
  terminating, PT 91
virtual specifier, C++
  described, LR+ 163
Virtual sync function
  streambuf class
    streambuf::sync, XRF 937
Virtual underflow function
  streambuf class, streambuf::underflow, XRF 938
Visibility
  external-level variables, LR 45
  function declarations, LR 44
  rules for function declarations, LR 50
    static storage class, LR 177
  global
    with /Ze option, LR 265
  overview, LR 35–36
  static definitions, LR 45
  table listing, LR 37
  variables, LR 44–48
VK_DELETE constant
  Foundation classes tutorial, XUG 198
VK_RETURN constant
  Foundation classes tutorial, XUG 198

_vload function, LIB 848–850; PT 92
_vlock function, LIB 851–853; PT 93
_vlockcnt function, LIB 854–856
VM command, CodeView, ET 423, 455–457
  and Restart command, ET 436–437
_vmalloc function, LIB 857–858; PT 91–92
_vmsize function, LIB 859
void expressions
  sequential evaluations, LR 140
  side effects, LR 51
__void keyword, PT 86, 239
  arguments to functions, LR 183–185
  expression list, LR 185–187
  if no arguments passed, LR 180
  uses, LR 51
Void pointers
  16-bit words keyed by, XRF 375
  arrays, CPtrArray class described, XRF 517
  CPtrList class described, XRF 519
  keyed to void pointers, XRF 373
  maps keyed by 16-bit words, XRF 393
  maps keyed by CString objects, XRF 387
  to identifier of unspecified type, LR 86
void*() operator
  ios class, XRF 866
void type
  as function return value, LR 160–161
  function calls, LR 119
  incomplete type, LR 101
void type, C++
  described, LR+ 50
  pointer conversions, LR+ 71
volatile keyword
  accessing objects, LR 244
  listed, LR 53
  modifying typedef, LR 102
  permitting value changes, LR 52
  repeating, LR 264
  type-qualifier nonterminal, LR 43
  using, LR 53
volatile keyword, C++
  pointers, effect on, LR+ 188–190
  this pointer modification, LR+ 246
vprintf function, LIB 840–843
_vrealloc function, LIB 860–861
_VRES2COLOR constant, PT 170
_VRES16COLOR constant, PT 170
_VRES256COLOR constant, PT 170
VS command, CodeView, ET 423, 457–460
  line numbers, ET 400
Vscroll switch, PWB, ET 265, 306
_vsnprintf function, LIB 840–843
  controlling string size, LR 260
vsprintf function, LIB 840–843
V-tables
  described, Foundation classes tutorial, XUG 203
  message maps, similarity to, XUG 96
  naming, CL option, ET 528–530
  pointers, PT 102–103
  tutorial information, TUT 125, 127
_vunlock function, LIB 862; PT 93

W

/W option
  CL, ET 544–545
  HELPMAKE, ET 713, 721
  LINK, ET 593
/W4 option
  CL, LR 6
W? command, CodeView, ET 423, 460
_wabout function, LIB 863–864; PT 151, 157
/WARNFIXUP option
  LINK, ET 593
Warning level
  setting, CL options, ET 544–545
warning pragma
  compiler errors, LR 213
  new in version 7.0, LR 258
warning pragma, C++, LR+ 391–392
Warnings
  See also "Error Messages," Part 2 of this book
  compiling errors, LR 213, 262
  declarations, LR 266
  escape sequence unrecognized, LR 265
  generated by untyped variables, LR 61
  increased number, reasons for, LR 266

Warnings (*continued*)
  indirection to different types, LR 263
  macro redefinition, LR 265
  overflow on constant expressions, LR 266
  packing size, LR 265
  translation units, LR 266
Watch command, CodeView, ET 373–374
Watch expressions
  adding, ET 364, 460
  deleting, ET 461
  listing, ET 365, 465–466
  saving, ET 344
  setting, ET 326–327
Watch window, CodeView
  exploring watch expressions, ET 327
  function, ET 351
  opening, ET 374
  overview, ET 348
Watchpoint
  defined, ET 818
WC command, CodeView, ET 423, 461
_wclose function, LIB 865–866; PT 162–163
wcstombs function, LIB 867–868
  translating characters, LR 8
wctomb function, LIB 869–870
  translating characters, LR 8
WDG command, CodeView, ET 377, 382, 387–388, 423, 462
WDL command, CodeView, ET 377, 383, 423, 462
WDM command, CodeView, ET 383, 423, 464
Wedges, drawing, _pie functions, LIB 577–579
Weeks, getting days of, CTime::GetDayOfWeek, XRF 611
_wgetexit function, LIB 871–872; PT 163
_wgetfocus function, LIB 873–874; PT 152, 161–162
_wgetscreenbuf function, LIB 875–876; PT 161
_wgetsize function, LIB 877–879; PT 153, 160
WGH command, CodeView, ET 377, 386, 423, 463–464
Which Reference command, PWB
  described, ET 76
  function, ET 145
while statements
  described, LR 164
  iterations, LR 154
  terminating, LR 152
while statements, C++, LR+ 142–143
White, color value, ET 273
_WHITE constant, PT 180
White space
  C++, LR+ 2–4
  converting, PWB, ET 276–277
  predefined expression syntax, ET 778, 780, 785
  searching, PWB, ET 91, 93
  tab switches, PWB, ET 127–128
  translating, PWB, ET 292
WHITERECT structure
  CStatic::Create, XRF 566
White-space characters
  argument delimiters, LR 32
  defined, LR 2
  ignored, LR 195
  in floating-point constants, LR 10
  restrictions, LR 14, 194
  separating # and pragma, LR 209
Width
  HELPMAKE text, ET 713
  internal field variable, setting, ios::width, XRF 865
  streams, setting internal field parameter, setw, XRF 871
  switch, PWB, ET 265, 306
Width member function
  CRect class, XRF 530
width member function
  ios class, XRF 865
Wildcards, ET 95
  C++, usage described, LR+ 39
  defined, ET 818
  expansion, LR 31–32
  HELPMAKE syntax, ET 711
  listing files, PWB, ET 94
  makefile names, NMAKE, ET 653–654
  regular expression syntax, ET 778, 780–781, 786
  SBRPACK, ET 740
  SETARGV.OJB files, PWB, ET 50
_WINARRANGE constant, PT 163
_WINBUFINF constant, PT 161
_WINBUFDEF constant, PT 161
_WINCASCADE constant, PT 163
_WINCURREQ constant, PT 160
Windlllibs switch, PWB, ET 310–311
Window classes
  designing, TUT 183
  hierarchy, TUT 197–200
  interfaces, TUT 187–191
  (list), XRF 6
  refining, TUT 184–185
  relationships, TUT 186
  requirements, TUT 181–182

Window coordinates, PT 184–185
Window function, PWB, ET 154, 220
Window menu
  PWB
    described, ET 77
    predefined macros, ET 145
  QuickWin, PT 150–151, 163–164
WindowFromPoint member function
  CWnd class, XRF 821
WindowProc member function
  CWnd class, XRF 821
Windows
  activating or deactivating, CWnd::OnActivate, XRF 715
  active. *See* Active window
  active CWnd object, returning pointer to, CWnd::GetActiveWindow, XRF 685
  applications. *See* Windows applications
  arranging
    QuickWin, PT 149–150
  base class (list), XRF 6
  BEGIN_MESSAGE_MAP macro
    Foundation classes cookbook, XUG 313–315
  bitmaps, loading, CBitmap::LoadOEMBitmaps, XRF 115–116
  C++ language features, XRF 8
  called when
    activating for different task, CWnd::OnActivateApp, XRF 716
    clipboard contents emptied, CWnd::OnDestroyClipboard, XRF 731
    device-mode settings changed, CWnd::OnDevModeChange, XRF 731
  caption titles
    copying into specified buffer, CWnd::GetWindowText, XRF 704
    returning length, CWnd::GetWindowTextLength, XRF 704
  carets, getting current position, CWnd::GetCaretPos, XRF 686
  changing
    position and dimensions, CWnd::MoveWindows, XRF 714–715

Windows (*continued*)
  changing (*continued*)
    size, position, ordering, CWnd::SetWindowPos, XRF 814–816
  child
    See also Child windows
    notification messages from, XUG 316–317
  classes
    categories, XRF 11–20
    described, XRF 5
    registration, XRF 12, 37
  client. *See* Client windows
  clipboard
    called for each window in viewer chain when contents change, CWnd::OnDrawClipboard, XRF 731–732
    viewer, getting first window in, CWnd::GetClipboardViewer, XRF 687
  closing, signaling confirmation, CWnd::OnClose, XRF 720
  CMenu class described, XRF 19
  CodeView
    8087 window, ET 355–356
    Command window, ET 351–353, 417
    Help window, ET 357–358
    Local window, ET 354
    Memory windows, ET 356–357
    navigation, ET 349
    opening, ET 373
    overview, ET 347–348
    Register window, ET 354–355
    Source windows, ET 350
    Watch window, ET 351
  colors, called when change made, CWnd::OnSysColorChange, XRF 788
  COMMDLG.DLL file
    Foundation classes tutorial, XUG 153, 167
  communication with
    Foundation classes tutorial, XUG 95–101
  compatibility, LIB xi
  containing given point, identifying, CWnd::WindowFromPoint, XRF 821

Windows (*continued*)
control
Foundation classes cookbook, XUG 335
controls, XRF 16
coordinates, translating view coordinates to, _getwindowcoord function, LIB 391
creating
CMainWindow constructor, XUG 92–93
containing application-supplied message, CWnd::MessageBox, XRF 711–714
Create member function, XRF 711–714
HELLO sample program, XUG 91–92
CWnd
displaying, CWnd::ShowWindow, XRF 818–819
making active, CWnd::SetActiveWindow, XRF 805–806
retrieving current font, CWnd::GetFont, XRF 692
DECLARE_MESSAGE_MAP macro
Foundation classes cookbook, XUG 313–314
default procedure, calling
CWnd::Default, XRF 673
CWnd::DefWindowProc, XRF 674
defining parameters for initialization, CREATESTRUCT structure, XRF 78–79
dialog boxes
derived controls, using in, XUG 340
deriving from CDialog class, XUG 332
Foundation classes cookbook, XUG 329–335
main window, using as, XUG 334–335
modal, creating, XUG 329
modal, customizing, XUG 331
modal, initializing, XUG 330
type-safe member functions, XUG 333
dialog classes (list), XRF 6
display context, retrieving, CWnd::GetWindowDC, XRF 703
displaying
CWnd::ShowWindow, XRF 818–819
HELLO sample program, XUG 93
edit control. See Windows edit control
editing tools, XUG 156
enabling for mouse and keyboard input, CWnd::IsWindowEnabled, XRF 710
END_MESSAGE_MAP macro
Foundation classes cookbook, XUG 313, 315
flashing once, CWnd::FlashWindow, XRF 683–684
fonts, called when changing, CWnd::OnFontChange, XRF 738–739

Windows (*continued*)
frame
See also Frame windows
CFrameWnd class described, XRF 336
changing, XUG 324
classes (list), XRF 6
creating, XUG 311
matching to MDI child windows, XUG 324
GDI objects
See also GDI objects
attaching, CGdiObject::Attach, XRF 344
CGdiObject class described, XRF 342
deleting from memory, CGdiObject::DeleteObject, XRF 346
detaching, CGdiObject::Detach, XRF 347
retrieving handle to, CGdiObject::CreateStockObject, XRF 345–346
getting boundaries of current text windows, _gettextwindow function, LIB 382
global functions described, XRF 7
handles
detaching from CWnd object, CWnd::Detach, XRF 676
getting, CWnd object, CWnd::Detach, XRF 676
hiding, called when, CWnd::OnShowWindow, XRF 783–784
iconic, specifying, CWnd::IsIconic, XRF 710
icons, changing
Foundation classes cookbook, XUG 326
initialization file, called after change made, CWnd::OnWinIniChange, XRF 798–799
input control, CWnd::OnGetDlgCode, XRF 739–740
keyboard events, XUG 356–357
macros described, XRF 7
making CWnd into system-modal, CWnd::SetSysModalWindow, XRF 813
management
Foundation classes cookbook, XUG 311–327
manager's list, searching for next or previous window, CWnd::GetNextWindow, XRF 695
MDI
activating different child window, CMDIFrameWnd::MDIActivate, XRF 405
creating client window, CMDIFrameWnd::CreateClient, XRF 404
parent, accessing, Foundation classes cookbook, XUG 323

Windows (*continued*)
memory compacting specification,
CWnd::OnCompacting, XRF 722
menus
returning pointer to CWnd's, CWmd::GetMenu, XRF 693
specifying handle to, CMenu::Attach, XRF 419
message processing, XRF 12
message-based environment, XRF 8
messages
calling default procedure, CWnd::Default, XRF 673
direct calls, XRF 16
noncontrol, XRF 14
sending, CWnd::SendMessage, XRF 805
minimized, called if about to be dragged,
CWnd::OnQueryDragIcon, XRF 775
minimizing, CWnd::CloseWindow, XRF 664
modifying extents, CDC::ScaleWindowExt, XRF 229
mouse clicks, handling, XUG 351–352
mouse, tracking in
Foundation classes cookbook, XUG 353–356
nonclient area, called when needing change to indicate active or inactive state,
CWnd::OnNcActivate, XRF 761
notification messages, XRF 13–14
ON_COMMAND macro
Foundation classes cookbook, XUG 315, 317
ON_WM_NCDESTROY macro
Foundation classes cookbook, XUG 323
open, called when user requests,
CWnd::OnQueryOpen, XRF 777
origin
modifying, CDC::OffsetWindowOrg, XRF 217
retrieving coordinates, CDC::GetWindowOrg, XRF 210
overlapping
bringing CWnd to top of stack, XRF 661
informing of dimensions,
MEASUREITEMSTRUCT structure, XRF 83–84

Windows (*continued*)
overlapping (*continued*)
painting information, DRAWITEMSTRUCT structure, XRF 81–83
owner-draw control
creating, CWnd::CreateEx, XRF 666
painting, marking end, CWnd::EndPaint, XRF 682
painting text in
Foundation classes tutorial, XUG 101–102
palettes, called after changed,
CWnd::OnPaletteChanged, XRF 773–774
pop-up
changing size, position, ordering,
CWnd::SetWindowPos, XRF 814, 816
creating with extended style, CWnd::CreateEx, XRF 666–670
determining most recently active,
CWnd::GetLastActivePopup, XRF 692–693
presentation graphics
chart windows, PT 204
data windows, PT 204
structure types, PT 223–224
procedure, providing, CWnd::WindowProc, XRF 821
procedure, traditional, using
Foundation classes cookbook, XUG 320
Program Manager
Foundation classes tutorial, XUG 9–11, 14
programs, setting strings for About dialog boxes,
_wabout function, LIB 863–864
PWB
activating, ET 262–263
cascade arrangement, ET 231
closing, ET 232–233
maximizing, ET 179–180, 242
minimizing, ET 182–183, 243–244
moving, ET 183–184, 244
opening, ET 201, 220, 245–246
resizing, ET 254–255
restoring, ET 255–256
screen display, ET 67–68
selecting, ET 212

Windows (*continued*)
  PWB (*continued*)
    styles, in PWB, ET 221, 275
    tiling, ET 258–259, 299–300
  QuickWin
    activating, _wsetfocus function, LIB 892–893
    getting current screen buffer size, _wgetscreenbuf function, LIB 875–876
    getting current size and position, _wgetsize function, LIB 877–879
    opening file stream for, _fwopen function, LIB 335–337
    opening, _wopen function, LIB 882–884
    setting screen-buffer size, _wsetscreenbuf function, LIB 894–895
    setting size, screen position, _wsetsize function, LIB 896–897
  reading from, QuickWin, PT 159–160
  real mode, LR 260
  removing from clipboard viewer chain, CWnd::OnChangeCbChain, XRF 717–718
  restoring minimized to original size and position, CWnd::OpenIcon, XRF 800
  retrieving
    coordinates associated with device context, CDC::GetWindowExt, XRF 209
    pointer to CWnd with input focus, CWnd::GetFocus, XRF 692
  scrolling, CWnd::ScrollWindow, XRF 803–804
  scrolling text in, _scrolltextwindow function, LIB 640–642
  searching
    for name-specified, CWnd::FindWindow, XRF 683
    for next or previous on manager's list, CWnd::GetNextWindow, XRF 695
    window manager's list for, CWnd::GetWindow, XRF 702
  selecting
    QuickWin, PT 149–150
  setting
    caption title to specified text, CWnd::SetWindowText, XRF 816
    origin of device context, CDC::SetWindowOrg, XRF 253
    x- and y-extents, CDC::SetWindowExt, XRF 252
  setting graphics, _setwindow function, LIB 702–705
  simple classes, deriving from CDialog class
    Foundation classes cookbook, XUG 333
  static control, CStatic class described, XRF 562
  style, returning, CWnd::GetStyle, XRF 698

Windows (*continued*)
  subclassed, original WndProc address, CWnd::GetSuperWndProcAddr, XRF 698
  system-modal, returning, CWnd::GetSysModalWindow, XRF 698
  text, _settextwindow function, LIB 687
  validating client area, CWnd::ValidateRgn, XRF 820
  visibility, determining, CWnd::IsWindowVisible, XRF 710
  writing to, QuickWin, PT 159–160
Windows applications
  accessing command-line arguments entered at start, CWinApp::m_lpCmdLine, XRF 639
  cleaning up at termination, CWinApp::ExitInstance, XRF 631
  components
    Foundation classes, XUG 305
  constructor, CWinApp::CWinApp, XRF 631
  customizing
    Foundation classes tutorial, XUG 112–113
  CWinApp class
    described, XRF 628
    Foundation classes cookbook, XUG 307
  data interface, simplifying, XUG 122–134
  database, Phone Book sample program, XUG 117
  debugging, XUG 11
  default window procedure, calling
    Foundation classes cookbook, XUG 319–320
  designing
    Foundation classes, using, XUG 305–310
  developing
    HELLO sample program, XUG 81
  dialog boxes, standard
    (list), XUG 167
    described, XUG 167
  DOS applications, converting from. *See* QuickWin
  extra bytes, adding
    Foundation classes cookbook, XUG 327
  handle
    to current instance, CWinApp::m_hInstance, XRF 639
    to previous instance, CWinApp::m_hPrevInstance, XRF 639
  HELLO sample program, XUG 81–114
  idle loop processing
    Foundation classes cookbook, XUG 309–310
  idle-time processing, CWinApp::OnIdle, XRF 637
  initializing
    Foundation classes cookbook, XUG 307–308
  instance initializing, CWinApp::InitInstance, XRF 632

Windows applications (*continued*)
loading cursor resources
CWinApp::LoadCursor, XRF 633
CWinApp::LoadOEMCursor, XRF 634
CWinApp::LoadStandardCursor, XRF 635
loading icon resources
CWinApp::LoadIcon, XRF 633
CWinApp::LoadOEMIcon, XRF 634
CWinApp::LoadStandardIcon, XRF 636
making main window visible, CWinApp::m_nCmdShow, XRF 640
messages
creating and displaying, CWnd::MessageBox, XRF 711–714
filtering, CWinApp::PreTranslateMessage, XRF 638
last retrieved, CWinApp::m_msgCur, XRF 639
providing default loop, CWinApp::Run, XRF 638
name, CWinApp::m_pszAppName, XRF 640
one-time initializing, CWinApp::InitApplication, XRF 632
overridable member functions, XRF 11
optimizing precautions, PT 28
Phone Book sample program
message handlers, XUG 197
resource files
Foundation classes cookbook, XUG 310
running, XUG 14
storing pointer to main window object, CWinApp::m_pMainWnd, XRF 640
writing
Foundation classes, using, XUG 305–306
yielding
QuickWin, PT 164
Windows classes
cbWndExtra
Foundation classes cookbook, XUG 327
CDialog
Foundation classes cookbook, XUG 332
CModalDialog
Foundation classes cookbook, XUG 329
creating, using mouse button, XUG 319

Windows classes (*continued*)
CWnd
Foundation classes cookbook, XUG 345
derived
constructors for, XUG 312
overriding, XUG 320
deriving, XUG 91
icons, changing
Foundation classes cookbook, XUG 326
MDI
child windows, deallocation memory, XUG 323
Foundation classes cookbook, XUG 323
member variables
extra bytes, adding, XUG 327
preregistered, XUG 92
registration
attributes, changing, XUG 325
attributes, passing on, XUG 326
described, XUG 325
Foundation classes tutorial, XUG 112
key attributes, XUG 325
Windows Dereference Global Handle command, CodeView, ET 377, 386, 423
Windows Dereference Local Handle command, CodeView, ET 377, 386, 423
Windows Display Global Heap command, CodeView, ET 377, 382, 423, 462–463
Windows Display Local Heap command, CodeView, ET 377, 383, 423, 462
Windows Display Modules command, CodeView, ET 377, 383, 423, 463
Windows edit control
creating and attaching to CEdit object, CEdit::Create, XRF 286–289
creating, CEdit class, XRF 282
current selection
clearing, CEdit::Clear, XRF 285
copying, CEdit::Copy, XRF 286
cutting, deleting, CEdit::Cut, XRF 289
described, XRF 282
undo flags, resetting, CEdit::EmptyUndoBuffer, XRF 290

Windows File Manager
  starting PWB, ET 67
Windows functions
  entry/exit codes, customizing, ET 515
Windows graphics
  DECLARE_MESSAGE_MAP macro
    Foundation classes cookbook, XUG 343
  device contexts
    Foundation classes cookbook, XUG 345
  objects
    described, XUG 346
  ON_WM_PAINT macro
    Foundation classes cookbook, XUG 343
  paint message
    Foundation classes cookbook, XUG 343–344
  tools, Foundation classes equivalents, XUG 346
  WM_PAINT message
    Foundation classes cookbook, XUG 343–345, 349
Windows Kill Application command, CodeView, ET 385, 387–388, 423, 464–465
Windows menu
  CodeView, ET 373
Windows messages
  BN_CLICKED
    Foundation classes cookbook, XUG 316
  categories
    Foundation classes cookbook, XUG 313
  child
    differentiating between, XUG 317
  handling
    Foundation classes cookbook, XUG 313–319
  menu-command, XUG 314–315
  message maps, using
    Foundation classes cookbook, XUG 313
  scrolling
    Foundation classes cookbook, XUG 322
  traditional Windows responses, XUG 320
  WM_CHAR
    Foundation classes cookbook, XUG 319, 356
  WM_COMMAND
    Foundation classes cookbook, XUG 313–316
  WM_HSCROLL
    Foundation classes cookbook, XUG 322
  WM_INITDIALOG
    Foundation classes cookbook, XUG 330, 332
  WM_KEYDOWN
    Foundation classes cookbook, XUG 356
  WM_KEYUP
    Foundation classes cookbook, XUG 356
  WM_LBUTTONDOWN
    Foundation classes cookbook, XUG 351, 353
Windows messages (*continued*)
  WM_LBUTTONUP
    Foundation classes cookbook, XUG 356
  WM_MDIACTIVATE
    Foundation classes cookbook, XUG 324
  WM_MOUSEMOVE
    Foundation classes cookbook, XUG 353, 355
  WM_NCDESTROY
    Foundation classes cookbook, XUG 323
  WM_PAINT
    Foundation classes cookbook, XUG 343–345, 349
  WM_RTBUTTONDOWN
    Foundation classes cookbook, XUG 351, 353
  WM_VSCROLL
    Foundation classes cookbook, XUG 322
  WM_XXX
    Foundation classes cookbook, XUG 318
Windows Program Manager
  help, getting, ET 768–769
  starting, in PWB, ET 66
Windows windows
  attaching, to CWnd object, CWnd::Attach, XRF 659
  called to know maximized position of dimensions, CWnd::OnGetMinMaxInfo, XRF 740–741
  desktop, returning, CWnd::GetDesktopWindow, XRF 689
  destroying, CWnd::DestroyWindow, XRF 675
Windows, child
  accessing overridable member functions
    CFrameWnd::GetChildFrame, XRF 339
    CFrameWnd::GetParentFrame, XRF 340
  activating, next child, CMDIFrameWnd::MDINext, XRF 407
  active MDI, returning, CMDIFrameWnd::GetChildFrame, XRF 405
  arranging
    in tiled format, CMDIFrameWnd::MDITile, XRF 409
    minimized, CWnd::ArrangeIconicWindows, XRF 659
  buttons as, XRF 16
  called on activation or deactivation, CWnd::OnMDIActivate, XRF 753–754
  called on creation or destruction, CWnd::OnParentNotify, XRF 774–775
  called when about to be drawn, CWnd::OnCtlColor, XRF 726–727
  called when changing size or position, CWnd::OnChildActivate, XRF 720

Windows, child (*continued*)
  changing
    parent, CWnd::SetParent, XRF 810
    size, position, ordering, CWnd::SetWindowPos, XRF 814–816
  classes (list), XRF 6
  creating
    and attaching, CMDIChildWnd::Create, XRF 397
    attaching to CWnd object, XRF 664–665
    constructor, CMDIChildWnd::CMDIChildWnd, XRF 397
  CWnd, returning ID, CWnd::GetDlgCtrlID, XRF 689
  determining which contains specified point, CWnd::ChildWindowFromPoint, XRF 662–663
  handling activation message, CMDIChildWnd::MDIActivate, XRF 398
  identifying, CWnd::IsChild, XRF 709
  MDI
    activating, CMDIFrameWnd::MDIActivate, XRF 405
    arranging in cascade, CMDIFrameWnd::MDICascade, XRF 406
    client, handle for, CMDIFrameWnd::m_hWndMDIClient, XRF 410
    destroying, CMDIChildWnd::MDIDestroy, XRF 399
    maximizing, CMDIChildWnd::MDIMaximize, XRF 399
    maximizing, CMDIFrameWnd::MDIMaximize, XRF 407
    restoring, CMDIChildWnd::MDIRestore, XRF 399
    returning current, CMDIFrameWnd::MDIGetActive, XRF 406
    returning parent MDI frame, CMDIChildWnd::GetParentFrame, XRF 398
  minimized, arranging, CMDIFrameWnd::MDIIconArrange, XRF 406
  restoring, CMDIFrameWnd::MDIRestore, XRF 408
  searching for top-level, CWnd::GetTopWindow, XRF 700
Windows, frame
  creating, CMDIFrameWnd::CMDIFrameWnd, XRF 403
  replacing menu of MDI, CMDIFrameWnd::MDISetMenu, XRF 408–409
Windows, parent, retrieving, CWnd::GetParent, XRF 696
Windows, pop-up, associated with CWnd object, showing or hiding, CWnd::ShowOwnedPopups, XRF 817
_windowtype structures, presentation graphics, PT 223–224
_WINEXITNOPERSIST constant, PT 163
_WINEXITPERSIST constant, PT 163
_WINEXITPROMPT constant, PT 163
_WINFRAMEHAND constant, PT 160
WIN.INI, called after change made, CWnd::OnWinIniChange, XRF 798–799
Winlibs switch, PWB, ET 310–311
WinMain member function
  Foundation classes tutorial, XUG 89–90, 112–113
  message loop
    Foundation classes tutorial, XUG 112
  substituting personal version, XUG 89–90
  Windows applications, writing
    Foundation classes cookbook, XUG 305
_WINMAXREQ constant, PT 160
_WINOPERSIST constant, PT 162
_WINPERSIST constant, PT 162
_WINSIZECHAR constant, PT 158–160
_WINSIZEMAX constant, PT 158–160
_WINSIZEMIN constant, PT 158–160
_WINSIZERESTORE constant, PT 160
_WINSTATBAR constant, PT 163
Winstyle function, PWB, ET 154, 221
_WINTILE constant, PT 163
_WINVER constant, PT 158
WKA command, CodeView, ET 377, 385, 464–465
WL command, CodeView, ET 423, 465–466
WLH command, CodeView, ET 377, 386, 423, 466–467

Key				
	ET	Environment and Tools	PT	Programming Techniques
	LIB	Run-Time Library Reference	TUT	C++ Tutorial
	LR	C Language Reference	XRF	Class Libraries Reference
	LR+	C++ Language Reference	XUG	Class Libraries User's Guide

WM_CHANGECBCHAIN message
  CWnd::OnChangeCbChain message handler, XRF 718
  CWnd::SetClipboardViewer, XRF 807
WM_CHAR message
  CWnd::OnCharToItem message handler, XRF 719
  CWnd::OnGetDlgCode message handler, XRF 739
  Foundation classes cookbook, XUG 319, 356
WM_CHARTOITEM message
  CListBox::Create, XRF 359
WM_CHILDACTIVATE message
  CWnd::OnChildActivate message handler, XRF 720
WM_COMMAND message
  CButton::Create, XRF 130
  CWnd::OnCommand message handler, XRF 721
  CWnd::OnSysCommand message handler, XRF 790
  Foundation classes cookbook, XUG 313–316
  Foundation classes tutorial, XUG 98–99
  naming conventions
    Foundation classes tutorial, XUG 204
WM_COMPAREITEM message
  CWnd::OnCompareItem message handler, XRF 722
WM_CREATE message
  CButton::Create, XRF 128
  CComboBox::Create, XRF 144
  CEdit::Create, XRF 286
  CListBox::Create, XRF 356
  CWnd::CreateEx, XRF 667
  CWnd::OnNcCreate message handler, XRF 762
  Phone Book sample program, XUG 198, 224–226
WM_CTLCOLOR message
  CStatic::Create, XRF 565
  CWnd::OnCtlColor message handler, XRF 727
WM_DESTROY message
  CWnd::DestroyWindow, XRF 675
  CWnd::SetClipboardViewer, XRF 807
WM_DESTROYCLIPBOARD message
  CWnd::OnDestroyClipboard message handler, XRF 731
WM_DEVMODECHANGE message
  CWnd::OnDevModeChange message handler, XRF 731
WM_DRAWCLIPBOARD message
  CWnd::OnDrawClipboard message handler, XRF 732
  CWnd::SetClipboardViewer, XRF 807
WM_DRAWITEM message
  CButton::Create, XRF 130
  CMenu::AppendMenu, XRF 417
WM_ENABLE message
  CWnd::EnableWindow, XRF 681
WM_ENDSESSION message
  CWnd::OnQueryEndSession message handler, XRF 776
WM_ENTERIDLE message
  CWnd::CreateEx, XRF 668
WM_ERASEBKGND message
  CWnd::GetUpdateRect, XRF 700
  CWnd::OnEraseBkgnd message handler, XRF 738
  CWnd::OnIconEraseBkgnd message handler, XRF 743
WM_FONTCHANGE message
  CWnd::OnFontChange message handler, XRF 738
WM_GETDLGCODE message
  CDialog::IsDialogMessage, XRF 269
  CWnd::OnGetDlgCode message handler, XRF 739
WM_GETMINMAXINFO message
  CButton::Create, XRF 128
  CComboBox::Create, XRF 144
  CEdit::Create, XRF 286
  CListBox::Create, XRF 356
  CWnd::CreateEx, XRF 667
WM_GETTEXT message
  CEdit::FmtLines, XRF 290
  CWnd::GetDlgItemInt, XRF 690
WM_HSCROLL message
  Foundation classes cookbook, XUG 322
  Phone Book sample program, XUG 198
WM_INITDIALOG message
  CDialog::Create, XRF 265
  CDialog::CreateIndirect, XRF 266
  CDialog::OnInitDialog message handler, XRF 271
  CWnd::OnMeasureItem message handler, XRF 755
  Foundation classes cookbook, XUG 330–332
WM_INITMENU message
  CWnd::GetSystemMenu, XRF 699
WM_KEYDOWN message
  CWnd::OnSysKeyUp message handler, XRF 794
  CWnd::OnVKeyToItem message handler, XRF 795
  Foundation classes cookbook, XUG 356
WM_KEYUP message
  CWnd::OnSysKeyUp message handler, XRF 794
  Foundation classes cookbook, XUG 356
WM_KILLFOCUS message
  CWnd::SetFocus, XRF 809
WM_LBUTTONDBLCLK message
  CWnd::OnLButtonDblClk message handler, XRF 748
  Phone Book sample program, XUG 198

WM_LBUTTONDOWN message
CWnd::OnLButtonDblClk message handler, XRF 748
Foundation classes cookbook, XUG 351, 353
Phone Book sample program, XUG 198
WM_LBUTTONUP message
Foundation classes cookbook, XUG 356
WM_MBUTTONDBLCLK message
CWnd::OnMButtonDblClk message handler, XRF 751
WM_MBUTTONDOWN message
CWnd::OnMButtonDblClk message handler, XRF 751
WM_MDIACTIVATE message
CMDIFrameWnd::MDIActivate, XRF 405
Foundation classes cookbook, XUG 324
WM_MEASUREITEM message
CButton::Create, XRF 130
CMenu::AppendMenu, XRF 417
WM_MENUCHAR message
CWnd::OnMenuChar message handler, XRF 757
WM_MOUSEACTIVATE message
CWnd::OnMouseActivate message handler, XRF 759
WM_MOUSEMOVE message
CWnd::OnMouseMove message handler, XRF 760
Foundation classes cookbook, XUG 353, 355
WM_NCACTIVATE message
CMDIFrameWnd::MDIActivate, XRF 405
CWnd::OnMDIActivate message handler, XRF 754
CWnd::OnNcActivate message handler, XRF 761
WM_NCCALCSIZE message
CButton::Create, XRF 128
CComboBox::Create, XRF 144
CEdit::Create, XRF 286
CListBox::Create, XRF 356
CWnd::CreateEx, XRF 667
WM_NCCREATE message
CButton::Create, XRF 128
CComboBox::Create, XRF 144
CEdit::Create, XRF 286
CListBox::Create, XRF 356
WM_NCCREATE message (*continued*)
CWnd::CreateEx, XRF 667
CWnd::OnNcCreate message handler, XRF 762
WM_NCDESTROY message
CWnd::DestroyWindow, XRF 675
CWnd::OnNcDestroy message handler, XRF 763
Foundation classes cookbook, XUG 323
WM_NCHITTEST message
CWnd::OnNcHitTest message handler, XRF 764
WM_ONERASEBKGND message
CWnd::BeginPaint, XRF 660
WM_OTHERWINDOWDESTROYED message
CWnd::DestroyWindow, XRF 675
WM_PAINT message
CWnd::BeginPaint, XRF 660
CWnd::OnPaint message handler, XRF 772
CWnd::ScrollWindow, XRF 803
CWnd::ValidateRect, XRF 820
Foundation classes cookbook, XUG 343–345, 349
Foundation classes tutorial, XUG 88
Phone Book sample program, XUG 198, 235–237, 241
WM_PARENTNOTIFY message
CWnd::DestroyWindow, XRF 675
WM_QUERYDRAGICON message
CWnd::OnQueryDragIcon message handler, XRF 775
WM_QUERYENDSESSION message
CWnd::OnQueryEndSession message handler, XRF 776
WM_QUERYNEWPALETTE message
CWnd::OnQueryNewPalette message handler, XRF 777
WM_QUERYOPEN message
CWnd::OnQueryOpen message handler, XRF 777
WM_QUIT message
CFrameWnd::~CFrameWnd, XRF 338
CWinApp::Run, XRF 638
WM_RBUTTONDBLCLK message
CWnd::OnRButtonDblClk message handler, XRF 778

WM_RBUTTONDOWN message
  CWnd::OnRButtonDblClk message handler, XRF 778
  Foundation classes cookbook, XUG 351, 353
WM_SETCURSOR message
  CWnd::OnSetCursor message handler, XRF 782
WM_SETFOCUS message
  CWnd::SetFocus, XRF 809
WM_SETFONT message
  CDialog::Create, XRF 265
  CDialog::CreateIndirect, XRF 266
  CDialog::OnSetFont message handler, XRF 271
WM_SETREDRAW message
  CListBox::Create, XRF 358
WM_SIZE message
  Phone Book sample program, XUG 198, 225–226
WM_SYSCHAR message
  CWnd::SetFocus, XRF 809
WM_SYSCOMMAND message
  CWnd::GetSystemMenu, XRF 699
  CWnd::OnCommand message handler, XRF 721
  CWnd::OnNcLButtonDblClk message handler, XRF 765
  CWnd::OnNcLButtonDown message handler, XRF 765
  CWnd::OnNcLButtonUp message handler, XRF 766
  CWnd::OnNcMouseMove message handler, XRF 769
  CWnd::OnSysCommand message handler, XRF 789
WM_SYSKEYDOWN message
  CWnd::OnSysChar message handler, XRF 787
  CWnd::OnSysKeyDown message handler, XRF 792
  CWnd::SetFocus, XRF 809
WM_SYSKEYUP message
  CWnd::OnSysChar message handler, XRF 787
  CWnd::OnSysKeyDown message handler, XRF 792
  CWnd::OnSysKeyUp message handler, XRF 793
  CWnd::SetFocus, XRF 809
WM_TIMECHANGE message
  CWnd::OnTimeChange message handler, XRF 794
WM_TIMER message
  CWnd::KillTimer, XRF 711
  CWnd::SetTimer, XRF 814
WM_VKEYTOITEM message
  CListBox::Create, XRF 359
  CWnd::OnVKeyToItem message handler, XRF 795
WM_VSCROLL message
  Foundation classes cookbook, XUG 322
  Phone Book sample program, XUG 198
WM_WININICHANGE message
  CWnd::OnWinIniChange message handler, XRF 798
WM_XXX messages
  Foundation classes cookbook, XUG 318
  Foundation classes tutorial, XUG 99
_wmenuclick function, LIB 880–881; PT 152–153, 163–164
wndBottom static data member
  CWnd class, XRF 822
wndTop static data member
  CWnd class, XRF 822
WO operator
  CodeView, ET 405, 414–415
_wopen function, LIB 882–884; PT 151, 157–158
_wopeninfo struct, PT 158–159
Word processor
  formatting HELPMAKE text, ET 726
  rich text format, HELPMAKE, ET 725–726
Word switch, PWB, ET 265, 307–308
Word wrapping
  controlling, _wrapon function, LIB 885–886
  PWB switches, ET 265, 293, 308
Words
  16-bit, CWordArray class described, XRF 823
  English, regular expression syntax, ET 778, 780, 785
  finding in CodeView, ET 361
  inputting from port, _inp and _inpw functions, LIB 428
  outputting at port, _outp and _outpw functions, LIB 542–544
  wrap controlling, _wrapon function, LIB 885–886
Wordwrap switch, PWB, ET 265, 308
_wrapon function, LIB 885–886
_write function, LIB 887–888
Write member function
  CArchive class, XRF 100
  CFile class
    Foundation classes cookbook, XUG 278
write member function
  ofstream class
    iostream classes tutorial, XUG 375–376
  CFile class, XRF 320
    CStdioFile::WriteString, XRF 570
  ostream class, XRF 905
WriteObject member function
  CArchive class, XRF 101
    CObject::Serialize, XRF 473
writeOnly member function
  CArchiveException class
    CArchiveException::m_cause, XRF 106
WriteString member function
  CStdioFile class, XRF 570

- Writing
  - bytes to streams, ostream::write, XRF 905
  - character strings to regions, CDC::ExtTextOut, XRF 188–189
  - characters
    - to console, _putch function, LIB 595–596
    - to streams, fputc and _fputchar functions, LIB 301–302
    - to streams, putc and putchar functions, LIB 593–594
  - data
    - to files, _write function, LIB 887–888
    - to streams, fwrite function, LIB 338–339
    - to strings, sprintf function, LIB 725–726
  - data from buffer
    - to CFile object-associated file, CFile::Write, XRF 320
    - to file associated with CStdioFile object, CStdioFile::WriteString, XRF 570
  - formatted output to argument lists, vfprintf, vprintf and vsprintf functions, LIB 840–843
  - integers to streams, _putw function, LIB 603–604
  - object to archive, CObject::Serialize, XRF 473–474
  - strings
    - to output, puts function, LIB 602
    - to streams, fputs function, LIB 303
    - to the console, _cputs function, LIB 167
  - text
    - horizontally on screen, _pg_hlabelchart function, LIB 569
    - vertically on screen, _pg_vlabelchart function, LIB 576
  - to archives, CArchive::Write, XRF 100
- Writing functions, inline assembly code, PT 118–119
- _wsetexit function, LIB 889–891
  - QuickWin windows, PT 148, 163
- _wsetfocus function, LIB 892–893; PT 152, 161–162
- _wsetscreenbuf function, LIB 894–895; PT 153, 160–161
- _wsetsize function, LIB 896–897; PT 153, 160
- _wsizeinfo struct, PT 158–159
- _wxycoord structure, PT 185
- _wyield function, LIB 898–899; PT 164

X

- X command, CodeView, ET 423, 467–468
- :x command, HELPMAKE, ET 722
- /X option
  - CL, ET 545
  - NMAKE, ET 650
- /x option
  - LINK, ET 562, 591, 594
- xalloc member function
  - ios class, XRF 865
- XENIX compatibility, LIB ix
- \xhhh (escape sequence), hexadecimal notation, LR 18
- XMS
  - *See also Getting Started*; Extended memory; Memory
  - defined, ET 818
  - Keepmem switch, PWB, ET 285
- XMS server defined, ET 818
- _XRES256COLOR constant, PT 170
- _xycoord structure, PT 185

Y

- :y command, HELPMAKE, ET 722
- _y0 function, LIB 103–105
- _y0l function, LIB 103–105
- _y1 function, LIB 103–105
- _y1l function, LIB 103–105
- /Yc option, CL, ET 546–548; LR 261; PT 34–35
- /Yd option, CL, ET 546, 548–549; LR 261; PT 38–39
- Years
  - getting, CTime::GetYear, XRF 614
- YELLOW.C sample graphics program, PT 178
- Yielding processing time
  - QuickWin applications, PT 164
- _yn function, LIB 103–105
- _ynl function, LIB 103–105

/Yu option
CL, ET 546, 549–550; LR 261; PT 35–36
Yellow
color value, ET 273

Z

:z command, HELPMAKE, ET 722
/Za option
CL, ET 550–552; LR 3–5, 169, 261, 265
/Zc option
CL, ET 552; LR 7, 235
/Zd option
CL, ET 324, 409, 553
/Ze option
CL, ET 550–552; LR 199, 258, 261, 265
Zero values, C++, conversion to null pointer, LR+ 71, 75
Zero-length files, LR 250
/Zf option
CL, ET 555; LR 261
/Zg option
CL, ET 552–553
/Zi option
CL, ET 324, 409, 553; PT 40
/Zl option
CL, ET 553–554
/Zn option
BSCMAKE, ET 732
CL, ET 555–556; LR 261
/Zp option
CL, ET 554–555; LR 70, 212
Zp2 option
CL, LR 243
/Zr option
CL, ET 556–557; LR 210
_ZRES16COLOR constant, PT 170
_ZRES256COLOR constant, PT 170
/Zs option
BSCMAKE, ET 732
CL, ET 557

Error Messages

Part 2

Overview 277

Overview

This section lists Microsoft C/C++ error and warning messages. Each message includes an explanation of what went wrong and what action to take to correct the problem. Error messages can also display the input file and line number where the error occurred.

Error numbers consist of a one-, two-, or three-letter prefix and four digits. The first digit indicates the severity level:

- Fatal errors stop execution and are numbered 1*xxx*.
- Errors numbered 2*xxx* are usually nonfatal; execution continues if possible.
- Warnings do not stop execution but indicate a possible problem; they are numbered 4*xxx*.

In this section, error messages are arranged in alphanumeric order:

Error Number	**Tool**
BK*xxxx*	BSCMAKE
C*xxxx*	Compiler
CK*xxxx*	CVPACK
CV*xxxx* and CXX*xxxx*	CodeView
D*xxxx*	Command-Line
H*xxxx*	HELPMAKE
IM*xxxx*	IMPLIB
L*xxxx*	LINK
M*xxxx*	Floating-Point Math
MP*xxxx*	MPC
PWB*xxxx*	PWB
R*xxxx*	Run-Time
SB*xxxx*	SBRPACK
U*xxxx*	NMAKE, EXEHDR, and LIB

Note that the CV*xxxx* error messages are an exception and are listed in alphabetical order by message text.

BSCMAKE Error Messages

Microsoft Browser Database Maintenance Utility (BSCMAKE) generates the following error messages:

- Fatal errors (BK1*xxx*) cause BSCMAKE to stop execution.
- Warnings (BK4*xxx*) indicate possible problems in the database-building process.

BSCMAKE Fatal Error Messages

Number **BSCMAKE Fatal Error Message**

BK1500 **UNKNOWN ERROR**
Contact Microsoft Product Support Services

BSCMAKE detected an unknown error condition.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

BK1501 **unknown character** *character* **in option** *option*

BSCMAKE did not recognize the given character specified for the given option.

BK1502 **incomplete specification for option** *option*

The given option did not contain the correct syntax.

BK1503 **cannot write to file** *filename*

BSCMAKE could not write to the given file.

One of the following may have occurred:

- The disk was full.
- A hardware error occurred.

BK1504 **cannot position in file** *filename*

BSCMAKE could not move to a location in the given file.

One of the following may have occurred:

- The disk was full.
- A hardware error occurred.
- The file was truncated. Truncation can occur if the compiler runs out of disk space or is interrupted when it is creating the .SBR file.

BK1505 **cannot read from file** *filename*

BSCMAKE could not read from the given file.

One of the following may have occurred:

- The file was corrupt.
- The file was truncated. Truncation can occur if the compiler runs out of disk space or is interrupted when it is creating the .SBR file.

BK1506 **cannot open file** *filename*

BSCMAKE could not open the given file.

One of the following may have occurred:

- No more file handles were available. Increase the number of file handles by changing the FILES setting in CONFIG.SYS to allow a larger number of open files. FILES=50 is the recommended setting.
- The file was locked by another process.
- The disk was full.
- A hardware error occurred.
- The specified output file had the same name as an existing subdirectory.

BK1507 **cannot open temporary file** *filename*

BSCMAKE could not open one of its temporary files.

One of the following may have occurred:

- No more file handles were available. Increase the number of file handles by changing the FILES setting in CONFIG.SYS to allow a larger number of open files. FILES=50 is the recommended setting.
- The TMP environment variable was not set to a valid drive and directory.
- The disk was full.

BK1508 **cannot delete temporary file** *filename*

BSCMAKE could not delete one of its temporary files.

One of the following may have occurred:

- Another process had the file open.
- A hardware error occurred.

BK1509 **out of heap space**

BSCMAKE ran out of memory.

One of the following may be a solution:

- Reduce the memory that BSCMAKE will require by using one or more options. Use /Ei or /Es to eliminate some input files. Use /Em to eliminate macro bodies.
- Run BSCMAKE (or PWB if you are building a database in PWB) in a DOS session under Windows to use virtual memory provided under Windows.
- Free some memory by removing terminate-and-stay-resident (TSR) software.
- Reconfigure the EMM driver.
- Change CONFIG.SYS to specify fewer buffers (the BUFFERS command) and fewer drives (the LASTDRIVE command).
- Run BSCMAKEV.EXE instead of BSCMAKE.EXE.

BK1510 corrupt .SBR file *filename*

The given .SBR file is corrupt or does not have the expected format.

Recompile to regenerate the .SBR file.

BK1511 invalid response file specification

BSCMAKE did not understand the command-line specification for the response file. The specification was probably wrong or incomplete.

For example, the following specification causes this error:

```
bscmake @
```

BK1512 database capacity exceeded

BSCMAKE could not build a database because the number of definitions, references, modules, or other information exceeded the limit for a database.

One of the following may be a solution:

- Exclude some information using the /Em, /Es, or /Ei option.
- Omit the /Iu option if it was used.
- Divide the list of .SBR files and build multiple databases.

BK1513 nonincremental update requires all .SBR files

An attempt was made to build a new database, but one or more of the specified .SBR files was truncated. This message is always preceded by warning BK4502, which will give the name of the .SBR file that caused the error.

BSCMAKE can process a truncated, or zero-length, .SBR file only when a database already exists and is being incrementally updated.

One of the following may be a cause:

- The database file was missing.
- The wrong database name was specified.
- The database was corrupted, and a full build was required.

BK1514 **all .SBR files truncated and not in database**

None of the .SBR files specified for an update was a part of the original database. This message is always preceded by warning BK4502, which will give the name of the .SBR file that caused the error.

One of the following may be a cause:

- The wrong database name was specified.
- The database was corrupted, and a full build was required.

BK1515 *bscfile* **: incompatible version; cannot incrementally update**

The given database (.BSC file) was not created with this version of BSCMAKE. A database can be incrementally built only by the same version of BSCMAKE as the one used to fully build the database.

BSCMAKE Warning Messages

Number **BSCMAKE Warning Message**

BK4500 **UNKNOWN WARNING**
Contact Microsoft Product Support Services

An unknown error condition was detected by BSCMAKE.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

BK4501 **ignoring unknown option** *option*

BSCMAKE did not recognize the given option and ignored it.

If the given option is /r, it must be specified first on the BSCMAKE command line.

BK4502 truncated .SBR file *filename* not in database

The given zero-length .SBR file, specified during a database update, was not originally part of the database.

If a zero-length file that is not part of the original build of the database is specified during a rebuild of that database, BSCMAKE issues this warning. One of the following may be a cause:

- The wrong database name was specified.
- The database was deleted. (Error BK1513 will result.)
- The database file was corrupted, requiring a full build.

BK4503 minor error in .SBR file *filename* ignored

The given .SBR file contained an error that did not halt the build. However, the resulting .BSC file may not be correct.

Recompile to regenerate the .SBR file.

Compiler Error Messages

The error messages produced by the compiler fall into three categories:

- Fatal error messages
- Error messages
- Warning messages

Compiler Fatal Error Messages

Fatal error messages indicate a severe problem, one that prevents the compiler from processing the program any further. These messages have the following format:

```
filename(line) : fatal error C1xxx: messagetext
```

After a compiler displays a fatal error message, it terminates without producing an object file or checking for further errors.

Compiler Error Messages

Error messages identify actual program errors. These messages have the following format:

```
filename(line) : error C2xxx: messagetext
```

The compiler does not produce an object file for a source file that has compiler errors. When the compiler encounters such errors, it attempts to recover from the error. If possible, it continues to process the source file and produce any additional error messages. If errors are too numerous or too severe, the compiler stops processing.

Compiler Warning Messages

Warning messages are informational only; they do not prevent compilation or linking. These messages have the following format:

```
filename(line) : warning C4xxx: messagetext
```

Compiler Fatal Error Messages

Number **Compiler Fatal Error Message**

C1000 **UNKNOWN FATAL ERROR**
Contact Microsoft Product Support Services

An unknown error condition was detected by the compiler.

Note the circumstances of the error, and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

C1001 **INTERNAL COMPILER ERROR**
(compiler file*filename***, line** *number***)**
Contact Microsoft Product Support Services

The compiler detected an internal inconsistency.

Note the circumstances of the error, and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals. Note the filename and line number where the error occurred; an internal file and line number are provided in addition to the file and line number of your source file.

C1002 **compiler is out of heap space in pass 2**

The compiler ran out of dynamic memory space during execution of the second pass of the compiler (C2.EXE). Usually this means the program has too many symbols and/or complex expressions.

One of the following may be a solution:

- Divide the file into several smaller source files.
- Break expressions into smaller subexpressions.
- Remove other programs or drivers running in the system that could be consuming significant amounts of memory.

C1003 **error count exceeds** ***number*****; stopping compilation**

Errors in the program were too numerous to allow recovery, and the compiler must terminate.

C1004 **unexpected end of file found**

The default disk drive did not contain sufficient space for compiler-generated temporary files. The space required is approximately two times the size of the source file.

This message also appears when an **#if** directive evaluates to false without a corresponding closing **#endif** directive.

C1005 **string too big for buffer**

A string in a compiler intermediate file overflowed a buffer.

C1006 **write error on compiler intermediate file**

The compiler was unable to create the intermediate files used in the compilation process.

This error can be caused by a disk media error or by an open floppy drive door.

C1007 **unrecognized flag** *string* **in** *option*

The string in the command-line option was not valid.

Check the CL command line and CL environment variable for option specifications.

C1008 **no input file specified**

The compiler was not given a file to compile.

The compiler must be given a C or C++ source file to compile. Check the CL command line and CL environment variable for filename specifications.

C1009 **compiler limit : macros nested too deeply**

Too many macros were being expanded at the same time.

This error occurs when a macro definition contains macros to be expanded and those macros contain other macros. The compiler has a limit of 256 levels of nested macros.

Try to split the nested macros into simpler macros.

C1011 **compiler limit :** *identifier* **: macro definition too big**

The macro definition was longer than allowed.

Try to split the definition into shorter definitions.

C1012 **unmatched parenthesis nesting : missing** *character*

The parentheses in a preprocessor directive were not matched. The missing character is either a left or right parenthesis.

C1013 **compiler limit : too many open parentheses**

Too many levels of parentheses were used.

Simplify the expression, or calculate part of the expression in a separate statement.

C1015 **compiler limit : too many segments**

Too many segments were opened.

This error occurs only with the /f or /qc fast-compilation options or with the /Oq p-code generation option.

There are several ways to correct this situation:

- Reduce the number of segments used by your program.
- Separate the source code into multiple files so that fewer segments are referenced in a single module.
- Recompile the program without the /f, /qc, or /Oq command-line option.

C1016 **#if[n]def expected an identifier**

The **#ifdef** or **#ifndef** conditional compilation directive was not supplied with an identifier to evaluate.

An identifier must be specified with the **#ifdef** and **#ifndef** directives.

C1017 **invalid integer constant expression**

The expression in an **#if** directive either did not exist or did not evaluate to a constant.

C1018 **unexpected #elif**

The **#elif** directive did not appear within an **#if**, **#ifdef**, or **#ifndef** construct.

Make sure that there is a **#if**, **#ifdef**, or **#ifndef** statement in effect before this statement.

C1019 **unexpected #else**

The **#else** directive did not appear within an **#if**, **#ifdef**, or **#ifndef** construct.

Make sure that there is a **#if**, **#ifdef**, or **#ifndef** statement in effect before this statement.

C1020 **unexpected #endif**

An **#endif** directive appeared without a matching **#if**, **#ifdef**, or **#ifndef** directive.

Make sure that there is a matching **#endif** for each **#if**, **#ifdef**, and **#ifndef** statement.

C1021 **invalid preprocessor command** *string*

The characters following the number sign (#) did not form a valid preprocessor directive.

The number sign cannot be used as the first character in an identifier.

C1022 **expected #endif**

An **#if**, **#ifdef**, or **#ifndef** directive was not terminated with an **#endif** directive.

Make sure that there is a **#if**, **#ifdef**, or **#ifndef** statement in effect before this statement.

C1023 **cannot open source file** *filename*

The given file either did not exist, could not be opened, or was not found.

Make sure that the environment settings are valid and that the correct path for the file is specified.

If this error appears without an error message, the compiler has run out of file handles. To increase the number of file handles available under DOS, change the FILES setting in CONFIG.SYS. FILES=50 is the recommended setting.

C1024 **cannot open include file** *filename*

The specified file in an **#include** preprocessor directive could not be found.

Make sure that the settings for the INCLUDE and TMP environment variables are valid and that the correct path for the file is specified.

If this error appears without an error message, the compiler has run out of file handles. To increase the number of file handles available under DOS, change the FILES setting in CONFIG.SYS. FILES=50 is the recommended setting.

C1026 **parser stack overflow : program too complex**

The program could not be processed because the space required to parse the program caused a stack overflow in the compiler.

Simplify the program by decreasing the complexity of expressions. Decrease the level of nesting in **for** and **switch** statements by putting some of the more deeply nested statements in separate functions. Break up very long expressions involving comma operators or function calls.

C1027 **DGROUP data allocation exceeds 64K**

More than 64K of variables were allocated to the default data segment.

For compact-, large-, or huge-model programs, use the /Gt option to move items into separate segments. In small- or medium-model programs, consider explicitly allocating some variables outside of DGROUP by using **__based** or **__far**.

This limit does not exist when compiling programs for 32-bit operating systems, such as the Microsoft DOS Extender.

C1031 **compiler limit : function calls nested too deeply**

The program exceeded the dynamic compiler limit on nested function calls.

Split the nested calls, saving the return value from one of the nested functions in a temporary variable.

C1032 **cannot open object code listing file** *filename*

The output listing file specified with the /Fl command-line option could not be opened.

There are several possible causes for this error:

- The given name is not valid.
- The file cannot be opened because of a lack of space.
- A read-only file with the given name already exists.
- The file is in use by another process.

C1033 **cannot open assembly language output file** *filename*

The output listing file specified with the /Fc or /Fa command-line option could not be opened.

There are several possible causes for this error:

- The given name is not valid.
- The file cannot be opened because of a lack of space.
- A read-only file with the given name already exists.
- The file is in use by another process.

C1035 **expression too complex; simplify expression**

The compiler was unable to generate code for a complex expression.

Try to split the expression into simpler subexpressions and recompile.

C1036 **cannot open source listing file** *filename*

The output listing file specified with the /Fs command-line option could not be opened.

There are several possible causes for this error:

- The given name is not valid.
- The file cannot be opened because of a lack of space.
- A read-only file with the given name already exists.
- The file is in use by another process.

C1037 **cannot open object file** *filename*

The object file specified with the /Fo command-line option could not be opened.

There are several possible causes for this error:

- The given name is not valid.
- The file cannot be opened because of a lack of space.
- A read-only file with the given name already exists.
- The file is in use by another process.

C1039 **unrecoverable heap overflow in pass 3**

The postoptimizer compiler pass overflowed the heap and could not continue.

One of the following may be a solution:

- Break up the function containing the line that caused the error.
- Recompile with the /Od option to remove optimization.
- Remove other programs or drivers running in the system that could be consuming significant amounts of memory.

C1040 **unexpected end-of-file condition in source file** *filename*

The compiler detected an unexpected end-of-file condition while creating a source listing or mixed source and object listing.

This error occurs if the source file is deleted or overwritten while it is being read.

C1041 **cannot open compiler intermediate file—no more files**

The compiler could not create intermediate files for use in the compilation process because no more file handles were available.

To increase the number of file handles available under DOS, change the FILES setting in CONFIG.SYS. FILES=50 is the recommended setting.

C1042 **cannot open compiler intermediate file—no such file or directory**

The compiler could not create intermediate files for use in the compilation process because the TMP environment variable was set to an invalid directory or path.

Use the SET command to change the TMP environment variable so that it points to a valid directory.

C1043 **cannot open compiler intermediate file**

The compiler could not create intermediate files for use in the compilation process. The exact reason could not be determined.

One of the following may be a solution:

- Make sure that the environment variable TMP points to a drive and directory in which a file can be created.
- Delete unneeded files in the TMP directory.

C1044 **out of disk space for compiler intermediate file**

The compiler could not create intermediate files for use in the compilation process because no more space was available.

Make more space available on the disk pointed to by the TMP environment variable and then recompile.

C1045 **compiler limit : linkage specifications nested too deeply**

The nesting of externals exceeded the capacity of the compiler. Nested externals are allowed when specifying the external linkage type (such as `extern "C++"`).

Make sure that nested externals have appropriate closing braces.

C1046 **compiler limit :** ***structure*** **nested too deeply**

The given structure, union, or class exceeded the nesting limit of the compiler.

Structures, unions, and classes cannot be nested to more than 15 levels.

Rewrite the definition so that fewer structures, unions, or classes are nested. The structure, union, or class can be split into two or more parts by defining one or more of the nested structures using **typedef**.

C1047 **limit of** ***option*** **exceeded at** ***string***

The given option was specified too many times. The given string is the argument to the option that caused the error.

Check the CL environment variable for additional occurrences of the given command-line option.

C1048 **unknown option** *character* **in** *option*

The given character was not a valid letter for the option.

For example, the following line:

```
#pragma optimize( "*", on )
```

causes the following error:

```
unknown option '*' in '#pragma optimize'
```

C1049 **invalid numerical argument** *string*

The compiler expected a numerical argument but received the given string.

This error may be caused by giving a hexadecimal number without the necessary **\x** prefix or by a misformed floating-point number.

C1050 *segment* **: code segment too large**

A code segment grew to within 36 bytes of 64K during compilation of a 16-bit program.

To avoid this error, choose a memory model that allows multiple code segments, such as medium, large, or huge. This error can also be avoided by using the /Gy command-line option to have the compiler generate packaged functions.

A 36-byte pad is used because of a bug in some 80286 chips that can cause programs to exhibit strange behavior when, among other conditions, the size of a code segment is within 36 bytes of 64K.

C1052 **compiler limit : #if or #ifdef nested too deeply**

The program exceeded the maximum of 32 nesting levels for **#if** and **#ifdef** directives.

This error can be caused by include files that use these preprocessor directives.

C1054 **compiler limit : initializers nested too deeply**

The compiler limit on the nesting of initializers was exceeded. The limit depends on the combination of types being initialized and may range from 10 to 15 levels.

Simplify the data types being initialized to reduce the levels of nesting, or assign initial values in separate statements after the declaration.

C1055 **compiler limit : out of keys**

The file being compiled contained too many symbols.

Try to split the file into smaller files, and compile them separately.

C1056 **compiler limit : out of macro expansion space**

The compiler overflowed an internal buffer during the expansion of a macro.

Try to split the macros into simpler macros or remove nonessential space and tab characters from macro definitions that were used in the expansion.

C1057 **unexpected end of file in macro expansion**

The compiler has encountered the end of the source file while gathering the arguments of a macro invocation. Usually this is the result of a missing right parenthesis in the macro invocation.

C1058 **compiler limit : too many formal arguments**

The function declaration had too many formal arguments.

Reduce the number of arguments by passing structures or pointers to structures.

C1059 **compiler is out of near heap space**

The compiler ran out of storage for items that it stores in its near (default data segment) heap.

One of the following may be a solution:

- Eliminate unnecessary include files, especially unneeded function prototypes.
- Split the function at the given line number into two or more functions.
- Split the current file into two or more files and compile them separately.

C1060 **compiler is out of far heap space**

The compiler ran out of storage for items that it stores in its far heap. Usually this is the result of having too many symbols.

One of the following may be a solution:

- Eliminate unnecessary include files, especially unneeded **#defines** and function prototypes.
- Eliminate some global variables.
- Split the current file into two or more files and compile them separately.
- Remove other programs or drivers running in the system, which could be consuming significant amounts of memory.

C1061 **compiler limit : blocks nested too deeply**

Nested blocks in the program exceeded the nesting limit allowed by the compiler.

This error occurs only with the /f or /qc fast-compilation option.

Rewrite the program, putting one or more nested blocks into a separate function, or recompile the program without the /f or /qc option.

C1062 **error while writing to preprocessor output file**

The compilation command included the /P option to produce a preprocessor output file, but not enough disk space was available to hold the file.

Free more space on the destination drive or choose an alternate output drive.

C1063 **compiler limit : compiler stack overflow**

The program was too complex and caused the compiler stack to overflow.

Simplify the program, making it more modular, and recompile. If the /f or /qc fast-compilation command-line option is being used, recompile without it.

C1064 **compiler limit : token overflowed internal buffer**

The compiler read an identifier that was longer than the internal buffer used for identifier names.

Shorten the name and recompile.

C1065 **compiler limit : out of tags**

The file being compiled contained too many symbols.

Try to split the file into smaller files, and compile them separately.

C1066 **compiler limit : decorated name length exceeded**

After the symbol's name was made unique ("decorated"), it became too long. The maximum length of a decorated C++ name is 247 characters.

Shorten the name of the symbol.

C1068 **cannot open file** *filename*

The given file either did not exist, could not be opened, or was not found.

Make sure that the environment settings are valid and that the correct path for the file is specified.

If this error appears without an error message, the compiler has run out of file handles. To increase the number of file handles available under DOS, change the FILES setting in CONFIG.SYS. FILES=50 is the recommended setting.

C1069 **write error on file** *filename*

An error occurred while the compiler was trying to write to the file. One possible cause of this error is insufficient disk space.

C1070 mismatched #if/#endif pair in file *filename*

The preprocessor found the **#if**, **#ifdef**, or **#ifndef** directive but did not find a corresponding **#endif** directive in the same source file.

C1071 unexpected end of file found in comment

The compiler found the end of a file while scanning a comment.

Probably a comment was not terminated. Begin at the end of the file and search backward for the beginning of a comment. A comment begins with **/*** and ends with ***/** as in:

```
/* This is a
comment */
```

A comment cannot be split across source files.

C1072 *filename* : cannot read file

The compiler encountered an error when trying to read a file.

This error can be caused by a disk error or by a file-sharing conflict.

C1090 *segment* data allocation exceeds 64K

The size of the named segment exceeded 64K.

In a 16-bit program, segments cannot exceed 64K. This error occurs with **__based** allocation.

Choose a segment that has more space available, or use the huge memory model.

C1126 *identifier* : automatic allocation exceeds *size*

The space allocated for the local variables of a function exceeded the given limit.

Use the **malloc** run-time function or the **new** function to allocate large amounts of space for data.

C1127 *segment* : segment redefinition

A segment was overwritten by another segment with the same name.

For example, compiling in large model with:

```
#pragma alloc_text( _TEXT, func1 )
```

creates two segments, the default segment `module_TEXT` and the specified segment `_TEXT`. However, in small model, the default segment is `_TEXT`, and the specified segment `_TEXT` will overwrite the default segment.

C1500 *filename* **: cannot open inline function definition file**

The given file either did not exist, could not be opened, or was not found.

Make sure that the environment settings are valid and that the correct path for the file is specified.

If this error appears without an error message, the compiler has run out of file handles. To increase the number of file handles available under DOS, change the FILES setting in CONFIG.SYS. FILES=50 is the recommended setting.

C1501 compiler limit : too many temporary variables

The source file contained more than 17,500 temporary variables.

Split the file into smaller parts.

C1502 inline member-function definition missing '}'

The compiler reached the end of the file and did not find a matching closing brace.

Make sure that curly braces are matched.

C1503 default parameter definition missing ',' or ')'

After one or more default parameters were defined, neither a comma to indicate continuation nor a closing parenthesis to indicate termination of the function declaration was found.

Make sure that there are matching parentheses around the set of formal parameters.

C1504 type ambiguous after *number* **tokens**

The compiler could not resolve the type after looking ahead in the code.

Simplify the code to make the statement clearer. This error can often be eliminated by using an explicit type cast to an ambiguous expression.

C1505 unrecoverable parser lookahead error

The compiler could not evaluate the code.

Simplify the code by making smaller classes or functions.

C1506 unrecoverable block scoping error

The block was too large to compile.

This error can be caused by mismatched curly braces or by an extremely large function or class.

Compiler Error Messages

Number Compiler Error Message

C2000 UNKNOWN ERROR
Contact Microsoft Product Support Services

The compiler detected an unknown error condition.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

C2001 newline in constant

A string constant was continued on a second line without either a backslash (\) or closing and opening double quotation marks (").

To break a string constant that is on two lines in the source file, do one of the following:

- End the first line with the line-continuation character, a backslash.
- Close the string on the first line with a double quotation mark, and open the string on the next line with another quotation mark.

It is not sufficient to end the first line with **\n**, the escape sequence for embedding a newline character in a string constant.

The following are examples of incorrect and correct usage:

```
printf("Hello,              // error
world");

printf("Hello,\n            //  error
world");

printf("Hello,\             //  OK
world");

printf("Hello,"             // OK
" world");
```

Note that any spaces at the beginning of the next line after a line-continuation character are included in the string constant and that neither solution actually places a newline character into the string constant. The following examples embed this character:

```
printf("Hello,\n\
world");

printf("Hello,\
\nworld");

printf("Hello,\n"
"world");

printf("Hello,"
"\nworld");
```

C2002 invalid wide-character constant

The use of the multibyte-character constant was not legal.

This error can be caused if a wide character contains more bytes than expected.

Wide characters cannot be concatenated with ordinary string literals and cannot be used if the standard header, STDDEF.H, is not included.

Wide-character strings and constants must be preceded by the character L.

The following example shows a wide-character constant:

```
L'mbconst'
```

Wide characters are not supported in C.

C2003 expected defined id

An identifier was expected after the specified preprocessing keyword.

C2004 expected defined(id)

An identifier was expected after the left parenthesis following the specified preprocessing keyword.

C2005 #line expected a line number, found *token*

A **#line** directive lacked the required line-number specification.

C2006 #include expected a filename, found *token*

An **#include** directive lacked the required filename specification.

C2007 **#define syntax**

An identifier was expected following **#define** in a preprocessing directive.

C2008 *character* **: unexpected in macro definition**

The given character was found immediately following the name of the macro.

C2009 **reuse of macro formal** *identifier*

The given identifier was used more than once in the formal parameter list of a macro definition.

C2010 *character* **: unexpected in macro formal parameter list**

The given character was used incorrectly in the formal parameter list of a macro definition.

C2011 *identifier* **:** *type* **type redefinition**

The specified identifier was already defined as type *type*.

The following is an example of this error:

```
struct S;
union S;
```

C2012 **missing name following '<'**

An **#include** directive lacked the required filename specification.

C2013 **missing '>'**

The closing angle bracket (>) was missing from a **#include** directive.

C2014 **preprocessor command must start as first nonwhite-space**

Nonwhite-space characters appeared before the number sign (#) of a preprocessor directive on the same line.

C2015 **too many characters in constant**

A character constant contained more than two characters.

Character constants are limited to one character (standard charcter constants) or two characters (long character constants).

Note that an escape sequence (for example, **\t** for tab) is converted to a single character.

C2016 **no closing single quotation mark**

A newline character was found before the closing single quotation mark of a character constant.

C2017 **illegal escape sequence**

An escape sequence appeared where one was not expected.

An escape sequence (a backslash, \, followed by a number or letter) may occur only in a character or string constant.

C2018 **unknown character** *hexnumber*

The ASCII character corresponding to the given hexadecimal number appeared in the source file but is an illegal character.

One possible cause of this error is corruption of the source file.

C2019 **expected preprocessor directive, found** *character*

The given character followed a number sign (#), but it was not the first letter of a preprocessor directive.

C2020 *member* **:** *class* **member redefinition**

The specified member of the specified base class or structure was redefined.

A function inherited from a base class or structure cannot be redefined.

If the function is redefined in the derived class, it should be declared as **virtual** in the base class.

C2021 **expected exponent value, not** *character*

The given character was used as the exponent of a floating-point constant but was not a valid number.

C2022 *number* **: too big for character**

The octal number following a backslash (\) in a character or string constant was too large to be represented as a character.

C2023 **divide by 0**

The expression resulted in a division by zero.

C2024 **mod by 0**

The expression resulted in a modulo 0 operation.

C2025 *identifier* **: enum/struct/union type redefinition**

The given identifier had already been used for an enumeration, structure, or union tag.

C2026 **string too big, trailing characters truncated**

The string was longer than the limit of 2048 characters.

After adjacent strings are concatenated, a string cannot be longer than 2048 characters.

C2027 **use of undefined type** *identifier*

The specified type was not defined.

A type cannot be used until it is defined.

C2028 **struct/union member needs to be inside a struct/union**

Structure or union member must be declared within the structure or union, respectively.

C2030 *identifier* **: struct/union member redefinition**

The identifier was used for more than one member of the same structure or union.

C2032 *identifier* **: function cannot be member of** *struct/union identifier*

The specified structure or union was declared with a member function.

Member functions are allowed in C++ but not in C.

C2033 *identifier* **: bit field cannot have indirection**

The given bit field was declared as a pointer (*), which is not allowed.

The following is an example of this error:

```
struct S
{
   int *b : 1;  // error
};
```

C2034 *identifier* **: type of bit field too small for number of bits**

The number of bits specified in the bit-field declaration exceeded the number of bits in the given base type.

C2036 **identifier** *class-key* **: unknown size**

The address of the specified undeclared identifier was used.

The size of an undeclared object cannot be used.

The following is an example of this error:

```
struct A* pA;
struct B { int i; };
B* pB;

void main()
{
   pA++;   // error, size of A not known
   pB++;   // OK, B has been declared
}
```

C2037 left of *operator* specifies undefined struct/union *identifier*

The expression before the member-selection operator (**–>** or **.**) identified a structure or union type that was not defined.

C2039 *identifier* : not struct/union member

A nonmember of a structure or union was incorrectly used.

The following is an example of this error:

```
struct S
{
   int mem0;
} *pS;

void main()
{
   pS->mem1 = 0;       // error, mem1 is not a member
   pS->mem0 = 0;       // OK
}
```

C2040 *operator* : different levels of indirection

An expression involving the specified operator had inconsistent levels of indirection.

If both operands are of arithmetic type or if both are not (such as array or pointer), then they are used without change. However, the compiler may DS-extend one of the operands if one is **__far** and the other is **__near**. If one operand is arithmetic, but the other is not, the arithmetic operator is converted to the type of the other operator.

C2041 illegal digit *character* for base *number*

The specified character was not a legal digit for the base that was used.

The following is an example of this error:

```
int i = 081; // error, 8 is not a legal digit
int i = 071; // OK
```

C2042 **signed/unsigned keywords mutually exclusive**

The keywords **signed** and **unsigned** were both used in a single declaration.

The following is an example of this error:

```
unsigned signed int i;      // error
```

C2043 **illegal break**

A **break** statement is legal only within a **do**, **for**, **while**, or **switch** statement.

C2044 **illegal continue**

A **continue** statement is legal only within a **do**, **for**, or **while** statement.

C2045 *identifier* **: label redefined**

The label appeared before more than one statement in the same function.

C2046 **illegal case**

The keyword **case** can appear only within a **switch** statement.

C2047 **illegal default**

The keyword **default** can appear only within a **switch** statement.

C2048 **more than one default**

A **switch** statement contained more than one default label.

C2049 **case value** *value* **already used**

The **case** value was already used in this **switch** statement.

This error can be caused by enumerations or macros that evaluate to the same value.

C2050 **nonintegral switch expression**

A **switch** expression did not evaluate to an integral value.

C2051 **case expression not constant**

Case expressions must be integral constants.

C2052 **case expression not integral**

Case expressions must be integral constants.

C2053 *identifier* **: wide string mismatch**

The specified wide string was assigned to an incompatible type.

The following is an example of this error:

```
char array[] = L"Rika";
```

C2054 **expected '(' to follow** *identifier*

The context requires parentheses after the function identifier.

One cause of this error is omitting an equal sign (=) on a complex initialization, as in:

```
int array1[] { 1, 2, 3 };   // error, missing =
int array2[] = { 1, 2, 3 }; // OK
```

C2055 **expected formal parameter list, not a type list**

A parameter type list instead of a formal parameter list appeared in a function definition.

In ANSI C, the formal parameters in a function definition must all be named unless they are **void** or an ellipsis (**...**).

C2056 **illegal expression**

An expression was illegal because of a previous error, which may not have produced an error message.

C2057 **expected constant expression**

The context requires a constant expression.

C2058 **constant expression is not integral**

The context requires an integral constant expression.

C2059 **syntax error :** *token*

The token caused a syntax error.

C2060 **syntax error : end of file found**

At least one more token was expected.

Causes of this error include omitting a semicolon (**;**), as in:

```
int *p    // error
```

or omitting a closing brace (**}**) from the last function, as in:

```
main()
{         // error
```

C2061 syntax error : identifier *identifier*

The identifier caused a syntax error.

C2062 type *type* unexpected

The compiler did not expect the given type to appear here, possibly because it already had a required type.

This error can also be caused by a missing semicolon.

C2063 *identifier* : not a function

The given identifier was not declared as a function but was used as a function.

The following example in C generates this error:

```
int i, j;
j = i();    // error, i is not a function
```

C2064 term does not evaluate to a function

A call was made to a function through an expression that did not evaluate to a function pointer.

This error is probably caused by attempting to call a nonfunction.

The following is an example of this error:

```
int i, j;
char* p;
void func()
{
   j = i();    // error, i is not a function
   p();        // error, p doesn't point to a function
}
```

C2065 *identifier* : undeclared identifier

The specified identifier was not declared.

A variable's type must be specified in a declaration before it can be used. The parameters that a function uses must be specified in a declaration before the function can be used.

This error can be caused if an include file containing the required declaration was omitted.

C2066 cast to function type is illegal

An object was cast to a function type, which is illegal.

In ANSI C, it is not legal to cast between a pointer to a function and a pointer to data.

C2067 **cast to array type is illegal**

An object was cast to an array type.

C2068 **illegal cast**

A type used in a cast operation was not legal for this expression.

C2069 **cast of void term to nonvoid**

A term of type **void** was cast to a different type.

Type **void** cannot be cast to any other type.

C2070 **illegal sizeof operand**

The operand of a **sizeof** expression was not an expression or a type name.

C2071 *identifier* **: illegal storage class**

The named identifier was declared with an illegal storage class.

C2072 *identifier* **: initialization of a function**

An initializer for a function was illegally specified.

C2073 *identifier* **: partially initialized array requires a default constructor**

An array of user-defined types or an array of **consts** was specified with too few initializers.

If an explicit initializer (and its corresponding constructor) is not specified for a member of an array, then a default constructor must be supplied.

The following is an example of this error:

```
class A
{
public:
   A( int );                 // constructor for ints only
};

A a[3] = { A(1), A(2) };  // error, no default constructor

class B
{
public:
   B();                      // default constructor declared
   B( int );
};

B b[3] = { B(1), B(2) };  // OK
```

C2074 *identifier* : *class-key* **initialization needs curly braces**

There were no curly braces ({}) around the specified class, structure, or union initializer.

C2075 *identifier* : **array initialization needs curly braces**

There were no curly braces ({}) around the specified array initializer.

C2077 **nonscalar field initializer** *identifier*

An attempt was made to initialize a bit-field member of a structure with a nonscalar value.

C2078 **too many initializers**

The number of initializers exceeded the number of objects to be initialized.

C2079 *identifier* **uses undefined class/struct/union** *name*

The specified identifier was declared as a class, structure, or union that was not defined.

This error can be caused by initializing an anonymous union.

C2080 **illegal __far __fastcall function**

If stack checking is enabled, a __far **__fastcall** function cannot be compiled with the /Gw or /Gq option.

C2082 **redefinition of formal parameter** *identifier*

A formal parameter to a function was redeclared within the function body.

C2083 **struct/union comparison illegal**

A structure or union was directly compared with another user-defined type.

A user-defined type cannot be compared with another user-defined type unless a comparison operator has been defined or unless a conversion to a scalar type exists.

The following is an example of this error:

```
struct A
{
   int i;
} a, b;

void func()
{
   if( a == b );   // error, structure comparison
}
```

C2084 **function *function* already has a body**

The function has already been defined.

C2085 ***identifier* : not in formal parameter list**

The identifier was declared in a function definition but not in the formal parameter list.

A common cause of this error is the omission of a semicolon (**;**) at the end of a function prototype, as in:

```
void func1( void )
void main( void )
{
}
```

With the semicolon missing, `func1()` is taken to be a function definition, not a prototype. This means the function `main()` is being defined within `func1()`. Error C2085 is generated for the identifier `main`.

This is an error in ANSI C only.

C2086 ***identifier* : redefinition**

The given identifier was defined more than once, or a subsequent declaration differed from a previous one.

The following examples generate this error:

```
int a;
char a;
main()
{
}

main()
{
   int a;
   int a;
}
```

The following is an error in C++ but not in ANSI C:

```
int a;
int a;
main()
{
}
```

C2087 ***identifier* : missing subscript**

The definition of an array with multiple subscripts was missing a subscript value for a dimension other than the first dimension.

The following is an example of this error:

```
int func(a)
char a[10][];         // error
{ }

int func(a)
char a[][5];          // OK
{ }
```

C2088 *operator* **: illegal for** *class-key*

The specified operator was not defined for the class, structure, or union.

This error can be eliminated by defining a conversion to convert the operands to the type for which the operator is defined. Alternatively, an overloaded conversion operator can be defined.

An example of this error and two solutions are:

```
class A
{
public:
   int i;
} a;
int i = 1 + a;       // this line causes the error

class B
{
public:
   operator int() { return i; }
   int i;
} b;
int j = 1 + b;       // OK, uses conversion operator

class C
{
public:
   int operator+( int j ) { return (j+i); }  // solution
   int i;
} c;
int k = c + 1;       // OK, uses overloaded operator+
int i = 1 + c;       // still an error, no conversion
                           // operator defined for class C
```

Note that the last line requires that a conversion operator be defined since overloaded operators cannot be defined for built-in types.

C2089 *identifier* **:** *class-key* **too large**

The specified structure or union was larger than 64K.

A structure or union cannot be larger than 64K.

C2090 **function returns array**

A function cannot return an array. It can return a pointer to an array.

C2091 **function returns function**

A function cannot return a function. It can return a pointer to a function.

C2092 **array element type cannot be function**

Arrays of functions are not allowed. Arrays of pointers to functions are allowed.

C2093 **cannot use address of automatic variable as static initializer**

The program tried to use the address of an automatic variable in the initializer of a static item, as in the following example:

```
func()
{
   int i;
   static int *ip=&i;    // error
}
```

C2094 **label** ***identifier*** **was undefined**

A **goto** label was found, but the specified label did not exist in the same function.

C2095 *function* **: actual parameter has type void : parameter** *number*

An attempt was made to pass a **void** parameter to a function. The given number indicates which parameter was in error.

Formal parameters and parameters to functions cannot have type **void**. They can, however, have type **void *** (pointer to **void**).

C2097 **illegal initialization**

One of the following was illegally attempted:

- Initialization of a variable using a nonconstant value
- Initialization of a short address with a long address
- Initialization of a local structure, union, or array with a nonconstant expression when compiling with /Za
- Initialization with an expression containing a comma operator (**,**)
- Initialization with an expression that is neither constant nor symbolic

C2098 **nonaddress expression**

An address was expected as the initialization expression.

C2099 **nonconstant initializer**

The initializer was not a constant.

The following is an example of this error when compiled as C:

```
int i, j;
int *p;
j = i();     // error, i() is not constant
j = *p;      // error, *p is not a constant
```

C2100 **illegal indirection**

The indirection operator (*) was applied to a nonpointer value.

C2101 **'&' on constant**

The address-of operator (**&**) did not have an l-value as its operand.

C2102 **'&' requires l-value**

The address-of operator (**&**) must be applied to an l-value expression.

C2103 **'&' on register variable**

An attempt was made to use the address of a register variable.

C2104 **'&' on bit field ignored**

An attempt was made to use the address of a bit field.

C2105 *operator* **needs l-value**

The given operator did not have an l-value operand.

C2106 *operator* **: left operand must be l-value**

The left operand of the given operator was not an l-value.

C2107 **illegal index; indirection not allowed**

A subscript was applied to an expression that did not evaluate to a pointer.

C2108 **nonintegral index**

A nonintegral expression was used in an array subscript.

C2109 **subscript on nonarray**

A subscript was used on a variable that was not an array.

C2110 **pointer + pointer**

An attempt was made to add one pointer to another using the plus operator (**+**).

C2111 **pointer + nonintegral value**

An attempt was made to add a nonintegral value to a pointer using the plus operator (+).

C2112 **illegal pointer subtraction**

An attempt was made to subtract pointers that did not point to the same type.

C2113 **pointer subtracted from nonpointer**

The right operand in a subtraction operation using the minus operator (–) was a pointer, but the left operand was not.

C2114 *operator* **: pointer on left; needs integral value on right**

The left operand of the given operator was a pointer. Therefore, the right operand must be an integral value.

C2115 *identifier* **: incompatible types**

An expression contained incompatible types.

C2116 **function parameter lists differed**

The parameters in the default parameter list did not match the formal parameter list.

C2118 **negative subscript**

A value defining an array size was negative.

C2119 **typedef types both define indirection**

Two **typedef** types were used to declare an item, and both **typedef** types had indirection.

For example, the declaration of `p` in the following example is illegal:

```
typedef int *p_int;
typedef short *p_short;
p_short p_int p;        // error
```

C2120 **void illegal with all types**

The **void** type was used in a declaration with another type.

C2121 *operator* **: bad left/right operand**

The left or right operand of the given operator was illegal for that operator.

C2122 *identifier* : **prototype parameter in name list illegal**

The specified parameter was not a legal type.

User-defined types are not supported in ANSI C.

C2123 *function1* : **cannot call __fastcall function** *function2* **from p-code**

There was an attempt to call a fastcall function from within a p-code function.

Rebuild *function2* with a different calling convention, or turn off p-code generation for *function1* by using `#pragma optimize( "q", off )`.

C2124 **divide or mod by zero**

A constant expression was evaluated and found to have a zero denominator.

C2125 *identifier* : **allocation exceeds 64K**

The given item exceeded the size limit of 64K.

C2126 *operand* : **incorrect operand**

The specified operator was used on an enumeration.

The increment and decrement operators (++ and – –) are not defined for enumerated types.

C2127 **parameter allocation exceeds 32K**

The storage space required for the parameters to a function exceeded the limit of 32K.

C2128 *function* : **no function with C linkage found**

The specified function was not found.

This error is caused by omitting the file containing the function definition from the project list or makefile, by not defining the function within the file scope, or by omitting the keyword **extern** in the function prototype.

C2129 **static function** *function* **declared but not defined**

A forward reference was made to a static function that was never defined.

A function declared with static linkage must be defined within file scope. If the function is defined in another file, it should be declared with the keyword **extern**.

C2130 **#line expected a string containing the filename; found** *token*

The optional token following the line number on a **#line** directive was not a string.

C2131 more than one memory attribute

More than one memory attribute (**__near**, **__far**, **__huge**, or **__based**) was applied to an item, as in the following example:

```
typedef int __near nint;
nint __far a;              // error
```

C2132 syntax error : unexpected identifier

An identifier appeared in a syntactically illegal context.

C2133 *identifier* : unknown size

An unsized array was declared as a member of a class, structure, union, or enumeration.

Unsized member arrays are not allowed when the /Za (ANSI) switch has been chosen.

The following is an example of this error:

```
struct X
{
   int a[0]; // error, unsized array
};
```

C2134 *identifier* : struct/union too large

The size of a structure or union exceeded the compiler limit of 64K.

C2135 *identifier* : illegal bit-field operation

The address of the specified bit field was taken.

The address-of operator (**&**) cannot be applied to a bit field.

The following is an example of this error:

```
struct S
{
   int i : 1;
   int j;
};

void main()
{
   &S::i;       // error, address of a bit field
   &S::j;       // OK
}
```

C2136 *function* **: prototype must have parameter types**

A function prototype declaration had formal parameter names, but no types were provided for the parameters.

A formal parameter in a function prototype must either have a type or be represented by an ellipsis (**...**) to indicate a variable number of parameters and no type checking.

One cause of this error is a misspelling of a type name in a prototype that does not provide the names of the formal parameters.

C2137 **empty character constant**

The illegal empty character constant (**' '**) was used.

C2139 **type following** *identifier* **is illegal**

Two types were used in the same declaration, as in:

```
int double a;
```

C2140 **parameter cannot be function type**

A function was illegally declared as a formal parameter of another function.

C2141 **value out of range for enum constant**

An enumeration constant had a value outside the range of values allowed for an integer type.

C2142 **function declarations differ; variable parameters specified only in one of them**

One declaration of the function contained a variable parameter list, but another declaration did not.

This causes an error in C when the /Za (ANSI) switch is chosen.

The following is an example of this error:

```
void func();
void func( int, ... );
```

C2143 **syntax error : missing** *token1* **before** *token2*

The compiler expected *token1* to appear before *token2.*

This error can be caused by a missing closing brace (**}**), right parenthesis, or semicolon (**;**). The missing token may belong on the line above where the error was detected.

This error can also be caused by an invalid tag in a class declaration.

The following are examples of this error:

```
class X
{
   int member
} x;                // error, missing ; on previous line

class + {};         // error, + is invalid tag name
```

C2144 **syntax error : missing** *token* **before type** *type*

The compiler expected the given token to appear before the given type name.

This error can be caused by a missing closing brace (**}**), right parenthesis, or semicolon (**;**).

C2145 **syntax error : missing** *token* **before identifier**

The compiler expected the given token to appear before an identifier.

This error can be caused by a missing semicolon (**;**) after the last declaration in a block.

C2146 **syntax error : missing** *token* **before identifier** *identifier*

The compiler expected the given token to appear before the given identifier.

C2148 **array too large**

An array exceeded the maximum legal size of 64K.

Either reduce the size of the array or declare it with **__huge**.

C2149 *identifier* **: named bit field cannot have zero width**

The given named bit field had zero width. Only unnamed bit fields are allowed to have zero width.

C2150 *identifier* **: bit field must have type int, signed int, or unsigned int**

The ANSI C standard requires bit fields to have types of **int**, **signed int**, or **unsigned int**. This message appears only when compiling with the /Za option.

C2151 **more than one language attribute**

More than one keyword specifying a calling convention (**__cdecl**, **__fortran**, **__pascal**, or **__fastcall**) for a function was given.

C2152 *identifier* **: pointers to functions with different attributes**

An attempt was made to assign a pointer to a function declared with one calling convention (**__cdecl**, **__fortran**, **__pascal**, or **__fastcall**) to a pointer to a function declared with a different calling convention.

C2153 **hex constants must have at least one hex digit**

The hexadecimal constants **0x**, **0X**, and **\x** are illegal. At least one hexadecimal digit must follow the **x** or **X**.

C2154 *segment* **: does not refer to a segment name**

A segment must be allocated when a based variable is declared unless it is **extern** and uninitialized.

C2156 **pragma must be outside function**

A pragma that must be specified at a global level (that is, outside a function body) occurred within a function.

The following is an example of this error:

```
main()
{
   #pragma optimize( "l", on )
}
```

C2157 *function* **: must be declared before use in pragma list**

The function name in the list of functions for an **alloc_text** pragma has not been declared prior to being referenced in the list.

C2158 *identifier* **: is a function**

The given identifier was specified in the list of variables in a **same_seg** pragma but was previously declared as a function.

C2159 **more than one storage class specified**

A declaration contained more than one storage class, as in:

```
extern static int i;      // error
```

C2160 **## cannot occur at the beginning of a macro definition**

A macro definition began with a token-pasting operator (**##**), as in:

```
#define mac(a,b) ##a
```

C2161 **## cannot occur at the end of a macro definition**

A macro definition ended with a token-pasting operator (##), as in:

```
#define mac(a,b) a##
```

C2162 **expected macro formal parameter**

The token following a stringizing operator (#) was not a formal parameter name.

The following is an example of this error:

```
#define print(a) printf(#b)
```

C2163 *function* **: not available as an intrinsic function**

A function specified in the list of functions for an intrinsic or function pragma is not one of the functions available in intrinsic form.

C2164 *function* **: intrinsic function not declared**

The given function was not declared before being used in an **intrinsic** pragma. This error appears only when compiling with the /Oi option.

C2165 *keyword* **: cannot modify pointers to data**

The **__fortran**, **__pascal**, **__cdecl**, or **__fastcall** keyword was used illegally to modify a pointer to data, as in the following example:

```
char __pascal *p;
```

C2166 **l-value specifies const object**

An attempt was made to modify an item declared with **const** type.

C2167 *function* **: too many actual parameters for intrinsic function**

A reference to a function declared as **intrinsic** contained too many actual parameters.

C2168 *function* **: too few actual parameters for intrinsic function**

A reference to a function declared as **intrinsic** contained too few actual parameters.

C2169 *function* **: intrinsic function; cannot be defined**

An attempt was made to provide a function definition for a function already declared as an **intrinsic**.

C2170 *identifier* **: not declared as a function; cannot be intrinsic**

The **intrinsic** pragma was used for an item other than a function or for a function that does not have an **intrinsic** form.

C2171 *operator* **: illegal operand**

The given unary operator was used with an illegal operand type, as in the following example:

```
int (*fp)();
double d, d1;

fp++;            // error
d = ~d1;         // error
```

C2172 *function* **: actual parameter is not a pointer : parameter** *number*

An attempt was made to pass an parameter that was not a pointer to a function that expected a pointer. The given number indicates which parameter was in error.

C2173 *function* **: actual parameter is not a pointer : parameter** *number1*, **parameter list** *number2*

An attempt was made to pass a nonpointer parameter to a function that expected a pointer.

This error occurs in calls that return a pointer to a function. The first number indicates which parameter was in error; the second number indicates which parameter list contained the invalid parameter.

C2174 *function* **: actual parameter has type void : parameter** *number1*, **parameter list** *number2*

An attempt was made to pass a **void** parameter to a function. Formal parameters and parameters to functions cannot have type **void**. They can, however, have type **void*** (pointer to **void**).

This error occurs in calls that return a pointer to a function. The first number indicates which parameter was in error; the second number indicates which parameter list contained the invalid parameter.

C2176 **static huge data not supported by** *identifier*

A huge array was declared in a p-code function.

Arrays declared using **__huge** are not allowed in p-code functions.

C2177 **constant too big**

A constant value was too large to be represented in the type to which it was assigned.

C2178 *identifier* **: storage class for same_seg variables must be extern**

The given variable was specified in a **same_seg** pragma, but the variable was not declared with **extern** storage class.

C2179 *identifier* **: was used in same_seg, but storage class is no longer extern**

The given variable was specified in a **same_seg** pragma, but the variable was redeclared with a storage class other than **extern**.

C2180 **controlling expression has type void**

The controlling expression in an **if**, **while**, **for**, or **do** statement was either a function with **void** return type or an expression cast to **void**.

C2182 *identifier* **: has type void**

The given variable was declared with the keyword **void**, which can be used only in function declarations.

C2184 **illegal return of a void value**

The function did not return a value.

The function was declared as returning a nonvoid value, but the return statement did not return a value.

C2185 *identifier* **: illegal based allocation**

A based-allocated variable that explicitly has **extern** storage class and is uninitialized cannot have any of the following bases:

- `(__segment) & var`
- `__segment ("_STACK")`
- `(__segment) __self`
- `void`

If the variable does not explicitly have **extern** storage class or is initialized, then its base must use `__segname("string")` where `string` is any segment name or reserved segment name except `_STACK`.

C2186 *operand* **: illegal operand of type void**

The specified operator had a **void** operand.

The following is an example of this error:

```
void func1( void );
int  func2( void );

int i = 2 + func1();   // error, func1() is type void
int j = 2 + func2();   // OK, both operands are type int
```

C2187 **cast of near function pointer to far function pointer**

A near function pointer was cast to a far function pointer.

This cast is not allowed.

C2188 *number* **: too big for wide character**

The given number is too large to be held in the wide-character type.

Choose a larger type to hold the given value.

C2189 **#error :** *string*

An **#error** directive was encountered.

The string is the descriptive text supplied in the directive.

C2190 **first parameter list longer than second**

The function was declared a second time with a shorter parameter list.

C does not support overloaded functions.

The following is an example of this error:

```
void func( int, float );
void func( int,  );         // error, different parameter list
```

C2191 **second parameter list longer than first**

The function was declared a second time with a longer parameter list.

C does not support overloaded functions.

The following is an example of this error:

```
void func( int );
void func( int, float ); // error, different parameter list
```

C2192 **parameter** *number* **declaration different**

The function was declared a second time with a different parameter list.

C does not support overloaded functions.

The following is an example of this error:

```
void func( float, int );
void func( int, float ); // error, different parameter list
```

C2193 *identifier* **: already in a segment**

A variable in the **same_seg** pragma has already been allocated in a segment, using **__based**.

C2194 *segment* **: is a text segment**

The given text segment was used where a data, **const**, or BSS segment was expected.

C2195 *segment* **: is a data segment**

The given data segment was used where a text segment was expected.

C2197 *identifier* **: too many actual parameters**

The specified function was called with too many parameters, or the function declaration was incorrect.

The following is an example of this error:

```
void func( int );
main()
{
   func( 1, 2 );   // error, two actual parameters
}
```

C2198 *identifier* **: too few actual parameters**

The specified function was called with too few parameters, or the function declaration was incorrect.

The following is an example of this error:

```
void func( int, int );
main()
{
   func( 1 );   // error, only one actual parameter
}
```

C2199 **syntax error : found** *identifier (* **at global scope (was a declaration intended?)**

The specified context caused a syntax error.

This error can be caused by incorrect declaration syntax.

The following is an example of this error:

```
struct S
{
public:
   int i;
   S( int si ) { i = si; }
};

S(1) s; // error, incorrect syntax
S s(1); // OK
```

C2200 *function* **: function has already been defined**

A function name passed as an parameter in an **alloc_text** pragma has already been defined.

C2201 *function* **: storage class must be extern**

A function declaration appeared within a block, but the function was not declared as **extern**. This causes an error if the /Za ANSI-compatibility option is in effect.

The following example causes this error when compiled with /Za:

```
main()
{
   static int func1();    // error
}
```

C2202 *function* **: not all control paths return a value**

The specified function can potentially not return a value.

The following is an example of this error:

```
int func1( int i )
{
   if( i ) return 3;  // error, nothing returned if i = 0
}

int func2( int i )
{
   if( i ) return 3;
   else return 0;     // OK, always returns a value
}
```

C2203 **delete operator cannot specify bounds for an array**

The **delete** operator can only delete an entire array; it cannot delete parts or specific members of the array. This error is generated only with the /Za ANSI-compatibility option.

This was not an error in C++ 2.0 but is an error in C++ 2.1.

The following statement generates this error:

```
delete [4] ArrayOfObjects;   // error
```

C2204 *identifier* **: type definition found within parentheses**

The specifed type was defined in prototype scope or as an operand.

The following is an example of this error:

```
( struct S {}; );  // error
```

C2205 *identifier* **: cannot initialize extern variables with block scope**

A variable with **extern** storage class cannot be initialized in a function.

C2206 *function* **: typedef cannot be used for function definition**

A **typedef** was used to define a function type.

For example:

```
typedef int functyp();
functyp func1 {};        // error
```

C2207 *member* **in struct/union** *tag* **has a zero-sized array**

The given member in the structure or union contains an array that does not have a subscript or that has a zero subscript. This kind of array is legal only as the last member of a structure or union.

C2208 *type* **: no members defined using this type**

An enumeration, structure, or union was defined without any members. This is an error only when compiling with /Za; otherwise, it is a warning.

C2209 **type cast in __based construct must be (__segment)**

Only type **__segment** can be used within a cast in a **__based** declarator.

C2210 *identifier* **: must be near/far data pointer**

The base in a **__based** declarator must be a near or far data pointer and cannot be an array, function, or based pointer.

C2211 **(__segment) applied to function identifier** *function*

A function cannot be cast in a **__based** declarator.

C2212 *identifier* **: __based not available for pointers to functions**

Based pointers cannot be used to point to functions. Function pointers can be **__near** or **__far** only.

C2213 *identifier* **: illegal parameter to __based**

The parameter used as a base must have type **__segment** or be a near or far pointer.

C2214 **pointers based on void require the use of :>**

A based pointer based on **void** cannot be dereferenced. Use the base operator (**:>**) to create an address that can be dereferenced.

C2215 **:> operator only for objects based on void**

The right operand of the base operator (**:>**) must be a pointer based on **void**, as in:

```
char __based( void ) *cbvpi;
```

C2216 *attribute1* **cannot be used with** *attribute2*

The given function attributes are incompatible.

Some combinations of attributes that cause this error are:

- **__saveregs** and **__interrupt**
- **__fastcall** and **__saveregs**
- **__fastcall** and **__interrupt**
- **__fastcall** and **__export**

C2217 *attribute1* **must be used with** *attribute2*

The first function attribute requires the use of the second attribute.

Some causes for this error include:

- An interrupt function explicitly declared as near. Interrupt functions must be declared as far.
- An interrupt function that is declared with the **__fortran**, **__pascal**, or **__fastcall** attribute. Functions declared with the **__interrupt** attribute must use C calling conventions.
- A function with a variable number of parameters that is declared with the **__fortran**, **__pascal**, or **__fastcall** attribute. These functions must use C calling conventions. Remove the **__fortran**, **__pascal**, or **__fastcall** attribute from the function declaration.

C2218 **type in __based construct must be void**

The only type allowed in a **__based** construct is **void**.

C2219 **syntax error : type qualifier must be after '*'**

Either **const** or **volatile** appeared where a type or qualifier is not permitted, as in:

```
int (const *p);
```

C2220 **warning treated as error—no object file generated**

When the compiler option /WX is used, the first warning generated by the compiler causes this error message to be displayed.

Either correct the condition that caused the warning or compile at a lower warning level or without /WX.

C2221 **'.' : left operand points to class/struct/union; use '–>'**

The left operand of the member-of operator (.) must be a class, (or structure or union) but not a pointer to a class type.

The class member access operator (–>) can be used with a pointer to a class, structure, or union.

C2222 **'–>' : left operand has class/struct/union type; use '.'**

The left operand of the class member access operator (–>) must be a pointer to a class (or structure or union) type but not a class type.

The member-of operator (.) can be used with a class, structure, or union type.

C2223 **left of** *–>member* **must point to class/struct/union**

The left operand of the member access operator (–>) is not a pointer to a class, structure, or union type.

This error can occur when the left operand is an undefined variable. Undefined variables have type **int**.

C2224 **left of** *.member* **must have class/struct/union type**

The left operand of the member-of operator (.) is not a class, structure, or union type.

This error can occur when the left operand is an undefined variable. Undefined variables have type **int**.

C2226 **syntax error : unexpected type** *type*

A syntax error occured before, or in, the given type specifier.

The following is an example of this error:

```
int func1( int, ... , float ); // error, misplaced ellipsis
int func2( int, float, ... );  // OK
```

C2227 **left of** *–>identifier* **must point to class/struct/union**

The left side of the specified class member access operator (–>) was not a pointer to a class, structure, or union.

The following is an example of this error:

```
int  *pInt;

struct S
{
public:
   int member;
} *pS;
```

```
void main()
{
   pInt->member = 0;   // error, pInt points to an int
   pS->member = 0;     // OK, pS points to a structure S
}
```

C2228 left of *.identifier* must have class/struct/union type

The left side of the specified class member access operator (**.**) was not a class (or structure or union) type.

The following is an example of this error:

```
int i;

struct S
{
public:
   int member;
} s, *ps;

void main()
{
   i.member = 0;   // error, i is not a class type
   ps.member = 0;  // error, ps is a pointer to a structure
   s.member = 0;   // OK, s is a structure type
   ps->member = 0; // OK, ps points to a structure S
}
```

C2229 type *identifier* has an illegal zero-sized array

The specified member of the structure or bit field contained a zero-sized array.

The following is an example of this error:

```
struct S
{
   int a[0];  // error, zero-sized array
   int b[1];  // OK
};
```

C2230 *identifier* : first member of *class-key* is unnamed

The first member of a bit field was unnamed.

The first member of a bit field must be named.

C2231 '.' : left operand points to *class-key*; use '–>'

The left operand to the member selection operator (**.**) was a pointer to a class, structure, or union.

The left operand to the member selection operator must be a class, structure, or union.

The following is an example of this error:

```
struct S
{
public:
   int member;
} s, *ps;

void main()
{
   ps.member = 0;  // error, ps points to structure S
   ps->member = 0; // OK, ps points to a structure S
   s.member = 0;   // OK, s is a structure type
}
```

C2232 '–>' : left operand has *class-key* type; use '.'

The class member access operator (–>) was used on a nonpointer.

The pointer form of the class member access operator can only be used with a pointer to a class, structure, or union. The dot (.) form of the operator should be used with a class, structure, or union type.

The following is an example of this error:

```
struct X
{
   int member;
} x, *px;

void main()
{
   x->member = 0;   // error, x is not a pointer
   px->member = 0;  // OK, px is a pointer to an X
   x.member = 0;    // OK
}
```

C2234 arrays of references are illegal

An array of references was declared.

Since pointers to references are not allowed, arrays of references are not possible. A pointer should be used to implement the array.

C2235 ';' in formal parameter list

A semicolon (;) was found in a formal parameter list.

The error is usually caused by using a semicolon instead of a comma (,) to separate parameters in a formal parameter list.

The following is an example of this error:

```
void func( int i; float f );   // error, uses semicolon
void func( int i, float f );   // OK, uses comma
```

C2236 **unexpected** *class-key identifer*

The specified identifier was already defined as a type and cannot be overridden by a new user-defined type.

C2237 **unexpected class-key** *identifier*

The specified class-key was not followed by a valid class name.

The following is an example of this error:

```
class + {};  // error, + is invalid class name
```

C2238 **unexpected token preceding** *token*

An incorrect token, or tokens, was found before the specified token.

This error can be caused by an invalid name in a bit-field declaration.

The following is an example of this error:

```
struct bits
{
   int field1 : 16;
   int 9      : 16;  // error, 9 is not a valid name
};
```

C2239 **unexpected token** *token* **following declaration of** *identifier*

An unexpected token was found in the specified declaration.

The following is an example of this error:

```
int i 8;   // error, missing =
int j = 7; // OK
```

C2241 *identifier* **: member access is restricted**

There was an attempt to access a private or protected member function or data.

If the program needs to access this member, change the object access level or make the member a **friend** of the function that needs to be accessed.

C2244 *identifier* : unable to resolve function overload

The specified overloaded function call was ambiguous.

The following is an example of this error:

```
int func( char );
int func( int );
void main ()
{
   +func;       // error, can't resolve which func to use
   +func( 0 );  // OK
}
```

C2245 nonexistent function *identifier* specified as friend

The specified identifier was not a function.

Only a function can be specified as a **friend**.

The following is an example of this error:

```
class C
{
public:
   int i;                  // i is not a function
   void func();
};

class S
{
public:
   friend void C::i();     // error
   friend void C::func();  // OK
};
```

C2246 *identifer* : illegal static data member in locally defined class

The specified member of a class, structure, or union with local scope was declared as **static**.

The following is an example of this error:

```
void func( void )
{
   class A
   {
      static int i;    // error, i is local to func
   };
};

class B
{
   static int i;       // OK
};
```

C2247 *identifier* **not accessible because** *class* **uses** *specifier* **to inherit from** *class*

The specified identifier was inherited from a class declared with private or protected access.

The following is an example of this error:

```
class A
{
public:
   int i;
};

class B : private A {};    // B inherits a private A

class C : public B {} c;   // so even though C's B is public

int j = c.i;               // error, i not accessible
```

C2248 *member* **: cannot access** *specifier* **member declared in class** *class*

The specified private or protected member of a class, structure, or union was accessed.

The member should be accessed through a member function with public access or should be declared with public access.

The following is an example of this error:

```
class X
{
public:
   int pubMemb;
   void setPrivMemb( int i ) { privMemb = i; }
protected:
   int protMemb;
private:
   int privMemb;
} x;

void main()
{
   x.privMemb = 0;        // error, privMemb is private
   x.protMemb = 0;        // error, protMemb is protected
   x.pubMemb = 0;         // OK, pubMemb is public
   x.setPrivMemb( 0 );    // OK, uses public access function
}
```

C2249 *identifier* **: no accessible path to** *specifier* **member declared in virtual base** *class*

The specified inherited member was inherited from a nonpublic virtual base class or structure.

The following is an example of this error:

```
class A
{
private:
   void privFunc( void ) {};
public:
   void pubFunc( void ) {};
};

class B : virtual public A {} b;

void main( void )
{
   b.privFunc();        // error, private member of A
   b.pubFunc();         // OK
}
```

C2250 *identifier* **: ambiguous inheritance of** *class::member*

The derived class inherited more than one override of a virtual function of a virtual base. These overrides are ambiguous in the derived class.

The following is an example of this error:

```
struct V { virtual void vf(); };
struct A : virtual V { void vf(); };
struct B : virtual V { void vf(); };
struct D : A, B {};                  //error
```

C2252 *identifier* **: pure specifier can only be specified for functions**

The given nonfunction was specified as pure virtual.

Only member functions specified as virtual can be declared with a pure specifier.

The following is an example of this error:

```
class A
{
   virtual int i = 0;          // error, i is an int
   virtual void func() = 0;    // OK, func is a function
};
```

C2253 *function* **: pure specifier only applies to virtual function**

The specified nonvirtual function was specified as pure virtual.

The specifier was ignored.

The following is an example of this error:

```
class A
{
public:
   void func1() = 0;          // error, not virtual
   virtual void func2() = 0;  // OK
};
```

C2254 *function* **: pure specifier not allowed on friend functions**

The specified friend function was specified as pure virtual.

The following is an example of this error:

```
class A
{
public:
   friend void func1() = 0;   // error, func1 is friend
   void virtual func2() = 0;  // OK, pure virtual
   friend void func3();       // OK, friend not virtual nor
};                            //     pure

void func1() {};
void func3() {};
```

C2255 *function* **: a friend function can only be declared in a class**

The specified function was declared with the **friend** specifier outside of a class, structure, or union.

The following is an example of this error:

```
class A
{
private:
   void func1();
   friend void func2();
};

friend void func1() {};    // error
void func2() {};           // OK
```

C2256 **illegal use of friend specifier on** *destructor*

The specified destructor was specified as a **friend**.

A destructor cannot be specifed as a **friend**.

The following is an example of this error:

```
class C
{
public:
   friend ~C(); // error
   ~C();        // OK
};
```

C2257 p-code generation pragma not allowed without /Oq

The "q" optimization pragma is not allowed without the /Oq p-code command-line option.

To generate p-code from CL, use the /Oq p-code command-line option. Once the option has been set, the optimization pragmas can turn p-code generation on and off.

C2258 illegal pure syntax, must be '= 0'

A pure virtual function was declared with incorrect syntax.

The following is an example of this error:

```
class A
{
public:
   void virtual func1() = 1; // error, not = 0
   void virtual func2() = 0; // OK
};
```

C2259 *class* : illegal attempt to instantiate abstract class

An object of the specified abstract class or structure was declared.

A class (or structure) with one or more pure virtual functions cannot be instantiated. Each pure virtual function must be overridden in a derived class before objects of the derived class can be instantiated.

The following is an example of this error:

```
class V
{
public:
   void virtual func() = 0;
};

class A : public V {};

class B : public V
{
public:
   void func();
};
```

```
V v;  // error, V is an abstract class
A a;  // error, A inherits func() as pure virtual
B b;  // OK, B defines func()
```

C2260 function pointer cast to a data pointer

A pointer to a function was cast to a pointer to data.

This cast is not legal in ANSI C.

This cast is legal in C++ and is allowed by the Microsoft extensions (using the /Ze switch).

C2261 data pointer cast to a function pointer

A pointer to data was cast to a pointer to a function.

This cast is not legal in ANSI C.

This cast is legal in C++ and is allowed by the Microsoft extensions (using the /Ze switch).

C2262 *identifier* : cannot be destroyed

The specified identifier was not instantiated because an appropriate destructor was not available.

The following is an example of this error:

```
struct S
{
   ~S() __near;
};

S __far fs;  // error, wrong memory model for destructor

class B
{
   ~B();
};

class D : public B {};
D d;         // error, B's destructor is private
```

C2263 function returns pointer based on __self

A function attempted to return a pointer based on the **__self** segment. Since **__self** refers to the code segment of the function, it is impossible to return this base to another function.

Modify the return statement so that it returns a different type, such as a far pointer or a pointer based on **void**.

C2264 *function* **: error in function definition or declaration; function not called**

Since the specified function was incorrectly defined or declared, it could not be called.

The following is an example of this error:

```
class C
{
public:
   operator int( int = 10 );    // incorrect declaration here
};

void main()
{
   int i;
   C c;
   i = c;                       // error
}
```

C2268 *operation* **: different const or volatile qualifiers**

The given operation was performed on a variable that was defined as being **const** or **volatile**. As a result, the **const** or **volatile** item could be modified without being detected by the compiler.

This error often occurs when a pointer to an item declared as **const** or **volatile** is assigned to a pointer that was not declared as pointing to either of these type modifiers.

The following is an example of this error:

```
const  char *p = "abcde";
int    str( char *s );
str( p );
```

C2269 *identifier* **: different ambient model than base class** *class*

The specified derived class or structure did not explicitly specify an ambient memory model.

A class that is derived from base classes with differing ambient memory models must specify an ambient memory model.

The following is an example of this error:

```
class __far F {};
class __near N {};
class Err : F, N {};           // error
class __far NoErr : F, N {};   // OK
```

C2270 *identifier* **: modifiers not allowed on nonmember functions**

The specified nonmember function was declared with a memory-model modifier.

Only functions that are members of a class, structure, or union can have memory-model modifiers.

The following is an example of this error:

```
void func1() __near;     // error, nonmember function

class C
{
public:
   void func2() __near;  // OK
};
```

C2271 *operator* **: new/delete cannot have formal list modifiers**

The specified operator was declared with a memory-model specifier.

A memory-model specifier cannot be specified for the **new** or **delete** operators.

The following is an example of this error:

```
class C
{
void *operator new( unsigned ) __far;   // error
};
```

C2272 *function* **: modifiers not allowed on static member functions**

The specified static member function was declared with a memory-model specifier.

The following is an example of this error:

```
class C
{
public:
   static void func1() __far;  // error, func1 is static
   void func2() __far;         // OK
};
```

C2273 *type* **: illegal as right side of –> operator**

The given type was specified on the right hand side of the class member access operator (–>).

To access a user-defined type conversion, use the **operator** keyword between the –> operator and the type.

The following is an example of this error:

```
i = ClassPtr->int( a );           // error
i = ClassPtr->operator int( a );  // OK
```

C2274 *type* : illegal as right side of ’.’ operator

The given type was specified on the right side of the class member access operator (**.**).

To access a user-defined type conversion, use the **operator** keyword between the dot operator (**.**) and the type.

The following is an example of this error:

```
i = ClassName.int( a );           // error
i = ClassName.operator int( a );  // OK
```

C2350 *identifier* is not a static member

A nonstatic member of a class or structure was defined.

Only a static member or member of an instance or a class or structure can be defined.

The following is an example of this error:

```
class C
{
public:
   int i;
   static int s;
};

void main()
{
   int C::i = 0;  // error, nonstatic member
   int C::s = 0;  // OK, static member
   C.c;
   c.i = 0;       // OK, member of instance of C
}
```

C2351 obsolete C++ constructor initialization syntax

A direct base class was not named in the constructor.

The new-style initialization list for a constructor member requires each direct base class to be explicitly named, even if it is the only base class in the list.

The following is an example of this error:

```
class B
{
public:
   B();
   B( int );
};
```

```
class D : public B
{
public:
   D( int i ) : ( i ) {}   // error, B was not named
   D( int i ) : B( i ) {}  // OK
};
```

C2352 *class::function* **: illegal call of nonstatic member function**

The specified nonstatic member function was called in a static member function.

The following is an example of this error:

```
class X
{
public:
   static void func1();
   void func2();
   static void func3()
   {
      func1();          // error, calls static func1
      func2();          // OK, calls nonstatic func2
   }
};
```

C2353 *constructor* **: improper use of constructor initializers**

The constructor initializer syntax was incorrect.

This error can be caused by omitting the constructor definition from a constructor initializer.

The following is an example of this error:

```
class C
{
public:
   C() : i( 10 );    // error, missing definition
   C() : i( 10 ){};  // OK
private:
   int i;
};
```

C2354 *reference* **: initialization of reference to member requires a temporary variable**

A reference to a member was initialized in a constructor.

The member should be initialized instead of its reference.

C2355 *this* : can only be referenced inside nonstatic member functions

The **this** pointer is only valid within nonstatic member functions. Make sure that the **this** pointer is being used in this context.

The following code generates this error in global scope:

```
char *p = this;     // error
```

C2356 initialization segment cannot change within translation unit

The **#pragma init_seg** statement cannot be preceded by segment initialization code. The **#pragma init_seg** statement must precede any code and cannot be preceded by another **#pragma init_seg** statement.

Move the segment initialization code to the beginning of the module. If multiple areas must be initialized, move them to seperate modules.

C2360 initialization of *identifier* is skipped by case label

The specified identifier initialization can be skipped in a **switch** statement.

It is illegal to jump past a declaration with an initializer unless the declaration is enclosed in a block.

The scope of the initialized variable lasts until the end of the **switch** statement unless it is declared in an enclosed block within the **switch** statement.

The following is an example of this error:

```
void func( void )
 {
    int x;
    switch ( x )
    {
    case 0 :
       int i = 1;       // error, skipped by case 1
       { int j = 1; }   // OK, initialized in enclosing block
    case 1 :
       int k = 1;       // OK, initialization not skipped
    }
 }
```

C2361 initialization of *identifier* is skipped by default label

The specified identifier initialization can be skipped in a **switch** statement.

It is illegal to jump past a declaration with an initializer unless the declaration is enclosed in a block.

The scope of the initialized variable lasts until the end of the **switch** statement unless it is declared in an enclosed block within the **switch** statement.

The following is an example of this error:

```
void func( void )
 {
    int x;
    switch (x)
    {
    case 0 :
       int i = 1;               // error, skipped by default
       { int j = 1; }           // OK, initialized in enclosing block
    default :
       int k = 1;               // OK, initialization not skipped
    }
 }
```

C2362 **initialization of** *identifier* **is skipped by** *goto label*

A jump to the specified label prevented the specified identifier from being initialized.

It is illegal to jump past a declaration with an initializer unless:

- The declaration is enclosed in a block that is not entered.
- The jump is from a point where the variable has already been initialized.

The following is an example of this error:

```
void func()
 {
    goto label1;
      int i = 1;                // error, initialization skipped
      {
         int j = 1;             // OK, this block is never entered
      }
   label1:;
}
```

C2370 *identifer* **: redefinition; different storage class**

The given identifier was already declared with a different storage-class specifier.

The following is an example of this error:

```
extern int i;
static int i;  // error
```

C2371 *identifier* **: redefinition; different basic types**

The specified identifier was already declared.

The following is an example of this error:

```
int i;
int i();    // error
```

C2372 *identifier* **: redefinition; different types of indirection**

The given identifier was already defined with a different derived type.

The following is an example of this error:

```
extern int *fp;
extern int fp[];  // error
```

C2373 *identifier* **: redefinition; different type modifiers**

The specified identifier was already defined with a different type modifier.

The following is an example of this error:

```
void __pascal func( void );
void __cdecl func( void );  // error
```

C2374 *identifier* **: redefinition; multiple initialization**

The specified identifier was initialized more than once.

A variable can be initialized only once.

The following is an example of this error:

```
int i = 0;
int i = 1;  // error
```

C2375 *function* **: redefinition; different linkage**

The specified function was already declared with a different linkage specifier.

The following is an example of this error:

```
extern void func( void );
static void func( void );  // error
```

C2377 *identifier* **: redefinition; typedef cannot be overloaded with any other symbol**

The specified **typedef** identifier was redefined.

The following is an example of this error:

```
typedef int i;
int i;  // error
```

C2378 *identifier* **: redefinition; symbol cannot be overloaded with a typedef**

The specified identifier was redefined as a **typedef**.

The following is an example of this error:

```
int i;
typedef int i;  //error
```

C2380 **type[s] preceding** *identifier* **(constructor with return type or illegal redefinition of current class name?)**

The specified constructor returned a value or redefined the class name.

A constructor cannot specify a return value.

The following is an example of this error:

```
class C
{
public:
   int C();  // error, specifies an int return
   int C;    // error, redefinition of i
   C();      // OK
};
```

C2390 *identifier* **: incorrect storage class** *specifier*

The storage class was not legal for the specified identifer with global scope.

The default storage class for this context was used in place of the illegal class.

Correct use of storage classes include:

- If the identifier is a function, it should be declared with **extern** storage.
- If the identifier is a formal parameter or local variable, it should be declared with **auto** storage.
- If the identifier is a global variable, it should be declared with no storage class (that is, **auto** storage).

The following example generates this error:

```
register int i;       //error
void main ()
{
   register int j;    //OK
}
```

C2400 **inline syntax error in** *context*; **found** *token*

The given token caused a syntax error within the given context.

C2401 *identifier* **: register must be base in** *context*

The register used within an indirect memory operand must be a base register in this context.

C2402 *identifier* **: register must be index in** *context*

The register used within an indirect memory operand must be an index register in this context.

C2403 *identifier* **: register must be base/index in** *context*

The register used within an indirect memory operand must be either a base or index register in this context.

C2404 *identifier* **: illegal register in** *context*

This register in this context is illegal.

C2405 **illegal short forward reference with offset**

Short forward references must refer only to a label. An additional offset cannot be used.

C2406 *identifier* **: name undefined in** *context*

The identifier used with the **SIZE** or **LENGTH** operator or as a specifier with the member-selection operator (**.**) was not defined.

C2407 **illegal float register in** *context*

An NDP register was specified in an illegal context.

C2408 **illegal type on PTR operator in** *context*

The first parameter of the **PTR** operator was not a legal type specification.

C2409 **illegal type used as operator in** *context*

An illegal type was used within the given context as an operator.

C2410 *identifier* **: ambiguous member name in** *context*

The given identifier within the given context is a member of more than one structure or union.

Use a structure or union specifier on the operand that caused the error. A structure or union specifier is an identifier of type **struct** or **union**, either a **typedef** name or a variable of the same type as the structure or union being referenced. The specifier token must be the left operand of the first member-selection operator (**.**) to use the operand.

C2411 *identifier* **: illegal struct/union member in** *context*

Either the given identifier used with this context is not a member of a visible structure or union or the identifier is not a member of the structure or union specified with the member-selection operator (**.**).

C2412 *identifier* **: case-insensitive label redefined**

The given label was defined more than once within the current function. Change the spelling of the label and its references.

C2413 *token* **: illegal align size**

The alignment size used with the **ALIGN** directive was either missing or outside the valid range.

C2414 **illegal number of operands**

The opcode does not support the number of operands used.

Check an assembly-language reference manual to determine the correct number of operands for this instruction.

It is possible that the instruction is supported with a different number of operands on a newer processor. The problem can be solved by compiling with the /G1 or /G2 option, but then only machines with the newer processor will be able to execute the extended instruction.

C2415 **improper operand type**

The opcode does not use operands of this type.

Check an assembly-language reference manual to determine the correct types of operands for this instruction.

It is possible that the instruction is supported with additional operand types on a newer processor. The problem can be solved by compiling with the /G1 or /G2 option, but then only machines with the newer processor will be able to execute the extended instruction.

C2416 *identifier* **: illegal opcode for processor**

The instruction is legal on a later processor but not on the current processor.

Check an assembly-language reference manual to determine which processors support this opcode.

The problem can be solved by compiling with the /G1 or /G2 option, but then only machines with the newer processor will be able to execute the extended instruction.

C2417 **divide by zero in** *context*

The second parameter to the division operator (**/**) used within the given context is zero.

C2418 *identifier* **: not in a register**

An inline assembler instruction referenced a variable with **register** storage class that was not actually allocated in a register.

To correct this, remove the **register** keyword from the variable definition and make sure that this instruction is legal with a memory operand.

C2419 **mod by zero in** *context*

The second parameter to the **MOD** operator used within the given context is zero.

C2420 *identifier* **: illegal symbol in** *context*

The given identifier is illegal within the given context.

C2421 **PTR operator used with register in** *context*

The **PTR** operator must not be used with a register operand.

C2422 **illegal segment override in** *context*

An illegal segment override was used within the given context.

C2424 *token* **: improper expression in** *context*

The given token was used to form an improper expression within the given context.

C2425 *token* **: nonconstant expression in** *context*

The given token was used to form a nonconstant expression within the given context.

C2426 *token* **: illegal operator in** *context*

The given token must not be used as an operator within the given context. For example, index operators cannot be nested.

C2427 *identifier* **: jump referencing label is out of range**

A branch to the specified label is farther than allowed.

For example, if the following example causes this error:

```
          jz label1
          inc AX

  label1: inc CX
```

then this error can be corrected by either removing excess code between the branch and the label or inverting the jump, as in:

```
        jnz label2
        jmp label1
label2: inc AX

label1: inc CX
```

C2429 *label* : illegal far label reference

FAR PTR cannot be used on jumps or calls to labels. Far references to functions are allowed only if the function has been declared.

C2430 more than one index register in *identifier*

More than one of the specified registers were scaled.

The 32-bit targeted compiler supports scaled indexing, but you can only scale one register.

The following is an example of this error:

```
_asm mov eax, [ebx*2+ecx*4]
```

C2431 illegal index register in *identifier*

The ESP register was scaled or used as both the index and base register.

The SIB encoding for the 80386 processor does not allow scaling by ESP or using ESP as both the index and base register.

The following examples cause this error:

```
_asm mov ax, [ESI+2*ESP]
_asm mov ax, [esp+esp]
```

C2432 illegal reference to 16-bit data in *identifier*

A 16-bit register was used as an index or base register.

The 32-bit targeted compiler does not support referencing 16-bit data, which is supported by the chip using the address size prefix. This means that 16-bit registers cannot be used as index or base registers if you are compiling for 32-bit code.

The following is an example of this error:

```
_asm mov eax, DWORD PTR [bx]
```

C2433 *identifier* : *modifier* not permitted on data declarations

The specified modifier was used for a data declaration.

The **friend**, **virtual**, and **inline** modifiers cannot be used for data declarations.

C2434 *identifier* : cannot convert the default parameter expression to type in formal parameter list

The indicated default parameter could not be converted into the type specified in the function's formal parameter list.

This error can be caused by an incorrect function prototype or by using the wrong value for a default parameter. To use the indicated default parameter, you should define a conversion operator or a constructor that takes a single parameter of the same type as the specified default parameter.

The following is an example of this error. Note that if the conversion operator in `A` is supplied, then there is no error.

```
class A
{
public:
   int i;
} a;

class B
{
public:
   operator int() { return i; }    // conversion operator
   int i;
} b;

void func1( int j = a ) {}  // error, can't convert a to int
void func2( int j = b ) {}  // OK
```

C2436 *identifier* : cannot initialize member functions

A member function of the specified class was initialized.

Unlike variables, functions cannot be initialized. This error can be caused by trying to initialize a function instead of a pointer to the function.

C2437 *identifier* : already initialized

The specified identifier was already initialized.

An object can be declared more than once but can be initialized only once.

C2438 *identifier* : cannot initialize static class data via constructor

A constructor was used to initialize a static member of a class.

Static members should be initialized in a definition outside of the class declaration.

The following example shows how static members are initialized:

```
class X
{
public:
   static const int i;
   static int j;
};
const int X::i = 1;
int X::j = 2;
```

C2439 *identifier* : member could not be initialized

The indicated class, structure, or union member could not be initialized.

This error can be caused by trying to initialize an indirect base class or structure or an inherited member of a class or structure. An inherited member should be initialized by the constructor of the class or structure.

C2440 *identifier* : cannot convert from *type1* to *type2*

The indicated object could not be converted to the required type.

This error can be caused by converting a user-defined type to some other type without supplying a conversion operator.

A conversion operator should be supplied as shown in the following example for the class X, which returns an integer. Note that a parameter type or a return type is not specified.

```
class X
{
public:
   int j;
} x;

class Y
{
public:
   operator int() { return j; }  // conversion operator
   int j;
} y;

void main()
{
   int i;
   i = x;  // error, x cannot be converted to an int
   i = y;  // OK
}
```

C2441 *function* : cannot use inline assembly in p-code function

There was inline assembly-language code included in the given p-code function. Inline assembly cannot be used in a p-code function.

Turn p-code generation off for the given function by using:

```
#pragma optimize( "q", off )
```

or move the inline assembly-language code to a separate module that is not selected for p-code generation.

C2442 p-code expression too complex for setjmp or Catch

The **setjmp** function or Windows **Catch** function was included in a complicated expression in a p-code function. The **setjmp** and **Catch** functions can only be called from p-code with simple statements because these functions have a special meaning when called from the stack-based p-code interpreter.

Simplify the statement containing the **setjmp** or **Catch** function call.

C2443 operand size conflict

The instruction required operands of the same size.

One of the operands must be changed so that both operands have the same size.

The following is an example of this error:

```
short var;

void main()
{
   __asm xchg ax,bl            // error
   __asm mov al,foo            // error
   __asm mov al,BYTE PTR var   // OK
}
```

C2446 *operator* : no conversion between *type1* and *type2*

The specified types could not be converted into the type required by the operator.

The following is an example of this error:

```
class C {};
class D {};

C aC;
D aD;

void main()
{
   aC = aD;  // error
}
```

C2447 missing function header (old-style formal list?)

An open curly brace ({) was found at global scope without a corresponding function header.

This error can be caused by using the old-style C-language formal list.

Check that the function being defined has an appropriate function declaration.

The following is an example of this error:

```
int c;
{}       // error
```

C2448 *identifier* **: function-style initializer appears to be a function definition**

The specified function definition was incorrect.

This error can be caused by using the old-style C-language formal list.

The following is an example of this error:

```
void func(c)
int c;
{}             // error
```

C2450 **switch expression of type** *type* **is illegal**

The specified **switch** expression evaluated to an illegal type.

A **switch** expression must evaluate to an integral type or a class type that has an unambiguous conversion to an integral type.

If the expression evaluates to a user-defined type, a conversion operator needs to be supplied.

The following is an example of this error:

```
class X
{
public:
   int i;
} x;

class Y
{
public:
   int i;
   operator int() { return i; }  // conversion operator
} y;

void main()
{
   int j = 1;
   switch ( x )              // error, x is not type int
   {
      default:  ;
   }
   switch ( y )              // OK
   {
      default:  ;
   }
}
```

C2451 **conditional expression of type *type* is illegal**

The conditional expression evaluated to an illegal type.

The second and third operands (b and c) of the ternary operator (a **?** b **:** c) must both be of the same type or must be able to be converted to a common type by standard conversions.

A conditional expression can be used as an l-value only if the second and third operands are of the same type and both are l-values.

C2458 ***identifier* : redefinition within definition**

The specified class, structure, union, or enumeration was redefined in its own declaration.

The following is an example of this error:

```
class C
{
   enum i { C };   // error
};
```

C2459 ***identifier* : is being defined; cannot add as an anonymous member**

The specified class, structure, or union was redefined in its own scope by a member of an anonymous union.

The following is an example of this error:

```
class C
{
   union { int C; };   // error
};
```

C2460 ***identifier1* : uses *identifier2*, which is being defined**

The given class or structure (*identifier2*) was declared as a member of itself (*identifier1*).

Recursive definitions of classes and structures are not allowed.

The following is an example of this error:

```
class C
{
   C aC;    // error
};
```

C2461 ***class* : constructor syntax missing formal parameters**

The constructor for the given class did not specify any formal parameters. The declaration of a constructor must specify a formal parameter list in parentheses. This list can be null.

Add a pair of parentheses after the *class*::*class* identifier.

The following is an example of this error:

```
class C
{
   C::C;          // error
   C::C();        // OK
};
```

C2462 *identifier* **: cannot define a type while using new**

A type cannot be defined in the operand field of the **new** operator.

Put the type definition in a separate statement.

The following is an example of this error:

```
void main()
{
   new struct S { int i; };     // error
}
```

C2463 **cannot define an anonymous type while using new**

An anonymous type cannot be defined in the operand field of the **new** operator.

Create a named type definition in a separate statement, then use the **new** operator.

The following is an example of this error:

```
void main()
{
   new struct { int x; };  // error
}
```

C2464 *identifier* **: cannot use new to allocate a reference**

The specified reference identifier was allocated with the **new** operator.

Since references are not memory objects, the **new** operator cannot return a pointer to them.

Use the standard variable declaration syntax to declare a reference.

The following is an example of this error:

```
void main()
{
   new ( int& ir );   // error
}
```

C2465 **cannot define an anonymous type inside parentheses**

An anonymous structure, union, or enumerated type was defined inside a parenthetical expression.

This is illegal in C++ programs, as the definition is meaningless in function scope.

C2500 *identifier1* : *identifier2* **is already a direct base class**

The specified class (or structure) appeared more than once in a list of base classes for a derived class.

A class is called a direct base if it is mentioned in the base list. A class is called an indirect base if it is not a direct base but is a base class of one of the classes mentioned in the base list.

A class cannot be specifed as a direct base class more than once. A class can be used as an indirect base class more than once.

The following is an example of this error:

```
class A { };
class B : public A, public A { };    // error
class C : public A { };
class D : public A { };
class E : public C, public D { };    // OK, contains two As
```

C2501 *identifier* **: missing decl specifiers**

The identifier was declared without specifying its type.

This error occurs when a type specifier is omitted in the declaration of an identifier.

C2502 *identifier* **: too many access modifiers on the base class**

The specified base class had more than one access modifier.

A base class or structure can be declared with only one access modifier (**public**, **private**, or **protected**).

The following is an example of this error:

```
class A { };
class B { };
class C : private public A { };     // error
class D : private A { };            // OK
class E : public A, private B { };  // OK
```

C2503 *class* **: base classes cannot contain zero-sized arrays**

The specified base class (or structure) contained a zero-sized array.

An array in a class must have at least one element.

C2504 *class* **: base class undefined**

The specified base class was declared but never defined.

This error can be caused by a missing include file or an external base class that was not declared with the **extern** specifier.

The following is an example of this error:

```
class A;                  // error, A is undefined
class A {};               // OK, A is defined
class B : public A {};    // the error is detected here
```

C2505 *identifier* : is not a legal base class

The specified identifier was not a class (or structure) but was used to derive a class.

A class can be derived only from classes. This error can be caused by naming a variable or type in the base class list.

The following is an example of this error:

```
class B { };
class D : public B { };  // OK, D is derived from B

typedef int I;
class E : public I { };  //  error, I is not a class
```

C2506 *class*::*identifier* : ambiguous

The specified name referred to more than one class member.

To access a base class or structure, an expression must refer to one unique function, object, type, or enumerator. The scope resolution operator (::) can be used to resolve the ambiguity.

The check for ambiguity is done before access control. A private base class containing only private members has the same potential for ambiguity as a public class.

The following is an example of this error:

```
class A
{
public:
   int a;
};

class B
{
private:
   int a;
};

class C : public A, private B { };

C c;
int j = c.a;          // error, could be A::a or B::a
int i = c.A::a;       // OK, resolved ambiguity
```

C2507 *identifier* **: too many virtual modifiers on the base class**

The specified class or structure was declared as **virtual** more than once.

Only one **virtual** modifier can be used for each base class in a list of base classes.

The following is an example of this error:

```
class A {};
class B : virtual virtual public A {}; // error
class C : virtual public A {};         // OK
```

C2508 *identifier* **: access denied**

The specified nonpublic class member could not be accessed.

A class (or structure or union) member that has been declared with private or protected access can only be accessed by member functions of the class.

C2509 *identifier* **: member function not declared in** *class*

The function was not declared in the specified class.

This error can be caused by specifying the wrong class when calling the function.

C2510 *identifier* **: left side of '::' must be a class/struct/union**

The named identifier was not a class, structure, or union.

A class, structure, or union name must appear on the left side of the scope resolution operator (**::**) if any name is used.

C2511 *identifier* **: overloaded member function not found in** *class*

The specified function was not declared for the given parameters.

This error can be caused by a mismatch in the parameter list of the specified function.

C2512 *identifier* **: no appropriate default constructor available**

No default constructor was available for the specified class, structure, or union.

The compiler will supply a default constructor only if user-defined constructors are not provided. If you provide a constructor that takes a nonvoid parameter, then you must also provide a default constructor. The default constructor can be called with no parameters, that is, a constructor with default values for all parameters.

C2514 *class* **: class has no constructors**

The specified class was initialized with parameters for which no constructor was defined.

Constructors must be declared with parameter lists that match every parameter list used to initialize objects of the class, structure, or union.

C2515 *identifier* **: not in class** *class*

The specified identifier was not a member of the given class.

C2517 *identifier* **: right side of '::' is undefined**

The identifier on the right side of the scope resolution operator (**::**) was not defined.

The identifier on the right side of the scope resolution operator must be a defined member of the class, structure, or union found on the left side of the operator. If no class, structure, or union is named, then the identifier on the right side of the operator must be declared with global scope.

C2519 **cannot convert** *type1 ** **to** *type2 **

A pointer to *type1* could not be converted to a pointer to *type2.*

Since *type1* was not derived from *type2*, implicit conversion was not possible.

A pointer to one type generally cannot be implicitly converted to a pointer to another type. Conversion to a **void** * is possible if the size of **void** * is greater than the size of the original pointer.

If a pointer to one type must be converted to a pointer to another type, then explicit conversion should be used.

C2523 *identifier1::~identifier2* **: destructor tag mismatch**

A destructor for the specified class was declared with a different name.

The destructor for a class must have the same name as the class itself and must be preceded by a tilde (~).

The constructor and destructor are the only members of a class that have the same name as the class.

C2524 *identifier* **: destructors must have a void formal parameter list**

The specified destructor had a nonvoid formal parameter list.

A destructor can take only a **void** parameter. Other parameter types are not allowed.

C2527 *identifier* **: array of references must be fully initialized**

Elements of the specified array were not initialized.

An initializer must be supplied for every element of an array of references.

The following is an example of this error:

```
int a, b, c;
int &ai[3];              // error, elements not initialized
int &ai[3] = { a, b, c }; // OK
```

C2528 **illegal pointer to a reference**

A pointer to a reference was declared.

The variable must be dereferenced before a pointer to it can be declared.

C2529 **illegal reference to a reference**

A reference to a reference was declared.

This error can be avoided by using pointer syntax and declaring a reference to a pointer.

C2530 *identifier* **: references must be initialized**

A reference was not initialized when it was declared.

The following cases are the only times a reference can be declared without initialization:

- It is declared with the keyword **extern**.
- It is a member of a class, structure, or union and is initialized in the class's constructor function.
- It is declared as a parameter in a function declaration or definition.
- It is declared as the return type of a function.

C2531 *identifier* **: reference to a bit field illegal**

A reference to the specified bit field was declared.

A reference to a bit field is not allowed.

C2532 *identifier* **: cannot modify references**

The specified reference was changed.

References cannot be modified to refer to another object.

If the reference must be modified, then it should be implemented as a pointer instead.

C2533 *identifier* **: constructors not allowed a return type**

The specified constructor was declared with a return type.

A constructor does not return a value and has no return type. A return type of **void** is not allowed.

The following is an example of this error.

```
class X
{
public:
   void X( void ) { ... };  // error, return type declared
   X( void ) { ... };       // OK, no return type declared
};
```

C2534 *identifier* : **constructor cannot return a value**

The specified constructor cannot return a value.

A constructor cannot return a value of any type, including a return type of **void**.

This error can be eliminated by removing the return statement from the constructor definition.

C2535 *identifier* : **member function already defined or declared**

The specified member function was defined or declared earlier.

This error can be caused by repeating the same formal parameter list in more than one function definition or declaration.

C2536 *identifier1::identifier2* : **cannot specify explicit initializer for arrays**

The specified member of a class, structure, or union could not be initialized.

This error can be caused if a constructor is not available to initialize one or more members of an array. If the size of the array is greater than the number of initializers, then a default constructor must be defined.

Alternatively, this error can be caused by declaring a nonstatic array with the **const** specifier. This kind of array cannot be explicitly initialized.

C2537 *identifier* : **illegal linkage specification**

The indicated linkage specifier was not legal.

This error can be caused by using a linkage specifier that is not supported. Only the “C” linkage specifier is supported.

This error can also be caused by overloading more than one function with “C” linkage. Only one of a set of overloaded functions can be declared with “C” linkage.

C2538 *new* : **cannot specify initializer for arrays**

An initializer was given for the specified array created with the **new** operator.

The **new** operator creates arrays of objects by calling the default constructor for each element of the array. The elements of the array cannot be initialized to distinct values.

C2539 *new* : *identifier* : **no default constructor to initialize array of objects**

A default constructor was not available to initialize an array of objects of the specified class, structure, or union.

Initializing an array of objects requires a default constructor. The default constructor is called separately for each object in the array.

This error can be caused by not defining a default constructor. If any constructor is defined, then the compiler will not generate a default constructor. A default constructor (one that can be called with no parameters) must be explicitly defined in this case.

C2540 **nonconstant expression as array bound**

The specified array bound was not a constant expression.

An array must be declared with a constant bound.

The following example shows an illegal way to declare an array:

```
int i;
int A[i];        // error, i is nonconstant
```

and legal ways to declare an array:

```
const j = 20;
int A[j];        // OK, j is constant
int B[32];       // OK, 32 is a literal
```

C2541 **delete : cannot delete nonpointer objects**

The **delete** operator was used on an object that was not a pointer.

The **delete** operator can only be used on pointers.

The following is an example of this error:

```
void main()
{
   int i;
   delete i;            // error, i is not a pointer
   int* ip = new int;
   delete ip;           // OK
}
```

C2542 *identifier* : **class object has no constructor for initialization**

There was no constructor to initialize the specified object.

A constructor with the same parameter list used in the initialization must be supplied.

This error can also be caused by initializing an object with incorrect parameters.

C2543 **expected ’]’ for operator ’[ ]’**

The left square bracket (]) of the subscripting operator was missing.

This error can be caused by expansion of a macro.

C2544 **expected ’)’ for operator ’( )’**

The left parenthesis of the function call operator was missing.

This error can be caused by expansion of a macro.

C2545 *identifier* **: unable to find overloaded operator**

The specified operator could not be used with the provided operands.

An overloaded operator should be supplied with the required operands.

This error can be caused by using the operator with operands of the incorrect type. Defining a conversion operator or a constructor that takes a single parameter may allow the operator to be used.

C2546 *operator* **: illegal mix of void pointer with nonvoid pointer**

The specified operator was called with incompatible pointer types.

This error can be caused by using an arithmetic operator on a void pointer. Pointer arithmetic can only be done with pointers to objects.

If the operator must be used with a void pointer, then the operator can be overloaded. Alternatively, cast the void pointer into a type that can be used with the specified operator.

C2547 **illegal cast of overloaded function**

A pointer to a function type was converted to an overloaded type.

Conversion of a pointer to a function into a pointer to an overloaded function is not allowed.

The following is an example of this error:

```
int func();
int func( int );
( int (*)() )func;      // error, func is overloaded

int func2();
( void (*)() )func2;    // OK, func2 is not overloaded
```

C2548 *identifier1::identifier2* **: missing default parameter for parameter** *identifier3*

A parameter was missing in a default parameter list.

If a default parameter is supplied anywhere in a parameter list, then all subsequent parameters on the right side of the default parameter must also be defined.

The following is an example of this error:

```
void func( int = 1, int, int = 3);  // error
void func( int, int, int = 3);      // OK
void func( int, int = 2, int = 3);  // OK
```

C2549 **user-defined conversion cannot specify return type**

A user-defined conversion cannot specify a return type.

The following is an example of this error:

```
class X
{
public:
   int operator int() { return value; }   // error
   operator int() { return value; }       // OK
private:
   int value;
};
```

C2551 **void * type needs explicit cast to nonvoid pointer type**

A void pointer was assigned to a nonvoid pointer by implicit conversion.

An explicit cast is necessary to convert a void pointer to a nonvoid pointer.

C2552 *identifier* **: nonaggregates cannot be initialized with initializer list**

The specified identifier was incorrectly initialized.

An initializer list is needed to initialize the following types:

- An array
- A class, structure, or union that does not have constructors, private or protected members, base classes, or virtual functions

These types are known as "aggregates."

C2553 **no legal conversion of return value to return type** *type*

The return value could not be converted to the required type.

You may need to supply a user-defined conversion operator to cast the return value.

C2555 *identifier1::identifier2* **: overriding virtual function differs only by return type**

The specified virtual function and a derived overriding function had identical parameter lists but different return types.

An overriding function in a derived class cannot be redefined to differ only by its return type from a virtual function in a base class.

A function in a derived class or structure overrides a virtual function in a base class only if their names, parameters, and return values are all identical. (If the functions have different parameters, then the compiler treats them as different functions and does not override the virtual function.)

It may be necessary to cast the return value after the virtual function has been called.

C2556 *identifier* **: overloaded functions only differ by return type**

The indicated overloaded functions had different return types but the same parameter list.

Each overloaded function must have a distinctly different formal parameter list.

C2557 *identifier* **: nonpublic members initialized without constructor**

Private and protected members cannot be assigned a value except by the class's member or friend functions. These members should be initialized in the class constructor.

C2558 *identifier* **: no copy constructor available**

No copy constructor was defined to copy the specified class.

A copy constructor is used to initialize an object with the values of another object of the same type, that is, to make a copy of the object.

If no copy constructor is provided, the compiler will generate a default copy constructor. A default copy constructor is not generated by the compiler if any user-defined copy constructor has been defined.

C2559 *identifier* **: no match for specified operator**

There was an attempt to use the **new** operator to call the constructor for the given identifier, but there was no constructor that corresponded to the given ambient memory model or distance.

Make sure that the appropriate constructors have been defined for each memory model or distance being used.

C2561 *identifier* **: function must return a value**

The specified function was declared as returning a value, but the function definition did not contain a return statement.

This error can be caused by an incorrect function prototype. If the function does not return a value, the function should be declared with a **void** return type.

C2562 *identifier* **: void function returning a value**

The indicated function was declared as a void function but returned a value.

This error can be caused by an incorrect function prototype. If the function returns a value, the return type must be specified in the function declaration.

C2563 **mismatch in formal parameter list**

The formal parameter list of a function or pointer to a function did not match those of another function or pointer to a member function, respectively.

The assignment of the functions or pointers could not be made because of incompatible declarations.

The following is an example of this error:

```
void func( int );
void func( int, int );
main()
{
   void *fp();
   fp = func;                 // error
   fp = func( int, int );     // OK
}
```

C2564 **formal/actual parameter mismatch in call through pointer to function**

The formal parameters in the function declaration did not match the actual parameters passed through the pointer to the function.

C2565 *::identifier* **was previously declared as a global function**

The indicated member function had the same name as a function declared earlier with global scope.

C2566 **overloaded function in a conditional expression**

The overloaded function in a conditional expression could not be evaluated.

The following is an example of this error:

```
int f( int );
int f( double );

void main()
{
   if( f );        // error, which f?
   if( f(1) );     // OK
}
```

C2568 *identifier1* **: unable to resolve function overload** *identifier2*

The compiler could not unambiguously determine which of the specified overloaded functions to call.

The actual parameters passed to the function should be cast to match the formal parameters listed in the declaration of one of the specified functions. One parameter match must be better than any of the other possible matches.

C2569 *identifier* **: union cannot be used as a base class**

A class was derived from the specified union.

A union is not allowed to be a base class. If a new user-defined type must be derived from the specified union, the union should be changed to a class or structure.

C2570 *identifier* **: union cannot have base classes**

A union was derived from a class, structure, or union.

The derived user-defined type must be declared as a class or structure.

C2571 *identifier* **: union cannot have virtual function** *identifier*

The specified union was declared to have a virtual function.

Virtual functions can only be used with a class or structure but not with a union. Change the specified union to a class or structure or make it a nonvirtual function.

C2572 *identifier1::identifier2* **: redefinition of default parameter : parameter** *identifier3*

The specified default parameter in a member function was redefined.

A default parameter cannot be redefined. If a new value for the parameter is required, then the default parameter should be left undefined.

C2573 *identifier* **: simple type cast must have exactly one expression**

The specified conversion had a wrong number of actual parameters.

The following is an example of this error:

```
void func()
{
   int i = int();     // error, no actual parameters
   int j = int( 1.0 ) // OK
}
```

C2574 *identifier* **: illegal static destructor declaration**

The specified destructor was declared as **static**.

Destructors cannot be declared as **static**.

C2575 *identifier* **: only member functions and bases can be virtual**

The specified global function or class was declared as **virtual**.

The keyword **virtual** refers only to members of a class or structure. A global function or a class or structure that is not a base class cannot be declared as **virtual.**

C2576 *identifier* **: virtual specifier used for static member function**

A static member function was declared as **virtual**.

The virtual function mechanism relies on the specific object that calls the function to determine which virtual function is used. Since this is not possible for static functions, they cannot be declared as **virtual**.

C2577 *identifier* **: destructor cannot return a value**

The specified destructor returned a value.

A destructor cannot return a value of any type. This error is caused by defining a destructor that returns a value of any type, including a **void** return type.

This error can be eliminated by removing the return statement from the destructor definition.

C2579 *identifier1::identifier2(identifier3)* **: parameter list not sufficiently different from** *identifier1::identifier2(identifier4)*

The parameter lists for the specified functions were not sufficiently different.

The formal parameter lists of overloaded functions must all be different.

C2580 **redefinition of class name** *identifier*

The specified class, structure, or union was already defined.

A class, structure, or union can be defined only once.

C2581 *identifier* **: static operator= function is illegal**

The assignment operator, **operator=( )**, was declared as **static**.

The assignment operator cannot be declared as **static**.

C2582 *identifier* **: operator= function is unavailable**

No assignment operator, **operator=( )**, was defined for the specified class.

If any assignment operator has been defined that takes the class as a parameter, then a default assignment operator will not be generated by the compiler. The assignment operator must be explicitly defined for each class and is not inherited by derived classes.

C2583 *identifier* **: illegal const/volatile this pointer used for constructors/destructors**

A constructor or destructor was declared with a **const** or **volatile** specifier.

The **const** and **volatile** declaration specifiers cannot be used for constructors or destructors.

C2584 *identifier1* **: direct base** *identifier2* **is inaccessible; already a base of** *identifier3*

The specified derived class (*identifier1*) was derived from a direct base class (*identifier3*) that was already a base class of the derived class (*identifier2*).

A class (or structure) cannot use the same class more than once as a direct base class (a class mentioned in the list of base classes for the derived class). A base class can be used more than once as an indirect base class (a class not in the list of base classes).

C2585 **explicit conversion to** *type* **is ambiguous**

The type conversion could produce more than one result.

Ambiguous conversion can result when converting from a class (or structure) type based on multiple inheritance. If the same base class is inherited more than once, the conversion function or operator must specify which of the inherited classes to use in the conversion. The scope resolution operator (::) can be used to do this.

Ambiguous conversion can also be caused when a conversion operator and a constructor making the same conversion have been defined.

C2586 **incorrect user-defined conversion syntax : illegal indirections**

Indirection of a conversion operator is not allowed.

The following is an example of this error:

```
class C
{
   * operator int();   // error, indirection on the operator
};

typedef int * pINT_t;

class D
{
   operator pINT_t();  // OK
};
```

C2587 *identifier* **: illegal use of local variable as default parameter**

A local variable was illegally used as a default parameter.

The following is an example of this error:

```
int i;
void func();
{
   int j;
   extern void func2( int k = j );  // error, local variable
   extern void func2( int k = i );  // OK
}
```

C2588 *::~identifier* **: illegal global destructor**

The specified destructor was not defined for a class, structure, or union.

A destructor can be defined only for a class, structure, or union.

This error can be caused by omitting the name of a class, structure, or union on the left side of the scope resolution operator (**::**).

The following is an example of this error:

```
~F();  // error
```

C2589 *identifier* **: illegal token on right side of '::'**

The token on the right side of the scope resolution operator (**::**) was not legal.

Only a member of a class (or structure or union) can be on the right side of the scope resolution operator if a class name is on the left side. Otherwise, any global identifier can be on the right side.

C2590 *identifier* **: ambiguous user-defined conversions in switch expression**

The specified **switch** expression could not be converted to an integral type.

An expression in a **switch** expression must unambiguously convert to an integral type.

Ambiguous conversion can be caused when a conversion operator and a constructor making the same conversion have been defined.

C2591 *identifier* **: ambiguous user-defined conversions in conditional expression**

The specified conditional expression could not be converted to an integral type.

An expression in a conditional expression must unambiguously convert to an integral type.

Ambiguous conversion can be caused when a conversion operator and a constructor making the same conversion have been defined.

C2592 **no legal conversion of initialization expression to type** *type*

The value of the initialization expression could not be converted to the specified type.

A conversion operator may need to be supplied to make the required cast.

C2595 *identifier* **: qualified name already has a constructor**

A constructor was already defined for the specified nested class, structure, or union.

Only one constructor can be defined for a class, structure, or union.

C2596 *identifier* **: qualified name already has a destructor**

A destructor was already defined for the specified nested class, structure, or union.

Only one destructor can be defined for a class, structure, or union.

C2597 *identifier* **: does not specify an object**

The specified identifier was not a member of a class, structure, or union.

A member access operator (**.** or **–>**) was used to refer to a function that was not defined as a member of the class, structure, or union.

C2598 linkage specification must be at global scope

The linkage specifier was declared in local scope.

The following is an example of this error:

```
void func()
{
   extern "C" int func2();   // error, linkage declared in
   .                         //     block with local scope
   .
   .
}
extern "C" int func( int i );  // OK
```

C2601 functions cannot be defined in local classes

A function definition was found in a class definition.

A function cannot be declared as local to a class, structure, or union.

The following is an example of this error:

```
void main()
{
   class C
   {
      int f() { return 0; }   // error, local function
   };
}
```

C2607 *identifier* **: cannot implicitly convert a** *type1* **to a nonconst** *type2*

The specified type was converted to a nonconst type *type2*.

The initializer for a reference to *type2* must be one of the following:

- An l-value of type *type2*
- A type derived from *type2* for which *type2* is an accessible base

If the reference is to a **const** type, then an object of that type will be created and initialized with the initializer. A temporary object of type *type2* was required but could not be initialized with a nonconst reference *type2***&**.

C2608 illegal reference cast—operand not an l-value

The reference could not be cast.

This error occurs when a temporary copy of the referenced value cannot be generated.

The following is an example of this error:

```
struct C
{
   int mem;
   operator int();
};

struct D
{
   operator C();
   void memfunc();
};

D aD[10];

void D::memfunc()
{
   C aC = ( C& )( aD + 1 );   // error
}
```

C2610 identifier *identifier* can never be instantiated; user-defined constructor is required

The specified class cannot be properly initialzied.

This error occurs when a constructor cannot be created for the class. A user-defined constructor should be defined.

C2611 *token* : illegal following '~' (expected identifier)

The specified token was not an identifier.

The following is an example of this error:

```
class C
{
   C::~operator int();     // error
   ~C();                   // OK, destructor declaration
};
```

C2612 trailing ',' illegal in base/member initializer list

A comma (,) was placed after the last base or member in an initializer list.

The following is an example of this error:

```
class A
{
public:
   int i;
   A( int ia ) : i( ia ), {}; // error, extra comma
   A( int ia ) : i( ia ) {};  // OK
};
```

C2613 trailing ',' illegal in base class list

A comma (,) was placed after the last base in a base class list.

The following is an example of this error:

```
class A {};
class B : public A, {};  // error, extra comma
class C : public A {};   // OK
```

C2614 *class1* : illegal member initialization: *class2* is not a base or member

A class in an initialization list was not a base class or member.

Only a member or base class can be in the initialization list for a class or structure.

The following is an example of this error:

```
class A
{
public:
   int i;
   A( int ia ) : B( i ) {};  // error, B is not a member of A
};
```

C2616 cannot change member's access

The access of the specified identifier was changed outside of its class declaration.

Access to members of a class (or structure or union) can only be specified in the declaration of the class to which they belong.

The following is an example of this error:

```
class A
{
public:
   int i;
};

class B
{
private:
   A::i;  // error, A::i is not a member of B
};
```

C2617 *function* : inconsistent return statement

The specified function did not have a return type declared, and a previous return statement did not supply a value.

The function return type should be declared.

The following is an example of this error:

```
int i;
func()                   // no return type prototype
{
   if( i ) return;       // no return value
   else return( 1 );     // error detected on this line
}
```

C2618 *function* : inconsistent return statement

The specified function did not have a return type declared, and a previous return statement had a different type.

The function return type should be declared.

The following is an example of this error:

```
int i;
func()                   // no return type prototype
{
   if( i ) return( 1 );  // int return value
   else return;          // error detected on this line
}
```

C2619 union *union* : cannot have static member variable *identifier*

The specified union member was declared as **static**.

A union cannot have a static data member.

The following is an example of this error:

```
void func()
{
   union
   {
      static int j; // error, j is static
      int i;        // OK
   };
}
```

C2620 union *union* : member *identifier* has default constructor

The specified union member was declared with a default constructor.

A union member is not allowed to have a default constructor.

The following is an example of this error:

```
class A
{
   A();      // A has a default constructor
};

union U
{
   A a;      // error
};
```

C2621 union *union* : member *identifier* has a copy constructor

The specified union member was declared with a copy constructor.

A union member is not allowed to have a copy constructor.

The following is an example of this error:

```
class A
{
   A( const A& );  // A has a copy constructor
};

union U
{
   A a;            // error
};
```

C2622 **union** *union* **: member** *identifier* **has assignment operator**

The specified union member was declared with an assignment operator, **operator=()**.

A union member is not allowed to have an assignment operator.

The following is an example of this error:

```
class A
{
   operator= ( const A& );  // A's assignment operator
};

union U
{
   ;                        // error
};
```

C2623 **union** *union* **: member** *identifier* **has a destructor**

The specified union member was declared with a destructor.

A union member is not allowed to have a destructor.

The following is an example of this error:

```
class A
{
   ~A();    // A has a destructor
};

union U
{
   A a;     // error
};
```

C2624 *identifier* **: references to void are illegal**

The specified identifier was declared as a reference to **void**.

References to **void** are not allowed.

C2625 **anonymous union did not declare any nonstatic data members**

Unions cannot have static data members.

The following is an example of this error:

```
static union {};         // error
static union { int i; }; // OK
```

C2626 **anonymous union defines protected/private member** *identifier*

The specified member was declared with **protected** or **private** access.

A member of an anonymous union must have **public** access.

The following is an example of this error:

```
void main()
{
   union
   {
   public:
      int i;     // OK, i is public
   protected:
      int j;     // error, j is protected
   private:
      int k;     // error, k is private
   };
}
```

C2627 anonymous union defines member function ***function***

The specified function was declared in an anonymous union.

An anonymous union cannot have member functions.

The following is an example of this error:

```
void main()
{
   union
   {
      int i;
      void func( void );     // error, union is anonymous
   };

   union U
   {
      void func2( void );    // OK
   };
}
```

C2628 ***type1*** **followed by** ***type2*** **is illegal (is a ";" missing?)**

A section of code between the two specified types was incorrect.

This error can be caused by a missing semicolon.

The following is an example of this error:

```
class C
{
public:
   void func( void ) {;}
}                          // semicolon is missing here

void main() {}             // error detected on this line
```

C2629 **unexpected** *token (*

A syntax error made the statement ambiguous.

This error can be caused by mixing declaration and expression syntax.

The following is an example of this error:

```
class B
{
   B( &B );  // error, misplaced &
   B( B& );  // OK
};
```

C2631 *class* **: destructors not allowed a return type**

A destructor in the specified class, structure, or union was declared with a return type.

A destructor cannot be declared with a return type.

The following is an example of this error:

```
class C
{
   int ~C();   // error, returns int
   void ~C();  // error, returns void
   ~C();       // OK
};
```

C2632 *type1* **followed by** *type2* **is illegal**

Two type specifiers had missing code between them.

The following is an example of this error:

```
int float i;   // error
```

C2633 *identifier* **: inline is the only legal storage class for constructors**

The constructor was declared as other than **inline**.

The following is an example of this error:

```
class C
{
   extern C();  // error, not inline
};
```

C2635 **cannot convert an *identifier1** to an *identifier2**; conversion from a virtual base class is implied**

The specified conversion required a cast from a virtual base class to a derived class.

Casting from a virtual base class to a derived class is not allowed.

C2636 **pointer to reference member is illegal**

A pointer to a reference member was declared.

The following is an example of this error:

```
struct S {};
int &S::*prs;        // error
```

C2637 *identifier* **: cannot modify pointers to data members**

The specified pointer to a data member was modified.

The following is an example of this error:

```
struct S {};
int __pascal S::*pms;     // error
```

C2638 *identifier* **: memory-model modifier illegal on pointer to data member**

A memory-model modifier was specifed for a pointer to a data member.

Memory-model modifiers can only be used for pointers to member functions.

The following is an example of this error:

```
class C
{
public:
   int i;
   int j;
   int func();
};

int __near C::* cpi = &C::i;  // error, __near modifier
int C::* cpj = &C::j;         //OK
int (__near C::* cpf)() = &C::func; // OK, not data member
```

C2639 cannot use pointer to member expression *&class::member*—base *class* is inherited as virtual

The pointer to the specified member was illegally inherited as **virtual** from the specifed base class or structure.

Pointers cannot point to members of derived classes that are inherited as virtual members.

The following is an example of this error:

```
struct V
{
   virtual void func();
};

struct D : virtual V
{
   void func();
};

void main()
{
   &D::func;  // error
}
```

C2640 cannot convert a pointer to member across a virtual inheritance path

A pointer to a member was illegally cast to a pointer to a member of a base class that is declared as **virtual**.

The following is an example of this error:

```
class A
{
public:
   int a;
};

class B
{
public:
   int b;
};

class C : virtual public A, public B {};

int C::* cpa = &C::a;          // error, C's A is virtual
int C::* cpb = &C::b;          // OK
```

C2641 illegal pointer to member cast across virtual inheritance path

A pointer to a member was cast to a base class that was inherited using virtual inheritance.

The following is an example of this error:

```
struct V {};
struct A : virtual V {};
int A::*pma;
int V::*pmv = (int V::*)pma;
```

C2642 cast to pointer to member must be from related pointer to member

A pointer to a member was cast to a pointer to a member of a class (or structure) that was not a derived or base class.

The following is an example of this error:

```
class B
{
public:
   int b:
};

class C {};
int C:: cpb = (int C::*)&B::b;  // error

class D : public B
{
public:
   int d;
};
int D:: dpb = (int D::*)&B::b;  // OK, B is a base class of D
int B:: bpd = (int B::*)&D::d;  // OK
```

C2643 illegal cast from pointer to member

A pointer to a member of a class, structure, or union was cast to a different type.

The following is an example of this error:

```
class C
{
public:
   int i;
   operator int*() { return &i; }
};

int C::* cpi = (int*)&C::i; // error
int C::* cpi2 = &C::i;      // OK
```

C2644 basis class *class* for pointer to member has not been defined

A pointer was declared that pointed to a class that was declared but not defined.

The following is an example of this error:

```
class C;
int C::* cp; // error, C has not been declared
class D {};
int D::*dp;  // OK
```

C2645 no qualified name for pointer to member (found ':: *')

The pointer to member declaration did not specify a class.

The declaration of a pointer to a member of a class (or structure or union) must specify the name of the class.

The following is an example of this error:

```
class A {};

int ::* cp;    // error, class not specifed
int B::* bp    // error, B not defined
int A::* ap;   // OK
```

C2646 global anonymous unions must be declared static

The anonymous union had global scope but was not declared as **static**.

The following is an example of this error:

```
union { int i; };         // error, not static
static union { int j; };  // OK
union U { int i; };       // OK, not anonymous
```

C2648 *identifier* : use of nonstatic member as default parameter

The specified nonstatic member was used as a default parameter.

The following is an example of this error:

```
class C
{
public:
   int i;
   static int j;
   void func1( int i = i );  // error, i is not static
   void func2( int i = j );  // OK, uses static j
};
```

C2649 *identifier* **: is not a** *class-key*

The specified class, structure, or union declaration used an incorrect tag.

The following is an example of this error:

```
struct S
{
   int i;
};

class S::i c;   // error
```

C2650 *operator* **: cannot be a virtual function**

The specified operator was declared as **virtual**.

The operators **new** and **delete** cannot be **virtual** because they are static member functions.

The following is an example of this error:

```
class A
{
   virtual void* operator new( unsigned int ); // error
};
```

C2652 *identifier* **: illegal copy constructor : first parameter must not be an** *identifier*

The first parameter in the specified copy constructor was the same type as the class, structure, or union for which it was defined.

A copy constructor for a type can take a reference to the type as the first parameter but not the type itself.

The following is an example of this error:

```
class A
{
   A( A );      // error, takes an A
};

class B
{
   B( B& );     // OK, reference to B
};
```

C2653 *identifier* **: is not a class**

The specified identifier was not a class, structure, or union.

An identifier that was not a class, structure, or union was accessed using incorrect syntax.

The following is an example of this error:

```
int x;
int main()
{
   return x::i; // error
}
```

C2654 *identifier* : attempt to access member outside a member function

The specifed identifier was accessed in a declaration.

Member data can only be accessed in member functions.

This error can be caused by trying to initialize variables in a declaration. A constructor should be used for this purpose.

The following is an example of this error:

```
class A
{
   int i;
   int j = i;  // error, access outside function
   void setj( void ) { j = i ; }  // OK, access in function
   A( int ai )
   {
      i = ai;  // OK, initializes i and j
      j = i;
   }
} a( 1 );
```

C2655 *identifier* : definition or redeclaration illegal in current scope

The specified identifier was redeclared or redefined in nonglobal scope.

An identifier can only be redeclared in global scope.

The following is an example of this error:

```
class A {};
class B
{
public:
   static int i;
};

void main()
{
   A B::i;  // error
}
```

C2656 *function* : function not allowed as a bit field

The specified function was declared as a member of a bit field.

This error can be caused by a syntax error in a constructor initializer list.

C2657 *class::** found at the start of a statement (was a type specified?)

The line began with the specified pointer to member identifier.

This error can be caused by omitting a type specifier in the declaration of a pointer to a member.

The following is an example of this error:

```
class C {};
void main() {
   C::* pmc1;        // error
   void C::* pmc2;   // OK
}
```

C2658 multiple conversions: *type1(type2)* and *type1::operator type2()*

The conversion was ambiguous because it could be done with either the specified constuctor or the specifed conversion operator.

The following is an example of this error:

```
struct A;

struct B
{
   B(A);
   B();
};

struct A
{
   operator B();
};

A a;
B b = B(a);  // error
```

C2659 *operator* : overloaded function as left operand

An overloaded function was on the left side of the specified operator.

The following is an example of this error:

```
int func( int );
int func( double );

void main()
{
   func = 10;  // error
}
```

C2660 *function* **: function does not take** *number* **parameters**

The specified function was called with an incorrect number of actual parameters.

This error can be caused by calling the function with incorrect actual parameters or by omitting a function declaration.

The following is an example of this error:

```
void func( int, int );

void main()
{
   func( 1 );     // error, func( int ) not declared
   func( 1, 0 );  // OK, func( int, int ) was declared
}
```

C2661 *function* **: no overloaded function takes** *number* **parameters**

The specified overloaded function was not declared for the given number of parameters.

This error can be caused by calling the function with incorrect actual parameters or by omitting a function declaration.

The following is an example of this error:

```
void func( int );
void func( int, int );

void main()
{
   func( );      // error, func( void ) was not declared
   func( 1 );    // OK, func( int ) was declared
}
```

C2662 *function* **: no legal conversion for the this pointer**

The required conversion for the **this** pointer was not defined.

A user-defined conversion operator may be required.

The following is an example of this error:

```
class __far C
{
public:
   void func() __near;
} c;

void main()
{
   c.func();  // error
}
```

C2663 *function* : *number* overloads had no legal conversion for the this pointer

The required conversion was not defined for the **this** pointer to the specified overloaded functions.

A user-defined conversion operator may be required.

The following is an example of this error:

```
class __far C
{
public:
   void func() __near;
   void func( int ) __near;
} c;

void main()
{
   c.func();  // error
}
```

C2664 *function* : no legal conversion for parameter *number*

The specified parameter of the specified function could not be converted to the required type.

An explicit conversion may need to be supplied.

The following is an example of this error:

```
class A {} a;

func( int, A );

void main()
{
   func( 1, 1 );  // error, no conversion from A to int
}
```

C2665 *function* : *number1* overloads have no legal conversion for parameter *number2*

The specified parameter of the overloaded function could not be converted to the required type.

A conversion operator or explicit conversion may need to be supplied.

C2666 *identifier* : *number* overloads have similar conversions

The specified overloaded function or operator was ambiguous.

This error is caused by formal parameter lists that are too similar to resolve ambiguity.

An explicit cast of one or more of the actual parameters can resolve the ambiguity.

The following is an example of this error:

```
void func( int, float ) {};
void func( float, int ) {};

func( 1, 1 );       // error, same conversion for each func
func( 1, (float)1 )  // OK
```

C2667 *function* **: none of** *number* **overloads have a best conversion**

The specified overloaded ambiguous function call could not be resolved.

The conversion required to match the actual parameters in the function call to one of the overloaded functions must be strictly better than the conversions required by all of the other overloaded functions.

C2668 *function* **: ambiguous call to overloaded function**

The specified overloaded function call could not be resolved.

An explicit cast of one or more of the actual parameters can resolve the ambiguity.

The following is an example of this error:

```
struct A {};
struct B : A {};
struct X {};
struct D : B, X {};

void func( X, X );
void func( A, B );

 D d;
void main()
{
   func( d, d );        // error, D has an A, B, and X
   func( (X)d, (X)d );  // OK, uses func( X, X )
}
```

C2671 *function* **: static member functions do not have this pointers**

The specified static member function could not access a **this** pointer.

The following is an example of this error:

```
struct S
{
   static S* const func() { return this; }  // error
};
```

C2672 *function* **:** *new/delete* **member functions do not have this pointers**

The specified **new** or **delete** operator did not have a **this** pointer to access.

The following is an example of this error:

```
class C
{
   void* operator new( unsigned int )
   {
      this = 10;      // error
      return 0;
   }
};
```

C2673 *function* **: global functions do not have this pointers**

The specified global function did not have a **this** pointer to access.

The following is an example of this error:

```
void main()
{
   this = 0;  // error
}
```

C2710 **cannot delete a pointer to a const object**

A pointer to an object declared as **const** was illegally deleted using the **delete** operator.

The following is an example of this error:

```
const int* pci;
const int i = 0;

void main()
{
   pci = &i;
   delete pci;  // error, pci points to a const
}
```

C2711 **cannot delete a pointer to a function**

A pointer to a function was illegally deleted using the **delete** operator.

The following is an example of this error:

```
void (*pf)();
void func();

void main()
{
   pf = func;
   delete pf;   // error, pf points to a function
}
```

C2720 *identifier* **:** *specifier* **storage class specifier illegal on members**

A storage class was illegally specified for the given identifier.

The following is an example of this error:

```
struct S
{
   static int i;
};
static S::i;   // error
```

C2721 *specifier* **: storage class specifier illegal between operator keyword and type**

A type conversion cannot specify a storage class. User-defined type conversions apply to all storage classes.

C2722 *::operator* **: illegal following operator command; use operator** *operator*

Either `::new` or `::delete` was redefined using the **operator** statement. Since the **new** and **delete** operators are global, the scope resolution operator (**::**) is meaningless in this context.

Remove the scope resolution operator preceding the given operator.

C2730 *class* **: cannot be a base class of itself**

Recursive base classes are illegal.

Remove the base class specifier, or specify another class as a base class.

C2732 **linkage specification contradicts earlier specification for** *function*

The specified function was already declared with a different linkage specifier.

This error can be caused by different linkage specifiers given in include files.

Change the **extern** statements so that the linkages agree.

The following is an example of this error:

```
extern void func( void );      // implicit C++ linkage
extern "C" void func( void );  // error
```

C2733 **second C linkage of overloaded function** *function* **not allowed**

More than one overloaded function was declared with C linkage.

When using C linkage, only one form of a given function can be made external.

Since overloaded functions have the same undecorated name, they cannot be used with C programs.

C2734 *identifier* **: nonextern const object must be initialized**

The specified identifier was declared as **const** but was not initialized.

An identifier must be initialized when declared as **const** unless it is declared as **extern**.

The following is an example of this error:

```
const int j;            // error
extern const int i;     // OK, declared as extern
```

C2735 *keyword* keyword is not permitted in formal parameter type specifier

The specified keyword was illegal in its context.

The following is an example of this error:

```
void main()
{
   int *myint;

   myint = new static int;     // error
   myint = new typedef int;    // error
   myint = new auto int;       // error
   myint = new register int;   // error
}
```

C2736 *keyword* keyword is not permitted in cast

The specified keyword was illegally used in a cast.

The following is an example of this error:

```
int main()
{
   return (virtual) 0;   // error
}
```

C2737 *class1* : base class *class2* must be exported

A derived class *class1* was exported, but its base class *class2* was not.

If *class1* is exported, *class2* must also be exported.

C2800 operator *operator* cannot be overloaded

The specified operator was overloaded.

The following operators cannot be overloaded: class member access operator (**.**), pointer to member operator (**.***), scope resolution operator (**::**), conditional expression operator (**?:**), and **sizeof** operator.

The following is an example of this error:

```
class C
{
   operator:: ();  // error :: is overloaded
};
```

C2801 operator *symbol* must be a nonstatic member

The specified overloaded operator was not a member of a class, structure, or union, and/or was declared as **static**.

The following operators can only be overloaded in class scope as nonstatic members: assignment operator '=', class member access operator '–>', subscripting operator '[]', and function call operator '()'.

The following is an example of this error:

```
operator[]();           // error, not a member

class A
{
   static operator->();  // error, static
   operator()();         // OK
};
```

C2802 static member operator *symbol* has no formal parameters

The specified static member operator had a void formal parameter list.

An operator declared by a static member function must take at least one parameter.

The following is an example of this error:

```
class A
{
   static operator+ ();     // error, void parameter list
   static operator* ( A& ); // OK
};
```

C2803 operator *symbol* must have at least one formal parameter of class type

The specified overloaded operator was declared without at least one class-type parameter.

The following is an example of this error:

```
int operator+( int, int );  // error
```

C2804 binary operator *symbol* has too many formal parameters

The specified overloaded binary operator was declared with more than one parameter.

The following is an example of this error:

```
class X
{
   X operator+ ( X , X );  // error, two parameters
   X operator+ ( X );      // OK, one parameter
};
```

C2805 **binary operator** ***symbol*** **has too few formal parameters**

The specified binary operator was called with a void parameter list.

The following is an example of this error:

```
class X
{
public:
   X operator< ( void );  // error, must take one parameter
   X operator< ( X );     // OK
};
```

C2806 **operator** ***symbol*** **has too many formal parameters**

The specified overloaded operator was declared with too many parameters.

The following is an example of this error:

```
class X
{
public:
   X operator++ ( int, int ); // error, more than 1 parameter
   X operator++ ( int );      // OK
} ;
```

C2807 **second formal parameter to postfix operator** ***symbol*** **must be int**

The second parameter to the specified postfix operator must be declared to be of type **int**.

The following is an example of this error:

```
class X
{
public:
   X operator++ ( X );    // error, nonvoid parameter
   X operator++ ( int );  // OK, int parameter
} ;
```

C2808 unary operator *symbol* has too many formal parameters

The specified overloaded unary operator was incorrectly declared with a nonvoid parameter list.

The following is an example of this error:

```
class X
{
public:
   X operator! ( X );     // error, nonvoid parameter list
   X operator! ( void );  // OK
};
```

C2809 operator *symbol* has no formal parameters

The specified operator was incorrectly declared with a void parameter list.

The following is an example of this error:

```
int operator+ ();  // error
```

C2810 second formal parameter for operator delete must be unsigned int

An overloaded two-parameter **operator delete** must have its second formal parameter declared as an unsigned integer.

The second formal parameter in this form of the **operator delete** specifies the size of the object being deleted, and therefore it must be of type **size_t**, that is, an unsigned integer.

C2811 too many formal parameters for based form of operator delete

There were too many formal parameters for the **operator delete** for based pointers.

When used on based pointers, the **operator delete** takes either two or three actual parameters.

The following are the prototypes for the based form of the **operator delete**:

```
void operator delete ( __segment, void __based( void ) * );
void operator delete ( __segment, void __based( void ) *, \
unsigned int );
```

C2812 second formal parameter required for based form of operator delete

There were too few formal parameters for the **operator delete** for based pointers.

The following are the prototypes for the based forms of the **operator delete**:

```
void operator delete ( __segment, void __based( void ) * );
void operator delete ( __segment, void __based( void ) *, \
unsigned int );
```

C2813 **too many formal parameters for nonbased operator delete**

Too many formal parameters were provided for the nonbased form of the **operator delete**. The **operator delete** can take one or two formal parameters.

The following are the prototypes for the nonbased forms of the **operator delete**:

```
void operator delete ( void __near * );
void operator delete ( void __near *, unsigned int );
void operator delete ( void __far * );
void operator delete ( void __far *, unsigned int );
void operator delete ( void __huge * );
void operator delete ( void __huge *, unsigned int );
```

C2814 **second formal parameter for based form of operator delete must be __based(void) ***

The second formal parameter for an overridden based form of the **operator delete** must be of type **__based(void) ***.

The following are the prototypes for the based forms of the **operator delete**:

```
void operator delete ( __segment, void __based( void ) * );
void operator delete ( __segment, void __based( void ) *, \
unsigned int);
```

C2815 **first actual parameter for based form of operator delete must be __segment**

The first formal parameter for an overridden based **operator delete** must be of type **__segment**.

The following are the prototypes for the based forms of the **operator delete**:

```
void operator delete ( __segment, void __based( void ) * );
void operator delete ( __segment, void __based( void ) *, \
unsigned int);
```

C2816 **alternative form of operator delete must be a member**

The two-parameter form of the **operator delete** can be redefined only within a class, structure, or union. Global redefinitions of **operator delete** are allowed to have only one parameter.

C2817 return type for operator delete must be void

A redefinition of the **operator delete** contained a return statement that returned a type other than **void**. The return value for the **operator delete** must be **void**.

C2818 incorrect return type for operator '–>'

A redefinition of the class member access operator (–>) contained a return statement that did not return an appropriate type.

The class member access operator (–>) must return one of the following:

- A pointer to a class, structure, or union
- A reference to a class, structure, or union where the –> operator is defined
- An instance of a class, structure, or union where the –> operator is defined

C2819 recursive return type for operator '–>'

A redefinition of the class member access operator (–>) contained a return statement that was recursive.

To use recursion in a redefinition of the class member access operator, move the recursive routine to a separate function that is called from the operator override function.

C2820 second formal parameter required for based form of operator new

There were too few formal parameters for the **operator new** for based pointers.

This is the prototype for the based form of the **operator new**:

```
void __based( void ) * operator new ( __segment, unsigned int );
```

C2821 first formal parameter for operator new must be unsigned int

The first formal parameter of the near or far forms of the **operator new** must be an **unsigned int**.

The following are the prototypes for the near and far forms of the **operator new**:

```
void __near * operator new ( unsigned int );
void __far * operator new ( unsigned int );
```

C2822 second formal parameter for huge form of operator new must be unsigned int

The second formal parameter of the huge form of the **operator new** must be an **unsigned int**.

This is the prototype for the huge form of the **operator new**:

```
void __huge * operator new ( unsigned long, unsigned int );
```

C2823 **return type for based form of operator new must be __based(void) ***

A redefinition of the based form of the **operator new** contained a return statement that returned a type that was not a based pointer. The return value for the **operator new** must be **void __based(void) ***.

C2824 **return type for operator new must be void ***

A redefinition of the nonbased form of the **operator new** contained a return statement that returned a type that was not a void pointer. The return value for the **operator new** must be **void**.

C2825 **first formal parameter for huge form of operator new must be unsigned long**

The first formal parameter of the huge form of the **operator new** must be an **unsigned long**.

This is the prototype for the huge form of the **operator new**:

```
void __huge * operator new ( unsigned long, unsigned int );
```

C2826 **second formal parameter required for huge form of operator new**

There were too few formal parameters for the **operator new** for huge objects.

This is the prototype for the huge form of the **operator new**:

```
void __huge * operator new ( unsigned long, unsigned int );
```

C2827 **operator *symbol* cannot be globally overridden with unary form**

The given operator cannot have a unary form outside of an object.

Make sure that the overloaded operator is local to the object or choose an appropriate unary operator to overload.

C2828 **operator *symbol* cannot be globally overridden with binary form**

The given operator cannot have a binary form outside of an object.

Make sure that the overloaded operator is local to the object or choose an appropriate binary operator to overload.

C2829 **operator *symbol* cannot have a variable parameter list**

The given operator cannot have a variable parameter list.

Only the **new** and parentheses operators can take variable parameters.

C2830 **only placement parameters to operator new can have default values**

The standard formal parameters for the **operator new** cannot be given default values. Only user-defined placement parameters can specify defaults.

Since the compiler automatically passes values to the standard formal parameters, a default value is meaningless.

C2831 **operator *symbol* cannot have default parameters**

Default parameters cannot be specified for the given operator.

Only the **new**, assignment (=), and left parenthesis operators can have default values.

C2833 **operator *symbol* is not a recognized operator or type**

The operator command was not followed by either an operator (to override) or a type (to convert).

Make sure that the operator command is followed by an operator or a type.

C2834 **operator *symbol* must be globally qualified**

The given operator cannot be local to a class. Since the **new** and **delete** operators are tied to the class in which they reside, the scope resolution operator (**::**) cannot be used to select a version of the operator from a different class.

To implement multiple forms of the **new** or **delete** operator, create a version of the operator that takes extra formal parameters.

C2835 **user-defined conversion *type* takes no formal parameters**

A user-defined type conversion was declared as having one or more formal parameters. User-defined type conversions cannot take formal parameters.

Remove the formal parameters or choose an operator to overload.

C2836 **cannot export *identifier*; a previous declaration did not export it**

The given identifier was declared to be exported, but a previous declaration did not export it.

All declarations of a given identifier must be either external or nonexternal.

The following is an example of this error:

```
extern int i;          // i not exported
int    __export i;     // error, i exported
```

C2837 *identifier* **: illegal local static variable in exported inline function**

There was an attempt to declare a local static variable inside an external inline function. External inline functions cannot declare local static variables.

C2850 **#pragma hdrstop cannot be nested in a function or definition**

The **hdrstop** pragma cannot be placed inside the body of a function or definition.

Move the **#pragma hdrstop** statement to an area that is not contained in curly braces or parentheses.

C2851 **#pragma hdrstop required for /Yu command-line option without filename**

The /Yu (use precompiled headers) command-line option did not specify the name of a precompiled header file, and there was no **#pragma hdrstop** statement.

This error can be avoided by specifying a filename after the /Yu command-line option or by using **#pragma hdrstop** in the source file.

C2852 *filename* **is not a valid precompiled header file**

The given filename is not a precompiled header file.

Make sure that all /Yu command-line option and **#pragma hdrstop** statements specify valid precompiled header files. The compiler assumes the .PCH extension if none is provided.

This error is caused by giving the filename of a file that is not a precompiled header, such as an .HPP file.

C2853 *filename* **is not a precompiled header file created with this compiler**

The given precompiled header is not compatible with this version of the compiler.

Recompile the program or the header with the same version of the compiler.

This error can be caused by mixing 16-bit and 32-bit source files and precompiled headers.

C2854 **syntax error in #pragma hdrstop**

The **#pragma hdrstop** statement gave an invalid filename.

The **hdrstop** pragma is followed by an optional filename enclosed in parentheses and quotation marks, as in:

```
#pragma hdrstop( "source\pchfiles\myheader.pch" )
```

The precompiled header filename cannot be enclosed in angle brackets (<>).

Compiler Warning Messages

Number **Compiler Warning Message**

C4000 **UNKNOWN WARNING**
Contact Microsoft Product Support Services
Level 1

The compiler detected an unknown error condition.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

C4001 **nonstandard extension** ***extension*** **was used**
Levels 1, 2, and 4

The given nonstandard language extension was used when the /Ze option was specified.

In C, this is usually a level 4 warning, while in C++, this is always a level 2 warning. When compiling C with either the /f or /qc fast-compilation command-line options, a function pointer cast to a data type will produce a level 1 warning.

If the /Za option has been specified, this condition generates a syntax error.

C4002 **too many actual parameters for macro** ***identifier***
Level 1

The number of actual parameters specified with the given identifier was greater than the number of formal parameters given in the macro definition of the identifier.

The additional actual parameters were collected but ignored during expansion of the macro.

C4003 not enough actual parameters for macro *identifier*

Level 1

The number of actual parameters specified with the given identifier was less than the number of formal parameters given in the macro definition of the identifier.

When a formal parameter is referenced in the definition and the corresponding actual parameter has not been provided, empty text is substituted in the macro expansion.

C4004 incorrect construction after defined

Level 1

The **defined** operator was incorrectly terminated, causing a warning or error to appear when the remainder of the line following **defined** was compiled.

The following example generates this warning and a fatal error:

```
#if defined( ID1 ) || ( ID2 )
```

The compiler assumed that the identifier `ID1` was the only operand for the **defined** operator. The rest of the line could not be parsed.

The following avoids this problem:

```
#if defined( ID1 ) || defined( ID2 )
```

C4005 *identifier* : macro redefinition

Level 1

The given identifier was defined twice. The compiler used the second macro definition.

This warning can be caused by defining a macro on the command line and in the code with a **#define** directive. It also can be caused by macros imported from include files.

To eliminate the warning, either remove one of the definitions or use an **#undef** directive before the second definition.

C4006 #undef expected an identifier

Level 1

The name of the identifier whose definition was to be removed was not given with the **#undef** directive. The **#undef** directive was ignored.

C4007 *identifier* : must be *attribute*

Level 2

The attribute of the given function was not explicitly stated. The compiler forced the attribute.

For example, the function **main** must have the **__cdecl** attribute.

C4008 *identifier* **:** *attribute* **attribute ignored**
Levels 2 and 3

The **__fastcall**, **static**, or **inline** attribute for the given identifier was ignored.

Data cannot be defined with the **__fastcall** attribute, and the **main** function cannot be defined with the **static** or **inline** attribute.

This is a level 2 warning for data and a level 3 warning for functions.

C4009 **string too big; trailing characters truncated**
Level 1

A string exceeded the compiler limit of 2047 on string size. The excess characters at the end of the string were truncated.

To correct this problem, break the string into two or more strings.

C4010 **single-line comment contains line-continuation character**
Level 3

A single-line comment (introduced by //) contains a line-continuation character (\). The line continuation character was ignored.

A single-line comment causes the compiler to ignore the rest of the physical line; the compiler does not consider line-continuation characters.

C4013 *function* **undefined; assuming extern returning int**
Level 3

An undefined function was called. The compiler assumed an external function that returned an **int**.

Make sure that the function name is spelled correctly. All external functions should be prototyped as **extern**.

C4014 **concatenating mismatched wide strings**
Level 1

A wide string literal was concatenated with a standard string literal.

Make sure that all wide string literals are prefixed with an "L" character, as in:

```
char   hello[] = L"Hello, " "world"   // Warning
char   hello[] = L"Hello, " L"world"  // OK
```

C4015 *identifier* **: bit-field type must be integral**
Level 1

The given bit field was not declared as an integral type. The compiler assumed the base type of the bit field to be **unsigned**.

Bit fields must be declared as **unsigned** integral types.

C4016 *function* **: no function return type; using int as default**
Level 3

Since the given function had not yet been declared or defined, the return type was unknown. A default return type of **int** was assumed.

This warning may be avoided by adding a prototype for the given function.

C4017 **cast of int expression to far pointer**
Level 1

The compiler extended the **int** expression to a 4-byte value.

A far pointer represents a full segmented address; casting an **int** value to a far pointer may produce an address with a meaningless segment value.

C4018 *expression* **: signed/unsigned mismatch**
Level 3

There was an attempt to compare a **signed** and **unsigned** number. The **signed** value was converted to an **unsigned** type for the comparison.

When performing an equal (==) or not equal (**!=**) comparison between **signed** and **unsigned** types, cast one type to the other to ensure proper comparison.

C4019 **empty statement at global scope**
Level 4

The compiler found a semicolon that was not preceded by a statement. This warning is generated only in global scope.

This warning can be avoided by removing the extra semicolon.

The following example demonstrates this warning:

```
#define declint( varname ) int #varname;

declint( a );     //Warning, int a;;

declint( b )      //OK, int b;
```

C4020 *function* **: too many actual parameters**
Level 1

The number of actual parameters specified in a function call was greater than the number of formal parameters specified in the function prototype or function definition.

The extra actual parameters were passed according to the calling convention used on the function.

C4021 *function* **: too few actual parameters**
Level 1

The number of actual parameters specified in a function call was less than the number of formal parameters specified in the function prototype or function definition.

Only the provided actual parameters were passed. If the called function references a variable that was not passed, the results are undefined.

C4022 *function* **: pointer mismatch for actual parameter** *number*
Level 1

The pointer type of the given actual parameter was different from the pointer type specified in the formal parameter list or function definition.

The actual parameter was passed without change. Its value will be interpreted as a pointer within the called function.

C4023 *function* **: based pointer passed to unprototyped function : parameter** *number*
Level 1

There was an attempt to pass a based pointer to an unprototyped function.

When using a memory model with near data, only the offset portion of a based pointer is passed to an unprototyped function. If the function expects a far pointer, the resulting code will be wrong.

In all memory models, if the function is defined to take a based pointer with a different base, the resulting code may be unpredictable.

This warning can be avoided by using a function prototype containing a reference to the proper base.

C4024 *function* **: different types for formal and actual parameter** *number*
Level 1

The type of the given actual parameter in a function call did not agree with the type given for the formal parameters in the function prototype or definition.

The actual parameter will be passed without change. The function will convert the parameter's type to the type expected by the function.

C4025 *function* **: based pointer passed to function with variable arguments: parameter** *number*
Level 1

A based pointer cannot be passed to the **varargs** part of a function without losing what it is based upon.

When using a memory model with near data, only the offset portion of a based pointer is passed to an unprototyped function. If the function expects a far pointer, the resulting code will be wrong.

To pass a based pointer to a function with variable arguments, cast the based pointer to a far pointer.

C4026 **function declared with formal parameter list**
Level 1

The function was declared to take formal parameters, but the function definition did not have formal parameters.

Subsequent calls to this function will assume that the function takes no actual parameters.

C4027 **function declared without formal parameter list**
Level 1

The function was declared to take no formal parameters (the formal parameter type list consisted of the keyword **void**), but either formal parameters were given in the function definition or actual parameters were given in a call to the function.

Subsequent calls to this function will assume that the function takes actual parameters of the types found in the function declaration or call.

C4028 **actual parameter** *number* **different from declaration**
Level 1

The type of the given actual parameter did not agree with the corresponding formal parameter in the declaration or definition.

The type in the original declaration was used.

C4029 **declared formal parameter list different from definition**
Level 1

The types of formal parameters given in the function declaration did not agree with the types of the formal parameters given in the function definition.

The formal parameter list of the definition was used.

C4030 **first formal parameter list longer than the second list**
Level 1

A function was declared more than once with different formal parameters.

The formal parameters given in the first declaration were used.

C4031 **second formal parameter list is longer than the first list**
Level 1

A function was declared more than once with different formal parameters.

The formal parameters given in the first declaration were used.

C4033 *function* **must return a value**
Level 1

The given function did not return a value.

Only functions with a return type of **void** can use the return command without an accompanying return value.

An undefined value will be returned when this function is called.

C4034 **sizeof returns 0**
Level 1

The **sizeof** operator was applied to an operand that yielded a size of zero.

This warning indicates that the operand of the **sizeof** operator was an empty structure, union, class, or enumerated type, or was of type **void**.

C4035 *function* **: no return value**
Level 3

A function did not include a return statement.

This warning is generated for all functions (including **main**) that do not have return statements. Functions with a return type of **void** will not generate this warning.

An undefined value will be returned when this function is called.

C4036 **unnamed** *type* **as actual parameter**
Level 1

The type of the structure, union, enumerated type, or class being passed as an actual parameter was not given.

The formal parameter in the generated function prototype will be commented out.

This warning occurs only when using the /Zg option for generating function prototypes and can be avoided by providing the appropriate type explicitly.

C4037 **conflicting ambient class modifiers**
Level 1

More than one ambient class modifier was specified for a class, structure, or union.

Only one ambient class modifier can be specified for a class, structure, or union.

C4038 *identifier* **: illegal ambient class modifier**
Level 1

The given identifier was not a valid ambient class modifier.

Valid ambient class modifiers are **__far**, **__near**, and the names of memory models.

C4039 **ambient class modifier on reference ignored**
Level 1

The ambient class modifier on a reference to a class, structure, or union was ignored.

An ambient class modifier can be used only in the declaration and definition of a class, structure, or union.

C4040 **memory attribute on** *identifier* **ignored**
Level 1

The **__near**, **__far**, **__huge**, or **__based** keyword has no effect in the declaration of the given identifier and was ignored.

One cause of this warning is a huge array that is not declared globally. Declare huge arrays outside of the **main** function.

C4041 **illegal modifier for the this pointer**
Level 1

The program attempted to change the **this** pointer to an illegal type.

The **this** pointer can be overloaded to **const**, **volatile**, **__near**, **__far**, or **__huge**.

C4042 *identifier* **: has bad storage class**
Level 1

The storage class specified for the identifier could not be used in this context.

A default storage class for this context was used in place of the illegal storage class. The storage class was selected using the following rules:

- If the identifier was a function, the compiler assumed **extern** class.
- If the identifier was a formal parameter or local variable, the compiler assumed **auto** class.
- If the identifier was a global variable, the compiler assumed the variable was declared with no storage class.

C4043 **function specifier used more than once**
Level 1

A function specifier was used more than once for a symbol.

The second function specifier was ignored.

The following example generates this warning:

```
inline inline void  no_op( void ) { };
```

C4044 **specifier __huge on *identifier* ignored; can only be applied to array**
Level 1

The compiler ignored the **__huge** memory attribute for the given identifier. Only arrays can be declared with the **__huge** memory attribute. For pointers, **__huge** must be used as a modifier, not as a memory attribute.

C4045 *identifier* **: array bounds overflow**
Level 1

Too many initializers were present for the given array. The excess initializers were ignored.

Make sure that the array elements and initializers match in size and quantity.

C4046 *identifier* **: unsized array treated as __far**
Level 1

An unsized array was allocated in the far data segment.

This warning occurs under the /Gx command-line option when initializing an unsized array of characters with multiple elements, as in:

```
char   StringArray[] = {"Windows", "Windows", "Windows"};
```

Since no explicit size was given, the data size threshold cannot be used.

C4047 *operator* **: different levels of indirection**
Level 1

An expression involving the specified operator had inconsistent levels of indirection.

If both operands are of arithmetic type or if both are not (such as array or pointer), then they are used without change. However, the compiler may DS-extend one of the operands if one is **__far** and the other is **__near**. If one operand is arithmetic, but the other is not, the arithmetic operator is converted to the type of the other operator.

For example, the following code generates this warning but is compiled without change:

```
char **p;
char *q;
p = q;   // Warning
```

C4048 different declared array subscripts
Level 1

An expression involved pointers to arrays of different size. The pointers were used without conversion.

This warning can be avoided by explicitly casting the arrays to the same or equivalent type.

C4049 *operator* **: indirection to different types**
Level 1

The pointer expressions used with the given operator had different base types.

The pointer expressions were used without conversion.

For example, the following code generates this warning:

```
struct ts1 *s1;
struct ts2 *s2;
s2 = s1;   // Warning
```

C4050 *operator* **: different code attributes**
Level 4

The function-pointer expressions used with the given operator had different code attributes, one of which is either **__export** or **__loadds**.

This is a warning and not an error because **__export** and **__loadds** affect only entry sequences and not calling conventions.

C4051 type conversion; possible loss of data
Level 2

Two data items in an expression had different base types, causing the type of one item to be converted. During the conversion, a data item was truncated.

This warning can be avoided by casting the data items to the appropriate type.

C4052 function declarations different; one contains variable arguments
Level 1

One declaration of the function did not contain variable arguments, but another did.

The declaration containing the variable arguments was used.

C4053 one void operand for '?:'
Level 1

An expression of type **void** was used as an operand.

The expression was evaluated using an undefined value for the **void** operand.

C4054 **function pointer cast to a data pointer**
Level 1

A function pointer was cast to a data pointer.

Make sure that the correct cast is being used.

This condition generates an error with the C compiler with the /Za ANSI-compatibility command-line option or under the medium memory model.

C4055 **data pointer cast to a function pointer**
Level 1

A data pointer was cast to a function pointer.

Make sure that the correct cast is being used.

This condition generates an error with the C compiler with the /Za ANSI-compatibility command-line option or under the compact memory model.

C4067 **unexpected characters following *directive* directive—newline expected**
Level 1

Extra characters followed a preprocessor directive and were ignored. This warning appears only when compiling with the /Za ANSI-compatibility option.

For example, the following code generates this warning:

```
#endif    NO_EXT_KEYS
```

To remove the warning, compile with /Ze or use comment delimiters, as in:

```
#endif    /* NO_EXT_KEYS */
```

C4068 **unknown pragma**
Level 1

The compiler did not recognize a pragma and ignored it.

Make sure that the given pragma is allowed by the type of compiler being used.

C4071 ***function*** **: no function prototype given**
Level 2

The given function was called before the compiler found the corresponding function prototype.

The function will be called using the default rules for calling a function without a prototype.

C4072 *function* **: no function prototype on __fastcall function**
Level 1

The given **__fastcall** function was called before the compiler found the corresponding function prototype.

The function will be called using the default rules for calling a function without a prototype.

C4073 **initializers put in library initialization area**
Level 3

The **#pragma init_seg(lib)** statement was used. The library initialization area should be used only by third-party library developers.

This warning is informational.

C4074 **initializers put in compiler reserved initialization area**
Level 1

The **#pragma init_seg(compiler)** statement was used. The compiler initialization area is reserved by Microsoft.

Code in this area may be executed before C run-time library initialization.

C4075 **initializers put in unrecognized initialization area**
Level 1

An explicit segment name was used as the actual parameter for **#pragma init_seg**. No compiler or C run-time library support was provided.

User-defined startup code must be supplied to execute the initializers.

C4076 *type* **: may be used on integral types only**
Level 1

The **signed** or **unsigned** type modifier was used with a nonintegral type.

The given qualifier was ignored.

The following example generates this warning:

```
unsigned double x;
```

C4077 **unknown check_ stack option**
Level 1

An unknown option was given with the old form of the **check_ stack** pragma. With the old form, the argument to the pragma must be empty, +, or –.

The current state of stack checking was left unchanged.

For example, the following generates this warning:

```
#pragma check_stack yes
```

The following code corrects this situation:

```
#pragma check_stack +    // Old form
```

or

```
#pragma check_stack ( on ) // New form
```

C4078 **case constant *value* too big for the type of the switch expression**
Level 1

A value appearing in a **case** statement was larger than the size of the type used in the **switch** expression. The compiler cast the type of the **case** constant to that of the **switch** expression.

A problem can occur when two **case** constants have different values before being cast but the same value afterward.

C4079 **unexpected token *token***
Level 1

An unexpected separator token was found in the argument list of a pragma.

The remainder of the pragma was ignored.

The following example generates this warning:

```
#pragma comment( "Kilroy was here", )
```

C4080 **expected identifier for segment name; found *token***
Level 1

The first actual parameter given to the **alloc_ text** pragma was missing a segment name. This happens if the first token in the argument list was not an identifier.

The pragma was ignored.

C4081 **expected a comma; found *token***
Level 1

A comma (,) was missing between two arguments of a pragma.

The pragma was ignored.

The following example generates this warning:

```
#pragma optimize( "l" on )
```

C4082 **expected an identifier; found** *token*
Level 1

An identifier was missing from the argument list.

The remainder of the pragma was ignored.

C4083 **expected ’(’; found** *token*
Level 1

A left parenthesis was missing from a pragma’s argument list.

The pragma was ignored.

The following example generates this warning:

```
#pragma check_pointer on )
```

C4084 **expected a pragma keyword, found** *token*
Level 1

The *token* following **#pragma** was not recognized as a directive.

The pragma was ignored.

The following example generates this warning:

```
#pragma (on)
```

C4085 **expected** *on* **or** *off*
Level 1

The pragma expected an on or off parameter, but the specified parameter was unrecognized or missing.

The pragma was ignored.

The following example generates this warning:

```
#pragma optimize( "t", maybe )
```

C4086 **expected ’1’, ’2’, ’4’, or ’8’**
Level 1

The pragma expected a parameter of either 1, 2, 4, or 8, but the specifed parameter was not recognized or was missing.

The **pack** pragma will not accept an 8 when generating 16-bit code.

The following example generates this warning:

```
#pragma pack( 3 )
```

C4087 *function* **: declared with void parameter list**
Level 1

The given function was declared with no formal parameters, but a call to the function specified actual parameters.

The extra parameters were passed according to the calling convention used on the function.

The following example generates this warning:

```
int f1( void );
f1( 10 );
```

C4088 *function* **: pointer mismatch in actual parameter** *number***, formal parameter** *number*
Level 1

The actual parameter passed to the given function had a different level of indirection than the corresponding formal parameter.

The actual parameter will be passed without change. Its value will be interpreted as a pointer by the called function.

C4089 *function* **: different types in actual parameter** *number***, formal parameter** *number*
Level 1

The actual parameter passed to the given function had a different type than the corresponding formal parameter.

The actual parameter will be passed without change. The function will cast the actual parameter to the type specified in the function definition.

C4090 *operation* **: different const or volatile qualifiers**
Level 1

The given operation was performed on a variable that was defined as being **const** or **volatile**. As a result, the **const** or **volatile** item could be modified without being detected by the compiler.

This warning often occurs when a pointer to an item declared as **const** or **volatile** is assigned to a pointer that was not declared as pointing to either of these type modifiers.

The expression was compiled without modification.

The following example generates this warning:

```
const  char *p = "abcde";
int    str( char *s );

str( p );
```

C4091 **no symbols were declared**
Level 2

The compiler detected an empty declaration, as in the following example:

```
int ;
```

The declaration was ignored.

C4092 **sizeof returns unsigned long**
Level 4

The operand of the **sizeof** operator was very large, so the **sizeof** operator returned an **unsigned long**.

This warning occurs only under the default /Ze Microsoft extensions. Under the /Za ANSI-compatibility option, this condition truncates the result of the **sizeof** operator.

C4093 **unescaped newline in character constant in inactive code**
Level 3

The constant expression of an **#if**, **#elif**, **#ifdef**, or **#ifndef** preprocessor directive evaluated to zero, making the code that follows inactive. Within that inactive code, a newline character appeared within a set of single or double quotation marks.

All text until the next double quotation mark was considered to be within a character constant.

C4094 **untagged** ***token*** **declared no symbols**
Level 2

The compiler detected an empty declaration using an untagged structure, union, or class.

The declaration was ignored.

The following example generates this warning:

```
struct  { };
```

This condition generates an error with the /Za ANSI-compatibility command-line option.

C4095 **expected ')'; found** *token*
Level 1

More than one argument was given for a pragma that can take only one argument.

The compiler assumed the expected right parenthesis and ignored the remainder of the line.

The following example generates this warning:

```
#pragma skip( 10, 18, 67 )
```

C4096 *attribute1* **must be used with** *attribute2*
Level 2

Attribute2 requires the use of *attribute1*.

For example, using a variable number of arguments (**...**) requires that **__cdecl** be used. Also, **__interrupt** functions must be **__far** and **__cdecl**.

The compiler assumed the *attribute1* attribute for the function.

C4098 *function* **: void function returning a value**
Level 1

A function that was declared with a return type of **void** contained a return statement that returned a value.

The compiler assumed the function returned a value of type **int**.

C4099 **type declared with** *objecttype1* **is defined with** *objecttype2*
Level 2

An object was declared as either a structure or a class but was defined as a class or a structure, respectively.

The object type given in the definition was used.

C4100 *identifier* **: unreferenced formal parameter**
Level 3 and 4

The given formal parameter was never referenced in the body of the function for which it was declared.

With a C++ program, this warning is level 3. Under the /Za ANSI-compatibility command-line option, this is a level 4 warning.

This warning is informational.

C4101 *identifier* **: unreferenced local variable**
Level 3

The given local variable was never used.

This warning is informational.

C4102 *label* **: unreferenced label**
Level 3

The given label was defined but never referenced.

This warning is informational.

C4103 *filename* **: used #pragma pack to change alignment**
Level 1

The given include file used the **pack** pragma to change the default structure alignment.

This warning is informational.

C4104 *identifier* **: near data in same_seg pragma; ignored**
Level 1

The given near variable was specified in a **same_seg** pragma.

The **same_seg** pragma applies only to external far variables.

The identifier was ignored.

C4105 *identifier* **: code modifiers only on function or pointer to function**
Level 1

The given identifier was declared with a code modifier that can be used only with a function or function pointer.

The code modifier was ignored.

C4106 **pragma requires an integer between 1 and 127**
Level 1

An integer constant between 1 and 127 must be specified with the **page** and **skip** pragmas.

The compiler assumed a value of 1.

The following example generates this warning:

```
#pragma skip( 0 )
```

C4107 **pragma requires an integer between 15 and 255**
Level 1

An integer constant between 15 and 255 must be specified with the **pagesize** pragma.

The compiler assumed a value of 63.

The following example generates this warning:

```
#pragma pagesize( 10 )
```

C4108 **pragma requires an integer between 79 and 132**
Level 1

An integer constant between 79 and 132 must be specified with the **linesize** pragma.

The compiler assumed a value of 79.

The following example generates this warning:

```
#pragma linesize( 42 )
```

C4109 **unexpected identifier** *identifier*
Level 1

The pragma contained an unexpected token.

The pragma was ignored.

The following example generates this warning:

```
#pragma optimize( on, off )
```

C4110 **unexpected token** *number*
Level 1

The pragma contained an unexpected integer constant.

The pragma was ignored.

The following example generates this warning:

```
#pragma title( 1984 )
```

C4111 **unexpected token** *string*
Level 1

The pragma contained an unexpected string.

The pragma was ignored.

The following example generates this warning:

```
#pragma linesize( "92" )
```

C4112 **#line requires an integer between 1 and 32,767**
Levels 1 and 4

The **#line** directive specified an integer outside the permitted range.

If the given line number is less than 1, the line counter is reset to 1. If the given line number is greater than 32,767, it is used as is.

This is a level 1 warning under the /Za ANSI-compatibility command-line option and a level 4 warning with the default /Ze Microsoft extensions.

C4113 **function formal parameter lists differed**
Level 1

A function pointer was assigned to another function pointer, but the formal parameter lists of the functions do not agree.

The assignment was compiled without modification.

C4114 **same type qualifier used more than once**
Level 1

A type qualifier (**const**, **volatile**, **signed**, or **unsigned**) was used more than once in the same type declaration or definition.

The second occurrence of the qualifier was ignored.

C4115 *symbol* **: named type definition in parentheses**
Levels 1 and 3

The given symbol was used to define a structure, union, or enumerated type inside a parenthetical expression. The scope of this definition may be unexpected.

In a C function call, the definition has global scope. In a C++ function call, the definition has the same scope as the function being called.

This is a level 1 warning with C++ programs or under the /Za ANSI-compatibility command-line option. It is level 3 otherwise.

C4116 **unnamed type definition in parentheses**
Level 1

A structure, union, or enumerated type with no name was defined in a parenthetical expression. The type definition is meaningless.

In a C function call, the definition has global scope. In a C++ function call, the definition has the same scope as the function being called.

C4117 **macro name** *name* **is reserved;** *command* **ignored**
Level 1

The given command attempted to define or undefine the predefined macro *name* or the preprocessor operator **defined**.

The given *command* is displayed in the warning message as either **#define** or **#undef**, even if the attempt was made using command-line options.

The command was ignored.

C4118 **pragma not supported**
Levels 1 and 3

The pragma was not supported by the compiler. The pragma was ignored.

This is a level 1 warning, except with **#pragma comment( )**, which causes a level 3 warning.

C4119 **different bases** *name1* **and** *name2* **specified**
Level 1

The based pointers in the expression have different symbolic bases. This may cause truncation or loss in generated code.

To avoid this warning, cast both based pointers to far pointers.

C4120 **based/unbased mismatch**
Level 1

The expression contains a conversion between a based pointer and another pointer that is not based. Some information may have been truncated.

This warning commonly occurs when a based pointer is passed to a function that expects a near or far pointer.

C4121 *symbol* **: alignment of a member was sensitive to packing**
Level 4

The given symbol was aligned to the current packing alignment.

The default alignment of structures can be specified with the **pack** pragma or the /Zp command-line option.

C4122 *function* **: alloc_text applicable only to functions with C linkage**
Level 1

The **alloc_text** pragma can be applied only to functions declared with **extern c**. This pragma cannot be used with external C++ functions.

The **alloc_text** pragma was ignored.

C4123 **different base expressions specified**
Level 1

The expression contained a conversion between based pointers, but the base expressions of the based pointers were different. Some of the based pointer conversions may be unexpected.

C4124 **__fastcall with stack checking is inefficient**
Level 1

The **__fastcall** keyword was used when stack checking was enabled.

The **__fastcall** calling convention is used for generating faster code, but stack checking causes slower code to be generated. Use the /Gs option or the **check_stack** pragma to turn off stack checking when using **__fastcall**.

This warning is informational and is issued only for the first function declared under these conditions.

C4125 **decimal digit terminates octal escape sequence**
Level 4

An octal escape sequence in a character or string constant was terminated by a decimal digit.

The compiler evaluated the octal number without the decimal digit and assumed the decimal digit was a character.

The following example generates this warning:

```
char array1[] = "\709";
```

If the digit 9 was intended as a character and was not a typing error, correct the example as follows:

```
char array[] = "\0709";  /* String containing "89" */
```

C4128 **storage-class specifier after type**
Level 4

A storage-class specifier (**auto**, **extern**, **register**, or **static**) appears after a type specifier in a declaration. The compiler assumed the storage-class specifier occurred before the type specifier.

New-style code places the storage-class specifier first.

C4129 *character* **: unrecognized character escape sequence**
Level 1

The *character* following a backslash in a character or string constant was not recognized as a valid escape sequence.

As a result, the backslash (\) is ignored and not printed, and the character following the backslash is printed.

To print a single backslash, specify a double backslash (\\).

C4130 *operator* **: logical operation on address of string constant**
Level 4

The operator was used with the address of a string literal. Unexpected code was generated.

For example, the following code generates this warning:

```
char *pc;
pc = "Hello";
if (pc == "Hello")
{ }
```

The **if** statement compares the value stored in the pointer `pc` to the address of the string `Hello`, which is allocated separately each time the string occurs in the code.

The **if** statement does not compare the string pointed to by `pc` with the string `Hello`.

To compare strings, use the **strcmp** function.

C4131 *function* **: uses old-style declarator**
Level 4

The specified function declaration is not in prototype form.

Old-style function declarations should be converted to prototype form.

An example of an old-style declaration is:

```
int addrec( name, id )
char *name;
int id;
{ }
```

The new-style prototype form is:

```
int addrec( char *name, int id )
{ }
```

C4132 *object* **: const object should be initialized**
Level 4

The constant was not initialized.

Since the value of a symbol with the **const** attribute cannot be changed after its definition, it should be given an initialization value.

It will not be possible to assign a value to *object.*

C4133 *type* **: incompatible types—void pointer combined with nonvoid pointer**
Level 3

An attempt was made to subtract two pointers, one of which is a pointer to a **void** type.

This warning can be avoided by providing the appropriate type cast.

C4134 **conversion between pointers to members of same class**
Level 4

A pointer to one member of a class was converted to a pointer to another member of the class. This is necessary because void pointers to members are not allowed.

This warning is informational.

C4135 **conversion between different integral types**
Level 3

Information was lost between two integral types.

This warning can be avoided by providing the appropriate type cast.

For example, the following code generates this warning:

```
int intvar;
long longvar;
intvar = longvar;
```

C4136 conversion between different floating-point types

Level 3

Information was lost or truncated in the conversion between two floating-point types.

For example, the following code generates this warning:

```
double doublevar;
float floatvar;
floatvar = doublevar;
```

Note that unsuffixed floating-point constants have type **double**, so the following code generates this warning:

```
floatvar = 1.0;
```

If the floating-point constant should be treated as type **float**, use the `F` (or `f`) suffix on the constant to prevent the warning:

```
floatvar = 1.0F;
```

C4137 *function* : no return value from floating-point function

Level 1

The given function had no return statement.

A **long double** function returns its value on the floating-point stack or the emulation stack. If the function does not return a value, a run-time floating-point stack underflow error may occur.

Make sure that all floating-point functions return a floating-point value.

C4138 */ found outside of comment

Level 1

The compiler found a closing comment delimiter (*/) without a preceding opening delimiter. The compiler assumed a space between the asterisk (*) and the forward slash (/).

The following example generates this warning:

```
int */*comment*/ptr;
```

In this example, the first comment delimiter is ambiguous.

Usually, the cause of this warning is an attempt to nest comments.

To comment out sections of code that may contain comments, enclose the code in an **#if/#endif** block and set the controlling expression to zero, as in:

```
#if 0
int my_variable;   /* Declaration currently not needed */
#endif
```

C4139 *hexnumber* **: hex escape sequence is out of range**
Level 1

A hexadecimal escape sequence appearing in a character or string constant was too large to be converted to a character.

If the hexadecimal escape sequence is in a string constant, the compiler casts the low byte of the hexadecimal number to a **char**. If in a **char** constant, the compiler made the cast and then sign extended the result. If in a **char** constant and the program was compiled with the /J command-line option, the compiler cast the value to an **unsigned char**.

Note that the following code generates this warning:

```
printf("\x7Bell\n");
```

The compiler reads the characters `7Be` as a legal hexadecimal number, but it is too large to fit in a character. To correct this example, use three hexadecimal digits:

```
printf("\007Bell\n");
```

C4140 *function* **redefined : preceding references may be invalid**
Level 1

The compiler issues this warning when a function definition changes between incremental compilations while compiling with the /f, /Gi, or /qc command-line options.

References previous to the redefinition use the previous definition. Subsequent references use the new definition.

For example:

```
main()
{
func1 ();
}
int func1 ()
{ }
```

If this program is compiled with the /f, /Gi, or /qc option and later the `func1` definition is changed to `long func1`, the compiler will issue this message to warn that calls to `func1` may be of the wrong type.

To avoid the problem, use function prototypes.

C4142 **benign redefinition of type**
Level 1

A type was redefined, but the redefinition had no effect on the code generated.

This warning commonly occurs in these cases:

- The return type of a member function of a derived class differed from the return type of the corresponding overridden member function.
- A type defined with the **typedef** command is redefined using different syntax. For example, the following code causes this warning when generating 16-bit code:

```
typedef   unsigned int WORD;
typedef   unsigned short WORD;
```

This warning is informational.

C4143 **pragma same_seg not supported; use __based allocation**
Level 1

The **same_seg** pragma is not supported with the /f or /qc fast-compilation command-line options.

To avoid this warning, use based pointers to specify the segment for external far variables. This warning can also be avoided by removing the /f or /qc command-line option.

C4144 *expression* **: relational expression as switch expression**
Level 1

The given relational expression was used as the control expression of a **switch** statement. The associated **case** statements will be offered boolean values.

This warning is informational.

C4145 *expression1* **: relational expression as switch expression; possible confusion with** *expression2*
Level 1

The given relational expression *expression1* was used as the control expression of a **switch** statement. The associated **case** statements will be offered boolean values.

The given expression *expression2* is a suggested **case** expression.

This warning is informational.

C4149 *class1* **: different ambient model than base class** *class2*
Level 3

The given derived class has a different ambient memory model than its base class.

This warning is informational.

C4150 **deletion of pointer to incomplete type** *type***; no destructor called**
Level 2

The **delete** operator was called to delete the given type, which was declared but not defined. The compiler was unable to find any destructors for the given type.

Make sure that the class, structure, or union is defined before using the **delete** operator.

The following example generates this warning:

```
class  IncClass;
void   NoDestruct( IncClass* pIncClass )
{
delete pIncClass;
}
```

C4151 *operator* **: operator should be explicitly** *model*
Level 1

The **new** operator was overloaded, but there was not an appropriate function for either **__based** or **__huge**.

To avoid this warning, write versions of the **new** operator to cover this case.

C4184 **near call to thunk for** *function* **in a different segment**
Level 1

A near member function was overridden by a near member function in another segment.

The compiler extended the near call to a far call.

This warning can be avoided by making both functions far or by making sure that the overriding member function is in the same segment as the original member function.

C4241 *member* **: member access is restricted**
Level 3

There was an attempt to access a private or protected member function or data.

If the program needs to access this member, change the object access level or make the member a friend of the function that needs to be accessed.

C4245 **friend specified for nonexistent function** *function*
Level 1

A nonfunction was declared to be a **friend**. The compiler ignored the statement.

Make sure that the friend function is accessible within the current scope.

C4247 *member* **not accessible because** *class1* **uses** *access* **to inherit from** *class2*
Level 3

There was an attempt to access a member of a class that was inherited from another class with the private or protected attribute.

This is a warning instead of an error, because the access violation occurred due to a type cast.

C4248 *class1* **: cannot access** *member* **member declared in class** *class2*
Level 3

The given member cannot be accessed from this class because it did not have access rights. The member was not accessed.

This warning can be avoided by giving *class1* access to members of *class2* by changing the class access specifier or using the **friend** specifier.

C4249 *class1* **: no path to** *member* **member declared in virtual base** *class2*
Level 3

The given member of the virtual base class cannot be accessed from the derived class because it did not have access rights. The member was not accessed.

This warning can be avoided by giving *class1* access to members of *class2* by changing the class access specifier or using the **friend** specifier.

C4250 *class1* **: inherits** *class2::member* **via dominance**
Level 2

There were two or more members with the same name. The one in *class2* was inherited since it was a base class for the other classes that contained this member.

This warning is informational.

C4300 **conversion of pointer to nonintegral type**
Level 1

A pointer was cast to a nonintegral type (such as **float**).

The cast pointer address may no longer be valid.

This warning is generated by the following example:

```
int     i = 24;
printf( "Address: %f\n", ( float ) &i );
```

C4301 truncation during conversion of pointer to integral type
Level 4

A pointer was converted to a smaller integral type. Information was lost in the casting.

The cast pointer address may no longer be valid.

This warning is generated by the following example:

```
int     i = 24;
printf( "Address: %hi\n", ( short ) &i );
```

C4302 truncation during conversion of pointer to function
Level 4

A function pointer was converted to a smaller function pointer type. Information was lost in the casting.

This warning is generated by the following example:

```
typedef void * ( *TYPE1 )( void );
typedef void * ( __far *TYPE2 )( void );

TYPE1   ptr1;
TYPE2   ptr2;

ptr1 = ( TYPE1 ) ptr2;
```

C4303 truncation during conversion of pointer to function to pointer to object
Level 4

A function pointer was converted to a smaller object or data pointer type. Information was lost in the casting.

This condition is an error with the C compiler.

This warning is generated by the following C++ example:

```
void __far myfunc( void ) {};
void * ptr;
 ptr = ( void * ) myfunc;
```

C4304 truncation during pointer conversion
Level 4

A function pointer was converted to a smaller object or data pointer type. Information was lost in the casting.

This warning is generated by the following example:

```
int __far * j;
int * k;

k = ( int * ) j;
```

C4305 truncation during conversion of integral type to pointer

Level 1

An integral type was converted to a smaller pointer type. Information was lost in the casting.

This warning is generated by the following example:

```
long   j;
long * k;

k = ( long * ) j;
```

C4306 conversion of integral type to pointer of greater size

Level 3

An integral type was cast to a larger pointer. The unfilled high bits of the new type will be zero-filled.

This warning may indicate an unwanted conversion. The resulting pointer may not be valid.

C4341 signed value out of range for enum constant

Level 3

A given enumerated constant exceeded the limit for an **int**. The value of the illegal constant is undefined.

Make sure that the given constants resolve to integers between –32,768 and +32,767 (**signed**) or 0 and +65,535 (**unsigned**).

The following example generates this warning:

```
enum ELEMENTS
{
Hydrogen = 1,
Helium,
Lithium,
Beryllium,
Impossibillium = 500000
} ;
```

C4342 precision lost in initialization of enum constant

Level 1

An enumerated constant was given in an expression too complex for the compiler to resolve accurately to an integer.

Make sure that the compiler can resolve the constant by simplifying the expression.

C4354 *reference* : **initialization of reference member requires a temporary variable**
Level 1

There was an attempt to initialize a member that was a reference. This condition causes an error under the default /Ze command-line option.

The compiler created a temporary stack variable to perform the initialization. Since the stack variable will be eliminated after the termination of the constructor, the pointer will be invalid.

This warning may be avoided by initializing the member instead of its reference.

C4401 *identifier* : **member is bit field**
Level 1

An attempt was made to access the given bit-field member within inline assembler code.

The identifier is a bit field. A bit-field member cannot be accessed within inline assembler code. The last packing boundary before the bit-field member was used.

To avoid this warning, cast the bit field to an appropriate type before making the reference in inline assembler code.

C4402 **must use PTR operator**
Level 1

A type was used on an operand without a **PTR** operator.

The compiler assumed the **PTR** operator.

The PTR operator is required when referring to or casting to a type in inline assembler code.

C4403 **illegal PTR operator**
Level 1

A **PTR** operator was used inappropriately in inline assembler code.

The compiler ignored the **PTR** operator.

C4404 **period on directive ignored**
Level 3

The optional period preceding the directive was ignored.

This warning is informational.

C4405 *identifier* : **identifier is reserved word**
Level 1

A reserved word was used as the name of the identifier.

Use of the identifier in this way may cause unpredictable results.

This warning can be avoided by not using reserved words as idenitifiers.

C4406 **operand on directive ignored**
Level 1

The directive does not take any operands, but an operand was specified.

The compiler ignored the given operand or operands.

C4409 **illegal instruction size**
Level 1

The instruction did not have a form with the specified size. The smallest legal size was used.

Make sure that the proper form of the instruction is being used.

C4410 **illegal size for operand**
Level 1

One of the operands on the instruction had an incorrect size. The smallest legal size for the operand was used.

Make sure that an operand of the proper size is being used.

C4411 *identifier* **: symbol resolves to displacement register**
Level 1

The identifier is a local symbol that resolves to a displacement register and therefore may be used on an operand with another symbol.

This warning is informational.

C4414 *function* **: short jump to function converted to near**
Level 3

Short jumps generate a 1-byte instruction. The instruction includes a short offset that represents the distance between the jump and the function definition. The compiler must generate a special record for the jump, which requires the **jmp** instruction to be either **NEAR** or **FAR**, but not **SHORT**. The compiler made the conversion.

For example, the following code generates this warning:

```
main()
{
_asm jmp SHORT main
}
```

C4500 *class* **: class has private/protected data members; user-defined constructor advised**
Level 2

The given class, structure, or union contains private or protected data. As this data is unavailable to many parts of the program, a user-defined constructor should be used.

C4501 *identifier* **: use of '::' unnecessary here**
Level 1

The given identifier was local in scope, so the scope resolution operator (**::**) was not required.

The compiler ignored the optional scope resolution operator.

C4502 *linkage* **requires use of keyword extern**
Level 1

A linkage was specified without the **extern** keyword. Linkage is not relevant to non-extern types.

The compiler assumed the **extern** keyword.

C4505 *function* **: unreferenced local function has been removed**
Level 4

The given function is local and not referenced in the body of the module; therefore, the function is dead code.

The compiler did not generate code for this dead function.

C4506 **no definition for inline function** *function*
Level 1

The given function was declared and marked for inlining but was not defined.

The compiler did not inline the function.

Make sure that external functions to be inlined are declared with the **extern** keyword.

C4507 **explicit linkage specified after default linkage was used**
Level 4

After an identifier was declared with the default C++ linkage, it was explicitly declared as having C++ linkage.

The following code generates this warning:

```
void   WhereProhibited( void );
extern "C++" { void   WhereProhibited( void ); };  //Warning
```

C4508 *function* **: function should return a value; void return type assumed**
Level 1

The given function did not specify a return type; the compiler assumed that the return statement returned type **void**.

This warning can be avoided by declaring functions **void** that do not return a value.

C4510 *class* **: default constructor could not be generated**
Level 3

The compiler was unable to generate a default constructor for the given class. No constructor was created.

This warning can be caused by having a default constructor for the base class that is not accessible by the derived class. An inherited constructor from a base class with a different ambient memory model can also cause this warning.

This warning can be avoided by specifying a user-defined default constructor for the class.

C4511 *class* **: copy constructor could not be generated**
Level 3

The compiler was unable to generate a copy constructor for the given class. No constructor was created.

This warning can be caused by having a copy constructor for the base class that is not accessible by the derived class. An inherited copy constructor from a base class with a different ambient memory model can also cause this warning.

This warning can be avoided by specifying a user-defined copy constructor for the class.

C4512 *class* **: assignment operator could not be generated**
Level 3

The compiler was unable to generate a default constructor for the given class. No constructor was created.

This warning can be caused by having an assignment operator for the base class that is not accessible by the derived class. An inherited assignment operator from a base class with a different ambient memory model can also cause this warning.

This warning can be avoided by specifying a user-defined assignment operator for the class.

C4513 *class* : **destructor could not be generated**
Level 3

The compiler was unable to generate a default constructor for the given class. No constructor was created.

This warning can be caused by having an assignment operator for the base class that is not accessible by the derived class. An inherited assignment operator from a base class with a different ambient memory model can also cause this warning.

This warning can be avoided by specifying a user-defined destructor for the class.

C4520 *class* : **multiple default constructors specified**
Level 3

There were multiple default constructors specified for the given class. The first constructor was used.

Make sure that there is only one default constructor defined for a distance.

C4521 *class* : **multiple copy constructors specified**
Level 3

There were multiple copy constructors of a single type specified for the given class. The first constructor was used.

Make sure that there is only one copy constructor defined for a type and distance.

C4522 *class* : **multiple assignment operators specified**
Level 3

There were multiple assignment operators of a single type specified for the given class. The first constructor was used.

Make sure that there is only one assignment operator defined for a type and distance.

C4523 *class* : **multiple destructors specified**
Level 3

There were multiple destructors specified for the given class. The first destructor was used.

Make sure that there is only one destructor defined for a distance.

C4524 *class* : **redundant use of friend on destructor**
Level 3

The **friend** access specifier was given for a destructor. Since the destructor is automatically a member function of the given class, this specification is irrelevant.

The extraneous **friend** specifier was ignored.

C4525 *class* **: redundant use of friend on constructor**
Level 3

The **friend** access specifier was given for a constructor. Since the constructor is automatically a member function of the given class, this specification is irrelevant.

The extraneous **friend** specifier was ignored.

C4527 **instances of type** *class* **can never be destroyed—user-defined destructor required**
Level 3

The given derived class did not include a destructor, and the compiler could not generate one. The class cannot be destroyed.

This warning can be avoided by including a user-defined destructor for the given class.

C4607 *const* **: must be initialized in constructor base/member initializer list**
Level 1

The given constant was not initialized with an initializer list in the object constructor. The compiler left the constant undefined.

If a **const** member variable is not given a value when initialized, it must be given a value in the object constructor.

C4610 **object** *class* **can never be instantiated—user-defined constructor required**
Level 1

The given class had neither user-defined nor default constructors; it cannot be instantiated. No instantiation was performed.

Make sure that all classes have valid user-defined or default constructors.

C4612 **bad #pragma syntax, pragma ignored**
Level 1

There was a syntax error in the **warning** pragma.

Make sure that all values and parentheses for this pragma are correct.

C4613 *segment* **: class of segment cannot be changed**
Level 1

There was an attempt to create a segment with the same class name as a segment used by the compiler.

No new segment class was created.

This warning is generated only when compiling 32-bit programs.

C4615 **#pragma warning : unknown user warning type**
Level 1

An illegal warning type was used with the **warning** pragma.

Valid warning types are “1”, “2”, “3”, “4”, “default”, “off”, and “error”.

C4616 **#pragma warning : warning number** ***number1*** **out of range, must be between** ***number2*** **and** ***number3***
Level 1

The warning number that was specified in the **warning** pragma was not in the range of assignable warnings. Only warnings between *number2* and *number3* can be reassigned.

The pragma was ignored.

C4617 **#pragma warning : invalid warning number**
Level 1

An illegal warning number was used with the **warning** pragma.

Make sure that the warning number that is supplied to the **warning** pragma corresponds to a valid warning.

C4620 **no postfix form of operator ++ found for type** ***type*****, using prefix form**
Level 1

There was no postfix incrememnt operator defined for the given type. The compiler used the overloaded prefix operator.

This warning can be avoided by defining a postfix **++** operator. This is done by creating a two-argument version of the **++** operator as shown below:

```
class prepost
{
public:
    prepost operator++ ()
    {
       /* inline prefix routine */
    };

    prepost operator++ ( int )
    {
       /* inline postfix routine */
    };
}
```

C4621 **no postfix form of operator – – found for type *type*, using prefix form**
Level 1

There was no postfix decrement operator defined for the given type. The compiler used the overloaded prefix operator.

This warning can be avoided by defining a postfix – – operator. This is done by creating a two-argument version of the – – operator as shown below:

```
class prepost
{
public:
    prepost operator-- ()
    {
       /* inline prefix routine */
    };

    prepost operator-- ( int )
    {
       /* inline postfix routine */
    };
}
```

C4630 ***symbol* : *extern* storage class specifier illegal on member definition**
Level 1

The given data member or member function was defined as external. Members cannot be declared as **extern**.

The compiler ignored the **extern** keyword.

Only entire objects can be made external.

C4650 **debugging information not in precompiled header; only global symbols from the header will be available**
Level 1

The precompiled header file was not compiled with Microsoft symbolic debugging information.

When linked, the resulting executable or dynamic-link library file will not include debugging information for local symbols contained in the precompiled header.

This warning can be avoided by recompiling the precompiled header file with the /Zi command-line option.

C4651 *definition* **specified for precompiled header but not for current compile**
Level 1

The given definition was specified when the precompiled header was generated, but not in this compilation.

The definition will be in effect inside the precompiled header, but not in the rest of the code.

C4652 **command-line option** *option* **inconsistent with precompiled header; PCH option ignored**
Level 1

The given command-line option differed from that given when the precompiled header (.PCH) was created. The option specified in the current command line was used.

This warning may be avoided by regenerating the precompiled header with the given command-line option.

C4700 **local variable** *identifier* **used without having been initialized**
Level 1

A reference was made to a local variable that had not been assigned a value. As a result, the value of the variable is undefined.

This warning is given only when compiling with the /Oe global register allocation command-line option.

C4701 **local variable** *identifier* **may be used without having been initialized**
Level 3

A reference was made to a local variable that may not have been assigned a value. As a result, the value of the variable may be unpredictable.

This warning is given only when compiling with the /Oe global register allocation command-line option.

C4702 **unreachable code**
Level 4

The flow of control can never reach the indicated line.

This warning is given only when compiling with one of the global optimization options (/Oe, /Og, or /Ol).

C4703 *function* **: function too large for global optimizations**
Level 1

The named function was too large to fit in memory and be compiled with the selected optimization option. The compiler did not perform any global optimizations (/Oe, /Og, or /Ol). Other /O*x* optimization options, such as /Oa and /Oi, are still performed.

One of the following may remove this warning:

- Divide the function into two or more smaller functions.
- Recompile with fewer optimizations.

C4704 *function* **: inline assembler precludes global optimizations**
Level 3

The use of inline assembler in the named function prevented the specified global optimizations (/Oe, /Og, or /Ol) from being performed.

C4705 **statement has no effect**
Level 4

The indicated statement will have no effect on the program execution.

Each of these statements will generate this warning:

```
1;
a + 1;
b == c;
```

C4706 **assignment within conditional expression**
Level 4

The test value in a conditional expression was the result of an assignment.

This warning is informational.

An assignment has a value (the value on the left side of the assignment) that can be used legally in another expression, including a test expression. However, the intention may have been to test a relation, not to make an assignment.

For example, the following line, which generates this warning, assigns `b` to `a` and compares the value of `a` with `0`:

```
if ( a = b ) ...
```

However, the following line tests whether `a` and `b` are equal:

```
if ( a == b ) ...
```

C4709 **comma operator within array index expression**
Level 4

The value used as an index into an array was the last one of multiple expressions separated by the comma operator.

An array index legally may be the value of the last expression in a series of expressions separated by the comma operator. However, the intent may have been to use the expressions to specify multiple indexes into a multidimensional array.

For example, the following line, which generates this warning, is legal in C and specifies the index c into array a:

```
a[b,c]
```

However, the following line uses both b and c as indexes into a two-dimensional array:

```
a[b][c]
```

C4710 **function *function* not expanded**
Level 4

The given function was selected for inline expansion, but the compiler did not perform the inlining.

Inlining is performed at the compiler's discretion. The **inline** keyword, like the **register** keyword, is used as a hint for the compiler.

C4711 **function *function* selected for inline expansion**
Level 1

The compiler performed inlining on the given function, although it was not marked for inlining by the user.

Inlining is performed at the compiler's discretion. This warning is informational.

C4712 ***symbol* : used as register—loss of debugging information**
Level 2

The given symbol was based on another symbol that was converted to a register, thereby losing debugging information.

This warning can be avoided by not using the /Oe register-allocation command-line option.

C4723 **potential divide by 0**
Level 3

The second operand in a divide operation evaluated to zero at compile time, giving undefined results.

The 0 operand may have been generated by the compiler, as in the following example:

```
int    i, j, k = 42;
i /= ( j && k );
```

C4724 **potential mod by 0**
Level 3

The second operand in a remainder operation evaluated to zero at compile time, giving undefined results.

C4726 *option* **: unknown memory-model command-line option**
Level 1

The character (or characters) used with the /A option was not recognized as a valid memory-model specifier.

The option was ignored.

C4727 **conditional expression is constant**
Level 4

The controlling expression of an **if** statement or **while** loop evaluated to a constant.

As a result, the code in the body of the **if** statement or **while** loop either always executes or never executes.

This warning is informational.

C4741 **/Oq option ignored for __fastcall or __saveregs function***function*
Level 1

The fastcall and saveregs calling convention is not allowed for p-code functions. The function was compiled into native code.

This warning can be avoided by using a different calling convention or by turning p-code generation off during the definition of fastcall and saveregs functions, as in:

```
#pragma optimize( "q", off )    //Turn p-code on
int __fastcall FastFunc( void )
{...};
#pragma optimize( "q" )         //Return to default p-code mode
```

C4746 *identifier* **: unsized array treated as __far**
Level 1

An unsized array was allocated in the far data segment.

This warning occurs under the /Gx command-line option when initializing an unsized array of characters with multiple elements, as in:

```
char   StringArray[] = {"Windows", "Windows", "Windows"};
```

Since no explicit size was given, the data size threshold cannot be used.

C4756 **overflow in constant arithmetic**
Level 2

The compiler generated a floating-point exception while doing constant arithmetic on floating-point items during compilation.

For example:

```
float fp_val = 1.0e100;
```

In this example, the floating-point constant `1.0e100` exceeds the maximum allowable value for a double-precision data item.

C4758 **address of automatic (local) variable taken; DS != SS**
Level 1

The program was compiled with the default data segment (DS) not equal to the stack segment (SS), and the program tried to point to an automatic (local) variable with a near pointer.

Dereferencing a pointer to that address is illegal when DS != SS and will give an unpredictable result.

C4759 **segment lost in conversion**
Level 2

The conversion of a far pointer (a full segmented address) or based pointer to a near pointer (a segment offset) or a different based pointer resulted in the loss of the segment address.

This warning is informational.

C4761 **integral size mismatch in argument; conversion supplied**
Level 1

The base types of the actual and formal parameters of a function were different.

The compiler converted the actual parameter to the type of the formal parameter.

C4762 **near/far mismatch in argument; conversion supplied**
Level 1

The pointer sizes of the actual and formal parameters of a function were different.

The compiler converted the actual parameter to the type of the formal parameter.

C4763 *function* **: function too large for postoptimizer**
Level 2

Not enough space was available to optimize the given function.

One of the following may be a solution:

- Divide the function into two or more smaller functions.
- Recompile with fewer optimizations.

C4765 **recoverable heap overflow in postoptimizer**
Level 2

The postoptimizing pass ran out of memory but was able to recover. Some optimization options may have been ignored.

One of the following may be a solution:

- Divide the function into two or more smaller functions.
- Recompile with fewer optimizations.

C4766 **local symbol-table overflow**
Level 2

The listing generator ran out of heap space for local variables, so the source listing may not contain symbol-table information for all local variables.

One of the following may be a solution:

- Divide the function into two or more smaller functions.
- Recompile with fewer optimizations.

C4769 **conversion of near pointer to long integer**
Level 2

The compiler converted a 16-bit near pointer to a **long** integer by extending the high-order word with the current data segment value, not with zeros.

This warning is informational.

C4773 **scoping too deep; deepest scoping merged when debugging**
Level 1

Declarations appeared at a static nesting level greater than 16. As a result, all declarations beyond this level will seem to appear at the same level when being viewed by the debugger.

C4785 **near call to *function* in different segment**
Level 1

The given function was specified in an **alloc_text** pragma without being declared with **__far**, and then the function was called from the text segment.

The compiler generated a near call.

Although this is a warning message rather than an error message, the resulting code will not work correctly.

C4786 **string too long—truncated to 255 characters**
Level 1

The string for a **title** or **subtitle** pragma exceeded the maximum allowable length and was truncated.

C4788 *identifier* **: identifier was truncated to** *number* **characters**
Level 1

Only the first 31 characters of an identifier are significant in C. In C++, the first 247 characters (after name decoration) are significant. The characters after the limit were truncated.

This may mean that two identifiers that are different before truncation may have the same identifier name after truncation.

C4790 **insufficient memory to process debugging information**
Level 2

The program was compiled with the /Zi option, but not enough memory was available to create the required debugging information.

One of the following may be a solution:

- Split the current file into two or more files and compile them separately.
- Remove other programs or drivers running in the system that could be consuming significant amounts of memory.

C4791 **loss of debugging information caused by optimization**
Level 2

Some optimizations, such as code motion, cause references to nested variables to be moved. The information about the level at which the variables are declared may be lost. As a result, all declarations will seem to be at nesting level 1.

This warning can be eliminated by turning off the optimization options for debugging purposes.

C4792 **long double type not supported by alternate math library**
Level 1

The **long double** numeric type is not supported by the /FPa alternate math emulation library. This can cause unresolved external errors from the linker.

This warning can be avoided by using another emulation library (such as /FPi) or by avoiding the **long double** type.

Ignore this warning if a special handler for the **long double** type is included in the link process.

CVPACK Error Messages

Microsoft Debugging Information Compactor (CVPACK) generates the following error messages:

- Fatal errors (CK1*xxx*) cause CVPACK to stop execution.
- Warnings (CK4*xxx*) indicate possible problems in the packing process.

CVPACK Fatal Error Messages

Number **CVPACK Fatal Error Message**

CK1000 **unknown error; contact Microsoft Product Support Services**

CVPACK detected an unknown error condition.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

CK1001 **out of memory**

The executable file is too big for the available memory. This error can occur under DOS when there is little extra memory. Even though CVPACK uses virtual memory, which involves swapping to disk, some information can be stored only in real memory.

One of the following may be a solution:

- Recompile one or more of the object files without debugging information. If the file was compiled using the /Zi option, use either /Zd or no option.
- Add more memory to your computer.

CK1002 **out of virtual memory**

There was not enough virtual memory for CVPACK to pack the executable file. Virtual memory can be any of the following:

- Conventional memory. Remove TSR (terminate-and-stay-resident) programs or run CVPACK outside of a shell or a makefile.
- Extended or expanded memory. Run CVPACK under a DPMI server, such as MSDPMI.EXE or the server provided in a DOS session under Windows enhanced mode.
- Disk space. Free some disk storage.

CK1003 cannot open file

CVPACK could not open the specified executable file.

One of the following may be a cause:

- The specified file does not exist. Check the spelling of the filename and path.
- The executable file was opened or deleted by another process.

CK1004 file is read-only

CVPACK cannot pack a read-only file. Change the read attribute on the executable file and run CVPACK again.

CK1005 invalid executable file

CVPACK could not process the executable file. One of the following may be a cause:

- The debugging information in the executable file is corrupt.
- The executable file is a zero-length file.

CK1006 invalid module ***module***

The given object file did not have a valid format.

Check the linker version.

CK1007 invalid ***table*** **table in module** ***module***

The given table in the given object file was not valid.

Check the compiler and linker versions.

CK1008 cannot write packed information

There was not enough space on disk for CVPACK to write the packed executable file. This leaves a corrupt file on disk.

Make more space available on disk and relink the program.

CK1009 module ***module*** **unknown type index** ***number*****;**
contact Microsoft Product Support Services

The debugging information in the executable file is corrupt. This is due to an internal error in either the compiler or CVPACK. Recompile the program. If the problem persists, note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

CK1010 symbol error in module *module*; contact Microsoft Product Support Services

The debugging information in the executable file is corrupt. This is due to an internal error in either the compiler or CVPACK. Recompile the program. If the problem persists, note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

CK1011 error in type *number* for module *module*; contact Microsoft Product Support Services

The debugging information in the executable file is corrupt. This is due to an internal error in either the compiler or CVPACK. Recompile the program. If the problem persists, note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

CK1012 no Symbol and Type Information

The executable file does not contain debugging information.

Link the program using the /CO option to put at least minimal debugging information in the executable file. To include full debugging information in an object file, compile or assemble using the /Zi option. To include minimal information and line numbers, compile or assemble using the /Zd option.

CK1013 debugging information missing or unknown format

One of the following has occurred:

- The program did not contain debugging information. Recompile using /Zi or /Zd, then link using /CO.
- The executable file was linked using an obsolete or unsupported linker. Use Microsoft LINK version 5.3x or later.
- The executable file was already packed using a previous version of CVPACK.

CK1014 module *module* type *number* refers to skipped type index; contact Microsoft Product Support Services

The debugging information in the executable file is corrupt. This is due to an internal error in the compiler. Recompile the program. If the problem persists, note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

CK1015 too many segments in module *module*

The **alloc_text** pragma was used more than 20 times in an object file that was compiled with Microsoft C version 6.x or earlier.

One of the following may be a solution:

- Recompile using Microsoft C/C++ version 7.0 or later.
- Split the object file into multiple files.
- Group the pragma statements according to segment.

CK1016 unable to execute MPC for CVPACK /PCODE

CVPACK could not find MPC.EXE on the path.

CK1017 precompiled types file *filename* **not found**

The program used a precompiled header, but the program was linked without the object file that was created when the header was precompiled.

CK1018 precompiled types object file *filename* **inconsistent with precompiled header used to compile object file** *filename*

The program used a precompiled header, but the object file linked to the program was not the object file that was created when the header was precompiled. Either the user or the creator changed since the last compilation.

Recompile and relink. If a makefile is used, check the makefile dependencies.

CK1020 packed type index exceeds 65535 in module *module*

The debugging information exceeded a CVPACK limit.

This error may occur when precompiled headers are used.

One of the following may be a solution:

- Eliminate unused type strings.
- Compile some object files without debugging information.

CK1021 error in precompiled types signature in module *module*

The program was compiled with an out-of-date precompiled header.

Delete the object file and recompile.

CVPACK Warning Messages

Number CVPACK Warning Message

CK4000 unknown warning; contact Microsoft Product Support Services

CVPACK detected an unknown error condition.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

CK4001 file already packed

CVPACK took no action because the executable file has already been processed by CVPACK 4.00.

CK4002 duplicate public symbol *symbol* **in module** *module*

The given symbol was redefined in the given module. CVPACK deleted the second occurrence of the symbol.

Probably an earlier version of the linker was used. Use LINK 5.30 or later.

CK4003 error in lexical scopes for module *module***, symbols deleted**

The scoping of symbols in the given object module was corrupted. CVPACK deleted the symbols in the module.

This is probably a compiler error. Recompile and relink the object file.

CodeViewError Messages

CV Error Messages

CodeView displays an error message whenever it detects a command it cannot execute. Most errors terminate the CodeViewcommand that is in error but do not terminate the debugger. Startup errors terminate the debugger.

Note Depending on the context of the error, CodeView may display only the text of the message without the error number. CodeView error messages are organized in alphabetical order by message text.

In some cases, CodeView ay display the error number by itself. To obtain the error message text and an explanation of the error in those cases, use Help. Click the right mouse button on the error number or use the Help (**H**) Command-window command. For example, error number `CV1027` displays Help for the following error:

```
invalid radix: specify 8, 10, or 16
```

You can also get Help for any CodeView error message by using PWB and QuickHelp.

Expression Evaluator Error Messages

The C and C++ expression evaluators produce error message numbers prefixed by CAN and CXX, respectively. The C++ error messages are a superset of the C error messages. Therefore, only the CXX prefix is listed for the messages in this section. The expression evaluator error messages are listed in numerical order after the CodeView error messages.

CV Error Messages

Number **CV Error Message**

CV0013 **access denied**

A specified file's permission setting does not allow the required access.

One of the following may have occurred:

- An attempt was made to write to a read-only file.
- A locking or sharing violation occurred.
- An attempt was made to open a directory instead of a file.

CV1043 application output lost; screen exchange is off

The program being debugged wrote to the display when the Flip (/F) or Swap (/S) was turned off. The program output was lost.

Programs usually write to video page 0 by default. When flipping is on, video page 1 is usually reserved for CodeView. Programs that write to video page 1 must be debugged with swapping on.

Turn Flip or Swap on to be able to view program output.

CV4011 assembler: divide by zero

CodeView encountered a divide-by-zero error while assembling the current instruction.

CV4007 assembler: extra characters

The instruction contained extra characters that could not be recognized. The instruction may have been mistyped.

The line was ignored.

CV4008 assembler: illegal operand

The wrong type of operand was used for this context.

The instruction may have been mistyped.

CV4003 assembler: illegal range

The size of a specified value exceeds the size expected by the instruction.

CV4010 assembler: illegal register

An illegal or nonexistent register was accessed.

The register name may have been mistyped.

This error can be caused by trying to access 80386- or 80486-specific registers when CodeView is running on a 8088- or 80286-based machine.

CV4009 assembler: illegal segment

An invalid segment was used.

CV4002 assembler: incorrect operand size

An instruction required an operand of a different size.

CV4000 assembler: not enough operands

Additional operands are required for this instruction.

CV4004 assembler: overflow

Numeric overflow occurred while assembling the current instruction.

CV4005 assembler: syntax error

The syntax for the current instruction is incorrect.

CV4001 assembler: too many operands

Too many operands were specified for the most recently issued instruction.

CV4006 assembler: unknown opcode

An instruction was not recognized.

Check that the instruction was typed correctly.

CV3620 bad DLL format in ***filename***

CodeView did not recognize the format of the specified CodeView dynamic-link library (DLL) file.

The DLL may be damaged or may be the wrong version.

This error is caused if the specified file is not a DLL.

CV1006 breakpoint number or '*' expected

A breakpoint was specified without a number or asterisk.

A Breakpoint Clear (**BC**), Breakpoint Disable (**BD**), or Breakpoint Enable (**BE**) command requires one or more numbers to specify the breakpoints or an asterisk to specify all breakpoints.

For example, the following command causes this error:

```
bc rika
```

CV1068 breakpoint specifier is out of range

A breakpoint number was specified that is higher than the number of current breakpoints.

CV4012 cannot assemble code with current execution model

This error can be caused by trying to assemble p-code in CodeView.

CV1063 cannot create CURRENT.STS

CodeView could not find an existing state file (CURRENT.STS), and CodeView tried to create one but failed.

One of the following may have occurred:

- There was not enough space either on the disk containing the program to be debugged or on the disk pointed to by the INIT environment variable.
- There were not enough free file handles. Increase the number of file handles by changing the FILES setting in CONFIG.SYS to allow a larger number of open files.
- The environment variable INIT pointed to a directory that does not exist.

CV5014 **cannot execute function in watch expression**

A watch expression cannot specify a function to be executed.

CV3621 **cannot find DLL** *filename*

CodeView could not find the specified dynamic-link library (DLL). This may be caused by a mistyped filename in the TOOLS.INI file.

CV3622 **cannot load DLL** *filename*

CodeView was unable to load the specified dynamic-link library (DLL) file.

Reinstall the DLL from the distribution disks.

CV3624 **cannot load execution model** *filename*: **limit is 1**

Too many execution models are specified in the TOOLS.INI file.

Only one execution model can be used at a time.

Remove those execution models you are not using in your debugging session.

CV1065 **cannot load expression evaluator** *filename*

CodeView could not load the specified expression evaluator.

Make sure that *filename* is a valid expression evaluator DLL. If not, try reinstalling the CodeView DLLs from the distribution disks.

CV1066 **cannot load expression evaluator** *filename*; **limit is 10**

Up to 10 expression evaluators can be specified in the TOOLS.INI file.

Try removing expression evaluators you won't be using in your debugging session.

CV2404 **cannot open response file** *filename*

The specified response file could not be opened.

Check that the name of the file is spelled correctly and that the response file is correct.

CV5004 cannot read file

CodeView could not open a file.

Read the file again. If the second read fails, exit and restart CodeView. If the read process still fails, the file may be corrupt.

CV1054 cannot read this version of CURRENT.STS

The state file (CURRENT.STS) has a version number that is not recognized by this version of CodeView.

The old CURRENT.STS was ignored, and a new one will be created when CodeView exits.

CV2209 cannot restart; current process is not the process being debugged

The debugging session was halted, and a different process was started.

Return to the process being debugged by setting a breakpoint in it and issue a Go command.

CV5001 cannot select

The cursor was not on the same line as an automatically selectable symbol.

CV2211 cannot terminate; current process is not the process being debugged

The debugging session was halted, and a different process was current.

Return to the process being debugged by setting a breakpoint in it and issue a Go command.

CV1056 cannot understand entry in *filename*

At least one line in the given file (either the state file or the TOOLS.INI file) could not be interpreted.

On startup, CodeView reads the state file (CURRENT.STS) and the TOOLS.INI file (if the latter is available).

Examine the given file to find the problem.

CV0104 CodeView information for *filename* **is newer than this version of CodeView**

The executable file was compiled or linked with a version of a Microsoft compiler that is newer than the version of CodeView you are using.

You should try one of the following:

- Reinstall CodeView that came with the new compiler.
- Remove older versions of CodeView that may be present on your hard disk.
- Recompile the program with an older version of a Microsoft compiler.

CV2405 command-line option *option* invalid for target operating system

The specified command line option was illegal in this context.

CV2206 corrupt CodeView information in *filename*; discarding

This error can be caused by using mismatched versions of development tools. Verify that the versions of all tools are current and synchronized.

Try recompiling the file with the /Zd switch (Prepare for Debugging option).

This option produces an object file containing only public symbols (global or external) and line numbers.

CV1057 CURRENT.STS not found; creating

Since the state file (CURRENT.STS) could not be located at startup, CodeView created a state file.

CV0008 executable file format error

The system is not able to load the program to be debugged. The file is not an executable file, or it has an invalid format for this operating system.

Try to run the program outside of CodeView to see if it is a valid executable file.

This error can be caused if there is not enough memory available to run the program.

Try making more memory available to the program.

CV1050 expression is not a memory address

The expression does not evaluate to an address.

For example, `buffer[count]` is a valid address because it points to a specific memory location. The logical comparison `zed != 0` is not a valid address because it evaluates to TRUE or FALSE, not a memory address.

CV1067 extension missing for expression evaluator in *filename* in TOOLS.INI

The Eval entry in the TOOLS.INI file expected a list of filename extensions.

CV1003 extra input ignored

The first part of the command line was interpreted correctly.

The remainder of the line could not be interpreted or was unnecessary.

CV1041 file error

CodeView could not write to the disk.

One of the following may have occurred:

- There was not enough space on the disk.
- The file was locked by another process.

CV1048 floating-point support not loaded

An attempt was made to access the math processor registers in a program that does not use floating-point arithmetic.

One of the following can cause this error:

- If the program does not perform floating-point calculations, this error can occur because the floating-point library code will not be loaded and cannot be used to access math processor registers. Math processor registers can only be accessed through the floating-point library code.
- If the program does not use floating-point instructions, this error can occur when you attempt to access the math processor before any floating-point instructions have been performed. The run-time library includes a floating-point instruction near the beginning so that the math processor registers are always accessible.
- If a floating-point instruction occurs in an assembly-language routine before such an instruction occurs in the high-level language code that calls the routine, this error occurs.

CV1250 general expression-evaluator error

An error occurred in a CodeView expression evaluator.

This error is probably caused by a lack of memory available to the expression evaluator. You can free memory by doing one or more of the following:

- Closing windows that are not needed. The Memory window, in particular, should be closed if possible.
- Delete breakpoints that are not needed.
- Disabling options that are not needed.

As a last resort, exit CodeView and start the debugging session again.

This error can also be caused by an expression that cannot be evaluated by the expression evaluator.

CV0014 invalid address

The command expected an address but was given an argument that could not be interpreted as a valid address.

A name or constant may have been specified without the period (.) that indicates a filename or line number.

CV1254 invalid address expression

The expression entered does not evaluate to an address.

For example, `buffer[count]` is a valid address because it points to a specific memory location. The logical comparison `zed != 0` is not a valid address because it evaluates to TRUE or FALSE, not a memory address.

CV0022 invalid argument

An invalid value was given as an argument.

CV1001 invalid breakpoint command

CodeView could not interpret the breakpoint command.

The command probably used an invalid symbol or the incorrect command format.

CV1062 invalid code-segment context change

An attempt was made to set the IP register to a line or address in a different segment.

CV1046 invalid executable file: relink

The executable file did not have a valid format.

One of the following may have occurred:

- The executable file was not created with the linker released with this version of CodeView. Relink the object code using the current version of LINK.EXE.
- The .EXE file may have been corrupted. Recompile and relink the program.

CV1022 invalid flag

An attempt was made to examine or change a flag, but the flag name was not valid.

Any flags preceding the invalid name were changed to the values specified. Any flags after the invalid name were not changed.

CV4500 invalid format length; using variable length

An invalid length was specified for the Memory window. CodeView will set the length based on the current window width.

Try specifying a different length.

Use the flag mnemonics displayed after entering the R FL command.

CV1027 invalid radix: specify 8, 10, or 16

The Radix (**N**) command takes three radixes: 8 (octal), 10 (decimal), and 16 (hexadecimal). Other radixes are not permitted. The new radix is always entered as a decimal number, regardless of the current radix.

CV1004 invalid register

The Register (**R**) command named a register that does not exist or cannot be displayed. CodeView can access the following registers:

AX
BX
CX
DX
SP
BP
SI
DI
DS
ES
SS
CS
IP
FL

When running under DOS or Windows on a 80386 or a 80486 machine, the 80386 registers option can be selected to access the following registers:

EAX
EBX
ECX
EDX
ESP
EBP
ESI
EDI
DS
ES
FS
CS
GS
SS
EIP
EFL

When debugging p-code, CodeView can also access the following registers:

TL
TH
PQ

CV2210 invalid tab setting; using 8

The value for tabs cannot be less than 0 or greater than 19. If you supply a value that is not in this range, the default tab value is 8.

CV4501 invalid window id

The window ID was invalid. It must be either 0 or 1.

CV0005 I/O error

An attempt was made to access an address that is not accessible to the program being debugged.

Check the previous command for numeric constants used as addresses and for pointers used for indirection.

CV1042 **library module not loaded**

The program being debugged uses load-on-demand dynamic-link libraries (DLLs). At least one of these libraries is needed but could not be found.

CV2207 **loaded symbols for** *module*

CodeView automatically loaded the symbols for the given dynamic-link library (DLL). The DLL can now be debugged.

This message is informational only and does not indicate an error.

CV1016 **match not found**

A string could not be found that matched the search pattern.

CV1251 *message*

An error occurred within a CodeView expression evaluator.

No further explanation is available.

CV2401 **missing argument for** *option* **option**

This error can be caused by splitting a response file line specifying a program to be debugged and its command-line options. The program and its command-line options must be on one line.

CV1051 **missing or corrupt emulator information**

Status information about the floating-point emulator is missing or corrupt.

The program probably wrote to this area of memory. Check that each pointer points to its intended object.

CV1023 **no code at this line number**

A line number was specified but code was not generated for that line. This error can be caused by a blank line, comment line, line with program declarations, or line moved or removed by compiler optimization.

To set a breakpoint at a line deleted by the optimizer, recompile the program with the /Od option to turn off optimization.

Note that in a multiline statement the code is associated only with one line of the statement.

This error can be caused by debugging a program whose source has been modified after it was compiled. Recompile the file before running it through CodeView.

CV0101 no CodeView information for ***filename***

The executable file or dynamic-link library (DLL) did not contain the symbols needed by CodeView.

Be sure to compile the program or DLL using the /Zi option. If linking in a separate step, be sure to use the /CO option. Use the most current version of LINK.

CV1059 no CodeView source information

A CodeView symbol listing for the source file or module being debugged does not exist.

Be sure the file was compiled with the /Zi option or the /Zd option. If linking in a separate step, be sure to use the /CO option.

CV1255 no data members

The class, structure, or union that was expanded did not have data members. A class must contain at least one data member to be expanded.

CV0000 no error condition

You should not normally receive this error message since CV0000 indicates that no error occurred.

CV3626 no execution model; exiting

CodeView needs an execution model in order to function.

Check your TOOLS.INI file, and make sure there is a Native entry specified.

CV5013 no expression evaluators found; exiting

CodeView needs at least one expression evaluator in order to operate.

Check the [cv] or [cvw] section of your TOOLS.INI file and specify at least one Eval entry.

CV5005 no file selected

A module must be selected before OK is chosen.

To exit the dialog box without selecting a module, choose Cancel.

CV1011 no previous regular expression

The Repeat Last Find command was executed, but a regular expression (search string) was not previously specified.

CV1061 no second monitor connected to system

CodeView was invoked with the /2 option, but there was only one monitor for CodeView to use.

CV1031 no source lines at this address

An attempt was made to view an address that does not have source code.

This error can be caused by debugging a program whose source has been modified after it was compiled. Recompile the file before debugging it with CodeView.

CV1058 no source window open

A command was entered to manipulate the contents of a Source window, but a Source window was not open.

CV0028 no space left on device

The disk does not have any space available for writing.

One of the following may have occurred:

- CodeView could not find room for writing a temporary file.
- An attempt was made to write to a disk that was full.

CV0002 no such file or directory

The specified file does not exist or a path does not specify an existing directory.

Check the file or directory name in the most recent command.

One of the following may have occurred:

- The View Source (**VS**) command or the Open Source command from the File menu was used to view a nonexistent file.
- An attempt was made to print to a nonexistent file or directory.

CV0018 no such file or directory

The specified file does not exist or a path does not specify an existing directory.

Check the file or directory name in the most recent command.

One of the following may have occurred:

- The View Source (**VS**) command or the Open Source command from the File menu was used to view a nonexistent file.
- An attempt was made to print to a nonexistent file or directory.

CV3630 no symbol handler found; exiting

A symbol handler dynamic-link library (DLL) could not be found. The DLLs that CodeView uses must be in a location specified by the Cvdllpath entry in the [cvw] or [cv] section of TOOLS.INI.

CV3625 no transport layer; exiting

CodeView needs a transport layer to make appropriate calls to the operating system in local debugging and to a remote computer in remote debugging.

Check your TOOLS.INI file, and make sure there is a Transport entry in the [cv] or [cvw] CodeView section.

CV5009 no watch expression to delete

An attempt was made to delete one or more watch variables (watch expressions), but watch expressions are not currently selected.

CV1039 not a text file

An attempt was made to load a file that is not a text file. The file may be a binary data file or an executable program.

This error can also occur if the first line of a file includes characters that are in the range of ASCII 0 to 8, 14 to 31, or 127 (0x0 to 0x8, 0xE to 0x1F, or 0x7F).

The Source window can only be used to view text files.

CV0007 number of arguments exceeds DOS limit of 128

CodeView is not able to restart the program that is being debugged because the number of arguments to the executable program exceeds the limit of 128.

CV0012 out of memory

CodeView was unable to allocate or reallocate the memory that it required because not enough memory was available.

Possible solutions include the following:

- Recompile without symbolic information in some of the modules. CodeView requires memory to hold information about the program being debugged. Compile some modules with the /Zd option instead of /Zi, or don't use either option.
- Remove other programs or drivers running in the system that could be consuming significant amounts of memory.
- Decrease the settings in CONFIG.SYS for FILES and BUFFERS.

CV3608 out of memory

CodeView needed additional memory, but insufficient memory was available.

Possible solutions include the following:

- Remove some drivers or applications that have been loaded in high memory.
- Recompile without symbolic information in some of the modules. CodeView requires memory to hold information about the program being debugged. Compile some modules with the /Zd option instead of /Zi, or don't use either option.
- Remove other programs or drivers running in the system that could be consuming significant amounts of high memory.
- Free some memory by removing terminate-and-stay-resident (TSR) software.
- Remove unneeded watch expressions or breakpoints.

CV1047 **overlay not resident**

An attempt was made to access machine code from an overlay section of code that is not currently resident in memory.

Execute the program until the overlay is loaded.

CV5012 **packed executable file**

CodeView cannot step through the beginning of files that are linked with the /EXEPACK option. There are two solutions to this problem:

- Relink without this option to debug the file, and then switch back to linking with /EXEPACK for the release version of your program.
- Execute the program through startup code, and set breakpoints only after the program has entered main.

CV0003 **program has terminated: restart to continue**

CodeView has detected a termination request by the program being debugged.

The program cannot be executed because it has terminated and has not been restarted. Program memory remains allocated and may still be examined at this point.

To run the program again, reload it using the Restart command.

CV1012 **regular expression too long**

The regular expression was too long or complex.

Use a simpler or more general regular expression.

CV0103 **relink** ***filename*** **with current linker**

This version of CodeView expects the executable file to be in the format produced by the current version of the linker.

Make sure PWB, NMAKE, or the compiler is not running an older version of the linker.

CV2403 response files cannot be nested

A response file cannot refer to another response file.

CV1017 syntax error

The command contained a syntax error.

This error is probably caused by an invalid command or expression.

CV0024 too many open files

CodeView could not open a file it needed because a file handle was not available.

Increase the number of file handles by changing the FILES setting in CONFIG.SYS to allow a larger number of open files. FILES=50 is the recommended setting.

The program being debugged may have so many files open that all available handles are exhausted. Check that the program has not left files open unnecessarily. The first four handles are reserved by the operating system.

Additional files can be made available by closing source windows. If more files are needed, set `helpbuffers=0` in the [pwb] section of TOOLS.INI. As a result, Help cannot be used but several file handles will be made available.

CV1053 TOOLS.INI not found

The directory listed in the INIT environment variable did not contain a TOOLS.INI file.

Check the INIT variable to be sure that it points to the correct directory.

CV3629 too many execution models: choose one

Only one execution model can be selected at a time.

Additional execution models should be removed.

CV3628 too many transport layers selected: choose one

Only one transport layer can be selected at one time.

CV1007 unable to open file

The specified file cannot be opened.

One of the following may have occurred:

- The file may not exist in the specified directory.
- The filename was misspelled.
- The file's attributes are set so that it cannot be opened.
- A locking or sharing violation occurred.

CV4502 unable to open the requested memory window

CodeView could not open a Memory window.

The only valid window IDs are 0 and 1. You may need to close some windows.

CV1021 unknown format specifier; specify one of A,B,I,IU,IX,L,LU,LX,R,RL,RT

An unknown format specifier was given to a View Memory (**VM**), Memory Dump (**MD**), or Memory Enter (**ME**) command.

The valid format specifiers are:

A	ASCII
B	byte
I	16-bit signed decimal integer
IU	16-bit unsigned decimal integer
IX	16-bit hexadecimal integer
L	32-bit signed decimal integer
LU	32-bit unsigned decimal integer
LX	32-bit hexadecimal integer
R	32-bit single-precision integer
RL	64-bit double-precision floating point
RT	80-bit 10-byte real (long double)

This error is probably due to a mistyped command.

CV2402 unknown option *option*; ignored

The specified option was not a valid option.

Check that the option was typed correctly.

CV1018 unknown symbol

The symbolic name specified could not be found.

One of the following may have occurred:

- The specified name was misspelled.
- The wrong case was used when case sensitivity was on. Case sensitivity is toggled by the Case Sensitivity command from the Options menu or is set by the Option (O) Command-window command.
- The module containing the specified symbol may not have been compiled with the /Zi option to include symbolic information.
- A search was made for an undefined label or function.

CV0102 unpacked CodeView information in *filename*: use CVPACK

For this version of CodeView, you must process all executable files using CVPACK, which compresses the debugging information in the file.

Pass the file through CVPACK.EXE before starting CodeView.

CV1040 video mode changed without /S option

The program being debugged changed screen modes, and CodeView was not set for swapping. The program output is now damaged or unrecoverable.

To be able to view program output, exit CodeView and restart it with the Swap (/S) option.

CV1064 window could not be opened

CodeView tried to open a window, but failed to do so.

This error is probably caused by a lack of memory available to CodeView.

Exit CodeView and make more memory available, then restart CodeView.

CV3623 wrong DLL *filename*

CodeView expected one type of dynamic-link library (DLL) but read in a different type. This error is probably caused by specifying an incorrect filename in the TOOLS.INI file. For example, you may have specified an execution model DLL in the expression evaluator entry.

Expression Evaluator Error Messages

Number Expression Evaluator Error Message

CXX0000 no error condition

An error has not occurred, and this message should not appear.

You can continue debugging normally.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

CXX0001 exception executing user function

The code being executed caused a general protection fault.

CXX0002 error accessing user memory

The expression attempted to reference memory that is not allocated to the program being debugged.

CXX0003 internal error in expression evaluator

The CodeView expression evaluator encountered an internal error.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

CXX0004 syntax error

The syntax of the expression is incorrect.

Retype the expression with the correct syntax.

CXX0005 operator not supported

An unsupported C/C++ operator was specified in an expression.

You can usually write an equivalent expression using supported C/C++ operators.

CXX0006 missing left parenthesis

Unbalanced parentheses were found in the expression.

Retype the expression with balanced parentheses.

CXX0007 missing right parenthesis

Unbalanced parentheses were found in the expression.

Retype the expression with balanced parentheses.

CXX0008 missing double quotation mark (") at end of string

The double quotation mark (") expected at the end of the string literal was missing.

Retype the expression, enclosing the string literal in double quotation marks.

CXX0009 missing single quotation mark (') after character constant

The single quotation mark (') expected at the end of the character constant was missing.

Retype the expression, enclosing the character constant in single quotation marks.

CXX0010 missing left bracket

The expression contains unbalanced square brackets.

Retype the expression with balanced square brackets.

CXX0011 missing right bracket

The expression contains unbalanced square brackets.

Retype the expression with balanced square brackets.

CXX0012 missing left curly brace

The expression contains an unbalanced curly brace.

Retype the expression with balanced curly braces.

CXX0013 missing operator

An operator was expected in the expression but was not found.

Check the syntax of the expression.

CXX0014 missing operand

An operator was specified without a required operand.

Check the syntax of the expression.

CXX0015 expression too complex (stack overflow)

The entered expression was too complex or nested too deeply for the amount of storage available to the C/C++ expression evaluator.

Overflow usually occurs because of too many pending calculations.

Rearrange the expression so that each component of the expression can be evaluated as it is encountered, rather than having to wait for other parts of the expression to be calculated.

Break the expression into multiple commands.

CXX0016 constant too big

The CodeView C/C++ expression evaluator cannot accept an unsigned integer constant larger than 4,294,967,295 (0xFFFFFFFF hexadecimal), or a floating-point constant whose magnitude is larger than approximately 1.8E+308.

CXX0017 symbol not found

A symbol specified in an expression could not be found.

One possible cause of this error is a case mismatch in the symbol name. Since C and C++ are case-sensitive languages, a symbol name must be given in the exact case in which it is defined in the source.

CXX0018 bad register name

A specified register does not exist or cannot be displayed.

CodeView can display the following registers:

AX
BX
CX
DX
SP
BP
SI
DI
DS
ES
SS
CS
IP
FL

When running under DOS on an 80386 machine, the 80386 option can be selected to display the following registers:

EAX
EBX
ECX
EDX
ESP
EBP
ESI
EDI
DS
ES
FS
CS
GS
SS
EIP
EFL

CXX0019 bad type cast

The CodeView C/C++ expression evaluator cannot perform the type cast as written.

One of the following may have occurred:

- The specified type is unknown.
- There were too many levels of pointer types.

  For example, the type cast:

  ```
  (char far * far *)h_message
  ```

  cannot be evaluated by the CodeView C/C++ expression evaluator.

CXX0020 operand types bad for this operation

An operator was applied to an expression with an invalid type.

For example, it is not valid to take the address of a register or subscript an array with a floating-point expression.

CXX0021 struct or union used as scalar

A structure or union was used in an expression, but no element was specified.

When manipulating a structure or union variable, the name of the variable may appear by itself, without a field qualifier. If a structure or union is used in an expression, it must be qualified with the desired specific element.

Specify the element whose value is to be used in the expression.

CXX0022 function call before _main

The CodeView C/C++ expression evaluator cannot evaluate a function before CodeView has entered the function **_main**. The program is not properly initialized until **_main** has been called.

Execute `g main;p` to enable function calls in expressions.

CXX0023 bad radix

The radix specified is not recognized by the CodeView C/C++ expression evaluator. Only decimal, hexadecimal, and octal radixes are valid.

CXX0024 operation needs l-value

An expression that does not evaluate to an l-value was specified for an operation that requires an l-value.

An l-value is an expression that refers to a memory location and appears on the left side of an assignment statement.

For example, `buffer[count]` is a valid l-value because it points to a specific memory location. The logical comparison `zed != 0` is not a valid l-value because it evaluates to TRUE or FALSE, not a memory address.

CXX0025 operator needs struct/union

An operator that takes an expression of structure or union type was applied to an expression that is not a structure or union.

Components of class, structure, or union variables must have a fully qualified name. Components cannot be entered without full specification.

CXX0026 bad format string

A format string was improperly specified.

Check the syntax of the expression.

CXX0027 invalid operand

An address operand (l-value) was expected in this expression, but some other operand was found.

Check the syntax of the expression.

See Help for more recent information on this error.

CXX0028 not struct/union element

An expression of the form `Struct.Member` or `pStruct->Member` was specified, but the member is not an element of the structure.

This error can be caused by an expression with incorrectly placed parentheses.

CXX0029 not struct pointer

The member-selection operator (**->**) was applied to an expression that is not a pointer to a structure.

Check that the parentheses within the expression are correct, or type cast the address expression to the appropriate structure pointer type.

CXX0030 expression not evaluatable

The expression could not be evaluated as written.

This error is frequently caused by dereferencing an invalid pointer.

Check that the syntax of the expression is correct and that the case for each symbol matches its definition in the program.

CXX0031 expression not expandable

The CodeView C/C++ expression evaluator encountered an internal error.

You may be able to write an equivalent expression that can be evaluated correctly.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

CXX0032 divide by 0

The expression contains a division by zero, which is illegal. This divisor may be the literal number zero, or it may be an expression that evaluates to zero.

CXX0033 error in OMF type information

The executable file did not have a valid OMF (object module format) for debugging by CodeView.

One of the following may have occurred:

- The executable file was not created with the linker released with this version of CodeView. Relink the object code using the current version of LINK.EXE.
- The executable file was not created with the high-level language released with this version of CodeView. Recompile the program with the current version of the compiler.
- The .EXE file may have been corrupted. Recompile and relink the program.

CXX0034 types incompatible with operator

The specified operand types are not legal for the operation.

For example, a pointer cannot be multiplied by any value.

You may need to type cast the operands to a type compatible with the operator.

CXX0035 overlay not resident

An attempt was made to access an overlay that is not currently resident in RAM.

Execute the program until the overlay is loaded.

CXX0036 bad context {...} specification

This message can be generated by one of the following incorrect uses of the context resolution operator ({}):

- The syntax of the context resolution operator ({ }) was given incorrectly.

  The syntax of the context resolution operator is:

  ```
  {[function],[module],[dll]}expression
  ```

  This specifies the context of expression. The operator has the same precedence and usage as a type cast.

  Trailing commas can be omitted. If `[function]`, `[module]`, or `[dll]` contains a literal comma, the entire name must be enclosed in parentheses.

- The function name was spelled incorrectly or does not exist in the specified module or dynamic-link library.

  Since C/C++ is a case-sensitive language, the case of the function name must exactly match the one found in the source. The expression evaluator ignores the CodeView case-sensitivity state set with the OC command or the Case Sensitive command in the Options menu.

- The module or DLL could not be found.

  Check the full path name of the specified module or DLL.

CXX0037 out of memory

The CodeView C/C++ expression evaluator ran out of memory evaluating the expression.

CXX0038 function argument count and/or type mismatch

The specified function call does not match the prototype for the function.

Retype the call with the correct number of arguments. Type cast each argument to match the prototype, as necessary.

CXX0039 symbol is ambiguous

The CodeView C++ expression evaluator cannot determine which instance of a symbol to use in an expression. The symbol occurs more than once in the inheritance tree.

You must use the scope resolution operator (::) to explicitly specify which instance to use in the expression.

CXX0040 function requires implicit conversion

Implicit conversions involving constructor calls are not supported by the CodeView C++ expression evaluator.

CXX0041 class element must be static member or member function

A nonstatic member of a class, structure, or union was used without specifying which instantiation of the class to use.

Only static data members or member functions can be used without specifying an instantiation.

CXX0043 this pointer used outside member function

The **this** pointer can only be used for nonstatic member functions.

CXX0044 use of _based(void) pointer requires :> operator

A pointer based on **void** cannot be used directly. You must form a complete pointer using the **:>** operator.

CXX0045 not a function

An argument list was supplied for a symbol in the program that is not the name of a function.

For example, this error is generated for the expression:

```
queue( alpha, beta)
```

when `queue` is not a function.

CXX0046 argument list required for member function

An expression called a member function but did not specify any actual parameters.

CXX0047 argument list does not match a function

An expression called a function with an actual parameter list that did not match the formal parameter list of any function with the same name defined in the program.

Overloaded functions can be called only if there is an exact parameter match or a match that does not require the construction of an object.

CXX0048 calling sequence not supported

A function specified in the expression uses a calling sequence not supported by the CodeView C/C++ expression evaluator. You cannot call this function in a CodeView expression.

CXX0049 obsolete OMF—please relink program

The program used an old OMF (object module format).

The program must be linked with LINK version 5.30 or later and packed with CVPACK version 4.0 or later.

CXX0050 left side of :: must be class/struct/union

The symbol on the left side of the scope resolution operator (::) was not a class, structure, or union.

CXX0051 more than one overloaded symbol specified in breakpoint

CodeView could not determine which of more than one overloaded symbol to use as a breakpoint.

CXX0052 member function not present

A member function was specified as a breakpoint but could not be found or was not defined. This error can be caused by setting a breakpoint at a function that has been inlined.

Recompile the file with inlining forced off (/Ob0) to set a breakpoint in this function.

CXX0053 nonfunction symbol match while binding breakpoints

A symbol used as a breakpoint was not a function. This error can be caused by specifying a data member as a breakpoint.

CXX0054 register in breakpoint expression illegal

A register cannot be used in a breakpoint expression.

CXX0055 ambiguous symbol in context operator

A symbol in the context resolution operator ({}) referred to more than one symbol in the program.

The scope resolution operator (::) may be able to resolve the ambiguity.

CXX0056 error in line number

An invalid line number was specified.

CXX0057 no code at line number

Code was not generated for the specified line number. It cannot be used as a breakpoint.

CXX0058 overloaded operator not found

A class type was specified as the left operand in an expression, but an overloaded operator was not defined for the class.

CXX0059 left operand is class not a function name

The left operand of a function call was a class name and could not be resolved to a function call. This error can be caused by omitting the name of a member function in an expression.

CXX0060 register is not available

An expression specified a register than cannot be used.

This error can be caused by trying to access a register that does not exist on the machine running CodeView, for example, accessing 80386-specific registers on an 8088-based machine.

CXX0061 function nesting depth exceeded

The expression contained a function whose nesting depth was greater than the permitted limit.

The expression should be modified to reduce the nesting depth.

CXX0062 constructor calls not supported

An expression made a call to a constructor.

Expressions cannot make explicit calls to constructors or make conversions that require a call to a constructor.

Command-Line Error Messages

Messages that indicate errors on the command line used to invoke the compiler fall into two categories:

- Error messages. These messages have the following format:

  ```
  Command line error D2xxx: messagetext
  ```

- Warning messages. These messages have the following format:

  ```
  Command line warning D4xxx: messagetext
  ```

If possible, the driver continues operating, printing error and warning messages. Errors prevent the CL driver from starting compilation.

Command-Line Error Messages

Number **Command-Line Error Message**

D2000 UNKNOWN COMMAND-LINE ERROR
Contact Microsoft Product Support Services

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

D2001 **too many symbols predefined with /D**

The number of predefined symbols exceeded the limit of 30 on the CL command line or 20 on the FL command line.

Check the CL or FL environment variable for option specifications.

D2002 **memory-model conflict**

More than one memory-model option was specified.

For example, the following command line generates this error:

```
cl /AS /AM program.c
```

Check the CL or FL environment variable for option specifications.

D2003 **missing source filename**

The command line did not specify a source file to compile.

D2008 **limit of *option* exceeded at *string***

The given option was specified too many times. The given string is the argument to the option that caused the error.

Check the CL or FL environment variable for option specifications.

D2011 **only one floating-point option allowed**

More than one floating-point option (/FP*x*) was specified on the command line.

Check the CL or FL environment variable for option specifications.

D2012 **too many linker arguments**

More than 128 options and object files were passed to the linker with the /link command-line option.

This error can be caused by using filename wildcards (* and ?) to specify files for the linker. If you need to link a large number of .OBJ files, use the LIB utility to combine multiple object files.

Check the CL or FL environment variable for option specifications.

D2013 **incomplete model specification**

Not enough characters were given for the /A*string* option.

For example, the following command line generates this error:

```
cl /As file1.c
```

Two types of options begin with /A:

- The /A*string* customized memory-model option requires three letters in *string*. The letters specify the code-pointer size, data-pointer size, and data-segment setup attributes.
- The /A*x* standard memory-model option requires one uppercase letter for *x*. CL interprets a lowercase letter as part of a customized memory-model specification. An uppercase letter can be combined with lowercase letters to customize a memory model.

D2016 ***option1* and *option2* command-line options are incompatible**

The given command-line options cannot be specified together.

For example, the following command line generates this error:

```
cl /Gw /NDxx program.c
```

In this example, the /Gw and /NDxx options are incompatible because each has a different special-entry sequence.

Check the CL or FL environment variable for option specifications.

D2018 **cannot create linker response file**

The compiler could not create a response file for passing arguments to the linker.

This error can occur when an existing read-only file has the same name as the filename the compiler gives to the response file.

D2019 **cannot overwrite source or object file** *filename*

A source or object filename was specified as an output file with the /Fo option. The compiler cannot overwrite input files.

Check the CL or FL environment variable for option specifications.

D2020 *option* **option requires extended keywords to be enabled (/Ze)**

The /Gc or /Gr option was specified on the same command line as the /Za option.

The /Gc and /Gr options require the extended keyword **__cdecl** to be enabled. To enable **__cdecl** and make library functions accessible, use the default /Ze option instead of /Za.

Check the CL environment variable for option specifications.

D2021 **invalid numeric argument** *number*

A number greater than 65,534 was specified as a numeric argument.

D2022 **cannot open** *messagefile*

The given file was not in the current directory or a directory specified in the PATH environment variable. The file contains a brief summary of compiler command-line syntax and options.

Move this file to the current directory or a directory in the current path. If this file cannot be found, run the SETUP program to copy it from the distribution disks.

D2023 **invalid model specification—flat model only**

A 16-bit memory model was specified to the 32-bit targeted compiler. The 32-bit compiler can create only flat-model executable files.

Remove any conflicting memory-model specifications (such as /AL or /AH) from the command line.

Check the CL environment variable for option specifications.

D2027 **cannot execute** *component*

The compiler could not run the given compiler component or linker.

One of the following may have occurred:

- Not enough memory was available to load the component. If this error occurred when NMAKE invoked the compiler, run the compiler outside of the makefile.
- The current operating system could not run the component. Make sure that the path points to the executable files appropriate to your operating system.
- The component was corrupted. Recopy the component from the distribution disks using the SETUP program.
- An option was specified incorrectly. For example, the following CL command generates this error:

```
cl /B1 file1.c
```

D2028 **too many open files; cannot redirect** *filename*

Redirection of one of the standard stream files was not possible because too many files were already open and a duplicate handle could not be created.

To increase the number of file handles available under DOS, change the FILES setting in CONFIG.SYS. FILES=50 is the recommended setting.

D2030 **INTERNAL COMPILER ERROR in** *component*
Contact Microsoft Product Support Services

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

D2031 **too many command-line options**

More than 128 options were specified to the compiler.

This error can be caused by using wildcards (* and **?**) to specify a large number of files to compile or link.

Check the CL or FL environment variable for option specifications.

Command-Line Warning Messages

Number **Command-Line Warning Message**

D4000 **UNKNOWN COMMAND-LINE WARNING**
Contact Microsoft Product Support Services

Note the circumstances of the warning and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

D4001 **listing overrides assembly output**

An assembly listing was not generated because another listing option (/Fc or /Fl) was specified. The other option took effect.

For example, the following command line generates this warning:

```
cl /Fc /Fa program.c
```

To create both listings, compile separately with each option.

Check the CL or FL environment variable for option specifications.

D4002 **ignoring unknown option** *option*

The compiler did not recognize the given command-line option; the option was ignored.

Check the CL or FL environment variable for option specifications.

D4003 **processor-option conflict**

More than one /G*n* option was specified with conflicting values for *n*. The compiler used the last one specified on the command line.

For example, the following command line generates this warning:

```
cl /G2 /G0 program.c
```

In this example, the compiler assumed /G0.

Check the CL or FL environment variable for option specifications.

D4005 **cannot find** *component*;
Please enter new filename (full path) or CTRL+C to quit:

The compiler was unable to find the given component in the current directory or in a directory in the PATH environment variable.

Move this file to the current directory or a directory in the current path. If this file cannot be found, run the SETUP program to copy it from the distribution disks.

D4007 *option1* **requires** *option2*; **option ignored**

An option was specified without a required related option. The compiler ignored *option1*.

For example, the following command line generates this warning:

```
cl /C program.c
```

In this example, the /C option was given instead of /c. The /C option must be used with /E, /P, or /EP.

Check the CL or FL environment variable for option specifications.

D4008 **nonstandard model; assuming large model**

A character other than M, L, or H was specified with FL's /A option. FL assumed /AL.

D4009 **threshold only for far or huge data; ignored**

A data size threshold was specified for near data.

The /Gt command-line option cannot be used in memory models that have a single data segment. Only compact, large, and huge models have multiple data segments and support /Gt.

Check the CL or FL environment variable for option specifications.

D4011 **preprocessing overrides source listing**

A source listing was not generated because a preprocessor listing option was specified.

To generate both a source listing and a preprocessor listing, run CL twice: once with the /Fs (source listing) option and once with the /E, /P, or /EP (preprocessed listing) option.

Check the CL or FL environment variable for option specifications.

D4012 **function declarations override source listing**

A source listing was not generated because a function prototype listing was requested.

To generate both a source listing and a function prototype declaration listing, run CL twice: once with the /Fs (source listing) option and once with the /Zg (function prototype listing) option.

Check the CL or FL environment variable for option specifications.

D4013 **combined listing overrides object listing**

When /Fc is specified along with /Fl, the combined listing specified by /Fc is created.

To generate both a combined listing and an object listing, run CL twice: once with the /Fc (combined listing) option and once with the /Fl (object-code listing) option.

Check the CL or FL environment variable for option specifications.

D4014 **invalid value** *number1* **for** *option***; assuming** *number2*

The given option was specified with an invalid numeric argument. The compiler assumed the value *number2*.

For example, the following command line generates this warning:

```
cl /Zp3 program.c
```

In this example, `3` is an invalid value. Valid arguments for the /Zp option are 1, 2, and 4.

D4018 **.DEF files supported for segmented executable files only**

A module-definition file was specified on the command line, but an /Lr or /Lc option was also specified. The /Lr and /Lc options are used to create DOS executable files.

Module-definition files are used to create EXEs and DLLs for Windows, and the Microsoft DOS Extender.

Check the CL or FL environment variable for option specifications.

D4019 **string too long; truncated to** *number* **characters**

A string longer than 40 characters was specified with the /ND, /NM, /NT, /Ss, or /St option. The compiler truncated the string.

D4020 *option* **: missing argument; option ignored**

A command-line option required an argument, but nothing was specified. CL ignored the option.

D4021 **no action performed**

A contradictory set of filenames and switches caused the compiler to perform no operation.

This warning can be generated by giving the /c compile-only command-line option and specifying no .C, .CPP, or .CXX files to compile.

D4022 **option *option* invalid for *number*-bit target**

The given command-line option was not available when generating code for this size target. The option was ignored.

This warning can be caused by using the /G3 command-line option from the 16-bit targeted compiler.

HELPMAKE Error Messages

Microsoft Help File Maintenance Utility (HELPMAKE) generates the following error messages:

- Fatal Errors (H1*xxx*) cause HELPMAKE to stop execution. No output file is produced.
- Errors (H2*xxx*) do not prevent an output file from being produced, but parts of the conversion are not completed.
- Warnings (H4*xxx*) do not prevent an output file from being produced, but problems may exist in the output.

HELPMAKE Fatal Error Messages

Number **HELPMAKE Fatal Error Message**

H1000 **/A requires character**

The /A option requires an application-specific control character.

The correct form is:

```
/Ac
```

where *c* is the control character.

H1001 **/E compression level must be numeric**

The /E option requires either no argument or a numeric value in the range 0–15.

The correct form is:

```
/En
```

where *n* specifies the amount of compression requested.

H1002 **multiple /O parameters specified**

Only one output file can be specified with the /O option.

H1003 invalid /S file-type identifier

The /S option was given an argument other than 1, 2, or 3.

The /S option requires specification of the type of input file. An invalid file-type identifier was specified.

The correct form is:

```
/Sn
```

where *n* specifies the format of the input file. Valid values are 1 (RTF), 2 (QuickHelp format), and 3 (minimally formatted ASCII).

H1004 /S requires file-type identifier

The /S option requires specification of the type of input file. There was no file-type identifier specified.

The correct form is:

```
/Sn
```

where *n* specifies the format of the input file. Valid values are 1 (RTF), 2 (QuickHelp format), and 3 (minimally formatted ASCII).

H1005 /W fixed width invalid

An invalid width was specified with the /W option. The valid range is 11–255.

H1006 multiple /K parameters specified

The option for specifying a keyword separator file, /K, was used more than once on the HELPMAKE command line.

Only one file containing separator characters can be specified.

H1050 option invalid with /DS

The /C, /L, and /O options for encoding are invalid with the /DS option for decoding.

H1051 improper arguments for /D

The /D option permits either no argument or an S or U argument. In addition, /D is invalid with the /C or /L option.

H1052 encode requires /O option

Database encoding was requested without a specified output-file name for the operation.

H1053 **compression level exceeds 15**

A value greater than 15 was specified with the /E option.

The /E option requires either no argument or a numeric value in the range 0–15. The correct form is:

```
/En
```

where *n* specifies the amount of compression requested.

H1097 **no operation specified**

The HELPMAKE command line did not contain an option for encoding, decoding, or Help.

HELPMAKE requires the /E, /D, /H, or /? option.

H1098 **unrecognized option**

An unrecognized name followed the option indicator.

An option is specified by a forward slash (/) or a dash (–) and an option name.

H1099 **syntax error on command line**

HELPMAKE cannot interpret the command line.

H1100 **cannot open file**

One of the files specified on the HELPMAKE command line could not be found or created.

H1101 **error writing file**

The output file could not be written, probably because the disk is full.

H1102 **no input file specified**

In an encoding operation, no input Help text file was specified.

H1103 **no context strings found**

No context strings were found in the input stream during encoding.

Either the file is empty or the specified /S value does not correspond to the Help text formatting.

H1104 **no topic text found**

No topic text was found in the Help text file.

Either the file is empty or the specified /S value does not correspond to the Help text formatting.

H1107 cannot overwrite input file

The /DS option for splitting a concatenated Help file was specified, but the Help file contained a database with the same name as the Help file. It may be that the Help file is not a concatenated file and contains only one database, and the database has the same name as its physical Help file.

One of the following may be a solution:

- Rename the Help file so that the filename does not match any of the database names.
- Run HELPMAKE from a directory other than the one that holds the physical Help file. Since HELPMAKE creates the split files in the current directory, no filename conflict occurs.

H1200 insufficient memory to allocate context buffer

There was insufficient memory to run HELPMAKE.

HELPMAKE requires 256K free memory.

H1201 insufficient memory to allocate utility buffer

There was insufficient memory to run HELPMAKE.

HELPMAKE requires 256K free memory.

H1250 not a valid compressed Help file

The input file specified for a decompression operation is not a valid Help database file.

H1251 cannot decompress locked Help file

An attempt was made to decompress a Help database file that is locked.

A file is locked if the /L option is specified when the Help file is created.

H1300 word too long in RTF processing

A single word was longer than the specified format width (set by the /W option) or was found to be longer than 128 characters when HELPMAKE was reformatting a paragraph.

H1302 attribute stack overflow processing RTF

RTF attribute groups are nested too deeply. HELPMAKE supports a maximum of 50 levels of attribute-group nesting in RTF format.

H1303 unknown RTF attribute

An unknown RTF formatting command was found.

One of the following may have occurred:

- A new RTF attribute was used. HELPMAKE recognizes a set of attributes that were current at the time this version of HELPMAKE was created. It interprets some of the attributes and knows to ignore the others. Any RTF attribute defined after HELPMAKE was created is not known by HELPMAKE and will cause this error.
- The RTF file is corrupted.

H1304 **topic too large**

A topic exceeded the limit for the size of topics.

A single topic cannot exceed 64K.

H1305 **topic text without context string**

The source file contained topic text that was not preceded by a .context definition.

H1900 **internal virtual memory error**

This message indicates an internal HELPMAKE error.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

H1901 **out of local memory**

This message indicates an internal HELPMAKE error.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

H1902 **out of disk space for swap file**

The current drive or directory is full.

HELPMAKE uses a temporary swap file, written to the current drive and directory. The temporary file can grow to 1.5 times the size of the input files (for large Help files) and is not removed until the final Help file is completed.

H1903 **cannot open swap file**

HELPMAKE was unable to create its temporary swap file on the current drive and directory for one of the following reasons:

- The current drive or directory is full.
- The device cannot be written to.

H1990 **internal compression error**

This message indicates an internal HELPMAKE error.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

HELPMAKE Error Messages

Number **HELPMAKE Error Message**

H2000 **line too long, truncated**

A line exceeded the fixed width specified by the /W option or the default of 76 characters. HELPMAKE truncated the extra characters.

H2001 **duplicate context string**

A context string preceded more than one topic in a Help database. A context string can be associated with only one block of topic text.

H2002 **zero length hot spot**

A cross-reference was specified, but the word or anchored text associated with it was of zero length.

With no visible text to associate with the cross-reference, the hot spot will be inoperative. This error is issued as a warning and does not prevent the building of a Help file. However, some applications may not be able to use the resulting Help file correctly.

The following example will cause this error:

```
\a\vcross_reference\v
```

H2003 **unrecognized dot command**

A line in the source file contained a dot (.) in column 1, but it was not followed by a command recognized by HELPMAKE.

HELPMAKE Warning Messages

Number HELPMAKE Warning Message

H4000 **keyword compression analysis table size exceeded no further new words will be analyzed**

The maximum number (16,000) of unique keywords has been encountered during keyword compression. This happens only in very large Help files. No further keywords will be included in the analysis. HELPMAKE continues to analyze how frequently words occur that it has already encountered.

H4002 **reference to undefined local context**

A string specifying a local context was used in a cross-reference but was not defined in a .context statement.

A local context begins with an at sign (@). Each local context that is used must be defined in a .context statement in one of the input files to HELPMAKE.

H4003 **negative left indent**

Topic text in an RTF file was formatted with a left indent to a position to the left of column 1. HELPMAKE deleted all text preceding column 1.

IMPLIB Error Messages

Microsoft Import Library Manager (IMPLIB) generates the following error messages:

- Fatal errors (IM16*xx*) cause IMPLIB to stop execution.
- Errors (IM26*xx*) prevent IMPLIB from creating an import library.
- Warnings (IM46*xx*) indicate possible problems in the output file being created.

IMPLIB Fatal Error Messages

Number IMPLIB Fatal Error Message

IM1600 **error writing to output file—***message*

IMPLIB could not create the import library for the given reason.

Probably the drive or directory where the import library is being created is full.

IM1601 **out of memory, near/far heap exhausted**

There was not enough room in memory for the heap needed by IMPLIB.

Increase the available memory. Some ways to do this include:

- Remove TSR (terminate-and-stay-resident) programs
- Run IMPLIB outside of an NMAKE session
- Run IMPLIB outside of a shell

IM1602 **syntax error in module-definition file**

IMPLIB could not understand the contents of a .DEF input file.

IM1603 *filename* **: cannot create file—***message*

IMPLIB could not create the given file for the given reason.

One of the following may be a cause:

- The file already exists with a read-only attribute.
- There is insufficient disk space to create the file.
- The drive cannot be written to.

IM1604 *filename* **: cannot open file—***message*

IMPLIB could not find the specified module-definition (.DEF) file or dynamic-link library (DLL) for the given reason.

IM1605 **too many nested include files in module-definition file**

A module-definition (.DEF) file contained an **INCLUDE** statement specifying a nested set of included files that exceeded the limit for nesting. The limit is 10 levels.

IM1606 **missing or invalid include file name**

A syntax error occurred in an **INCLUDE** statement in a module-definition (.DEF) file.

One of the following may have occurred:

- A filename was not specified.
- More than one filename was specified.
- A long filename was specified without being enclosed in quotation marks or was enclosed in one single and one double quotation mark.

IM1607 *extension* **: invalid extension for target library**

The given extension was specified for the import library.

An import library cannot be given a .DEF or .DLL extension.

IM1608 **no .DLL or .DEF source file specified**

No input file was specified on the IMPLIB command line.

IMPLIB Error Messages

Number **IMPLIB Error Message**

IM2601 *symbol* **multiply defined**

The given symbol was defined more than once in the input files.

IM2602 **unexpected end of name table in DLL**

A dynamic-link library (DLL) specified to IMPLIIB was corrupted.

IM2603 *filename* **: invalid .DLL file**

IMPLIB did not recognize the given input file as a dynamic-link library (DLL).

IMPLIB Warning Messages

Number **IMPLIB Warning Message**

IM4600 **line** *number* **too long; truncated to 512 characters**

The given line in the module-definition (.DEF) file exceeded the limit on line length. IMPLIB ignored text after the first 512 characters.

IM4601 **unrecognized option** *option***; option ignored**

The given option was not a valid IMPLIB option. IMPLIB used the rest of the command line to try to build an import library.

LINK Error Messages

Microsoft Segmented Executable Linker (LINK) generates the following error messages:

- Fatal errors (L1*xxx*) cause LINK to stop execution.
- Errors (L2*xxx*) do not stop execution but might prevent LINK from creating the main output file.
- Warnings (L4*xxx*) indicate possible problems in the output file being created.

LINK Fatal Error Messages

Number LINK Fatal Error Message

L1001 *option* **: option name ambiguous**

A unique option name did not appear after the option indicator.

An option is specified by a forward slash (/) and a name. The name can be specified by an abbreviation of the full name, but the abbreviation must be unambiguous.

For example, since many options begin with the letter **N**, the following command causes this error:

```
LINK /N main;
```

This error can also occur if the wrong version of the linker is being used. Check the directories in the PATH environment variable for other versions of LINK.EXE.

L1003 **/Q and /EXEPACK incompatible**

LINK cannot be given both the /Q option and the /EXEPACK option.

L1004 *value* **: invalid numeric value**

An incorrect value was specified with a LINK option.

For example, this error occurs if a nonnumeric string is specified with an option that requires a number.

L1005 *option* **: packing limit exceeds 64K**

The value specified with the /PACKC or /PACKD option exceeded the limit of 65,536 bytes.

L1006 *number* **: stack size exceeds 64K–2**

One of the following may have occurred:

- The given value specified with the /STACK option exceeded the limit of 65,534 bytes.
- A space appeared before or after the colon (**:**) between /STACK and the argument specified with it.

L1007 **/OVERLAYINTERRUPT : interrupt number exceeds 255**

An overlay interrupt number greater than 255 was specified with the /OV option value.

Check the *DOS Technical Reference* or other DOS technical manual for information about interrupts.

L1008 **/SEGMENTS : segment limit set too high**

The value specified with the /SEG option exceeded 16,375.

L1009 *value* **: /CPARM : illegal value**

The value specified with the /CPARM option was not in the range 1–65,535.

L1020 **no object files specified**

The object-files field was empty.

LINK requires the name of at least one object file.

L1021 **cannot nest response files**

A response file was specified in a response file.

L1022 **response line too long**

A line in a response file was longer than 255 characters.

To extend a field to another line, put a plus sign (**+**) at the end of the current line.

L1023 **terminated by user**

The LINK session was halted by CTRL+C or CTRL+BREAK.

L1024 **nested right parentheses**

The parentheses for assigning overlays were specified incorrectly.

L1025 **nested left parentheses**

The parentheses for assigning overlays were specified incorrectly.

L1026 **unmatched right parenthesis**

The parentheses for assigning overlays were specified incorrectly.

L1027 **unmatched left parenthesis**

The parentheses for assigning overlays were specified incorrectly.

L1030 **missing internal name**

An **IMPORTS** statement specified an ordinal value but not an internal name for the routine or data item being imported.

An item imported by ordinal must be given an internal name.

L1031 **module description redefined**

The module-definition (.DEF) file contained more than one **DESCRIPTION** statement.

L1032 **module name redefined**

The module-definition (.DEF) file contained more than one **NAME** or **LIBRARY** statement.

L1033 **input line too long;** *number* **characters allowed**

The LINK command line cannot exceed the given number of characters.

L1034 **name truncated to** *string*

A name specified either on the LINK command line or in a module-definition (.DEF) file exceeded 255 characters. The name was truncated to the given string.

This is a warning, not a fatal error. However, it indicates a serious problem. This message may be followed by another error as LINK tries to use the specified name. For example, if the string is a filename, LINK issues an error when it cannot open the file.

L1035 **syntax error in module-definition file**

A statement in the module-definition (.DEF) file was incorrect.

L1040 **too many exported entries**

The program exceeded the limit of 65,535 exported names.

L1041 **resident names table overflow**

The size of the resident names table exceeded 65,535 bytes.

An entry in the resident names table is made for each exported routine designated **RESIDENTNAME** and consists of the name plus three bytes of information. The first entry is the module name.

Reduce the number of exported routines or change some to nonresident status.

L1042 **nonresident names table overflow**

The size of the nonresident names table exceeded 65,535 bytes.

An entry in the nonresident names table is made for each exported routine not designated **RESIDENTNAME** and consists of the name plus three bytes of information. The first entry is the **DESCRIPTION** statement.

Reduce the number of exported routines or change some to resident status.

L1043 **relocation table overflow**

More than 32,768 long calls, long jumps, or other long pointers appeared in the program.

Replace long references with short references wherever possible.

L1044 **imported names table overflow**

The size of the imported names table exceeds 65,535 bytes.

An entry in the imported names table is made for each new name given in the IMPORTS section, including the module names, and consists of the name plus one byte.

Reduce the number of imports.

L1045 **too many TYPDEF records**

An object file contained more than 255 TYPDEF records.

TYPDEF records describe communal variables. (TYPDEF is a DOS term. It is explained in the *Microsoft MS-DOS Programmer's Reference* and in other reference books on DOS.)

This error appears only with programs created by the Microsoft FORTRAN Compiler or other compilers that support communal variables.

L1046 **too many external symbols in one module**

An object file specified more than 1023 external symbols.

Break the object file into smaller files.

L1047 **too many group, segment, and class names in one module**

An object file contained too many group, segment, and class names.

Reduce the number of groups, segments, or classes in the object file, or break the object file into smaller files.

L1048 **too many segments in one module**

An object file had more than 255 segments.

Either create fewer segments or break the object file into smaller files.

L1049 **too many segments**

The program contained more than the maximum number of segments.

The maximum number of segments is set with the /SEG option (in the range 1–16,384). If /SEG is not specified, the default is 128.

If this error occurs when linking a p-code program, recompile and use CL's /NQ option to combine the temporary p-code segments.

L1050 **too many groups in one module**

An object file contained more than 21 group definitions (GRPDEF).

Reduce the number of group definitions or split the module.

(Group definitions are explained in the *Microsoft MS-DOS Programmer's Reference* and in other reference books on DOS.)

L1051 **too many groups**

The program defined more than 20 groups, not counting DGROUP.

Reduce the number of groups.

L1052 **too many libraries**

An attempt was made to link with more than 32 libraries.

Combine libraries, or use modules that require fewer libraries.

L1053 **out of memory for symbol table**

The program had more symbolic information than could fit in available memory. Symbolic information includes public, external, segment, group, class, and file names.

One of the following may be a solution:

- Eliminate as many public symbols as possible.
- Combine object files or segments.
- Link from the command line instead of from a makefile or PWB.
- Remove terminate-and-stay-resident programs or otherwise free some memory.

L1054 **requested segment limit too high**

LINK did not have enough memory to allocate tables describing the requested number of segments. The number of segments is the value specified with the /SEG option or the default of 128.

One of the following may be a solution:

- Link again using the /SEG option to set fewer segments.
- Remove terminate-and-stay-resident programs or otherwise free some memory.

L1056 **too many overlays**

The program defined more than 127 overlays.

L1057 **data record too large**

An LEDATA record in an object module contained more than 1024 bytes of data. This is a translator error. (LEDATA is a DOS term explained in the *Microsoft MS-DOS Programmer's Reference* and in other DOS reference books.)

Note which translator (compiler or assembler) produced the incorrect object module. Please report the circumstances of the error to Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

L1063 **out of memory for debugging information**

LINK ran out of memory for processing debugging information.

Reduce the amount of debugging information by compiling some object files with /Zd instead of /Zi or with neither option.

L1064 **out of memory—near/far heap exhausted**

LINK was not able to allocate enough memory for the given heap.

One of the following may be a solution:

- Reduce the size of code, data, and symbols in the program.
- If the the program is a segmented executable file, put some code into a dynamic-link library.

L1065 **too many interoverlay calls**
use /DYNAMIC:*nnn*; current limit is *number*

The program had more than the given limit of interoverlay calls.

The maximum number of interoverlay calls is set with the /DYNAMIC option (in the range 1–10,922). If /DYNAMIC is not specified, the default is 256.

To determine the setting needed by the program, run LINK with the /INFO option. The output gives the number of interoverlay calls that are generated and the current limit.

L1066 **size of** *overlaynumber* **overlay exceeds 64K**

The overlay represented by the given number exceeded the MOVE size limit of 65,535 bytes.

L1067 **memory allocation error**

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

L1070 *segment* **: segment size exceeds 64K**

A single segment contained more than 65,536 bytes of code or data.

Try changing the memory model to use far code or data as appropriate. If the program is in C, use CL's /NT option or the **__based** keyword (or its predecessor, the **alloc_text** pragma) to build smaller segments.

L1071 **segment _TEXT exceeds 64K–16**

The segment named _TEXT grew larger than 65,520 bytes. This error is likely to occur only in small-model C programs, but it can occur when any program with a segment named _TEXT is linked using the LINK /DOSSEG option.

Small-model C programs must reserve code addresses 0 and 1; this range is increased to 16 for alignment purposes.

Try compiling and linking using the medium or large model. If the program is in C, use CL's /NT option or the **__based** keyword (or its predecessor, the **alloc_text** pragma) to build smaller segments.

L1072 **common area exceeds 64K**

The program had more than 65,536 bytes of communal variables. This error occurs only with programs produced by the Microsoft FORTRAN Compiler or other compilers that support communal variables.

L1073 **file-segment limit exceeded**

The number of physical or file segments exceeded the limit of 255 imposed by Windows for each application or dynamic-link library.

A file segment is created for each group definition, nonpacked logical segment, and set of packed segments.

Reduce the number of segments, or put more information into each segment. Use the /PACKC option or the /PACKD option or both.

L1074 *group* **: group exceeds 64K**

The given group exceeds the limit of 65,536 bytes.

Reduce the size of the group, or remove any unneeded segments from the group. Refer to the map file for a listing of segments.

L1075 **entry table exceeds 64K–1**

The entry table exceeded the limit of 65,535 bytes.

The table contains an entry for each exported routine and for each address that is the target of a far relocation, when **PROTMODE** is not enabled and the target segment is designated **MOVABLE**.

Declare **PROTMODE** if applicable, reduce the number of exported routines, or make some segments **FIXED** if possible.

L1078 **file-segment alignment too small**

The segment-alignment size specified with the /ALIGN option was too small.

L1080 **cannot open list file**

The disk or the root directory was full.

Delete or move files to make space.

L1081 **out of space for run file**

The disk or the root directory was full.

Delete or move files to make space.

L1082 *filename* **: stub file not found**

LINK could not open the file given in the **STUB** statement in the module-definition (.DEF) file.

The file must be in the current directory or in a directory specified by the PATH environment variable.

L1083 **cannot open run file**

One of the following may have occurred:

- The disk or the root directory was full.
- Another process opened or deleted the file.
- A read-only file existed with the same name.
- The floppy disk containing the file was removed.
- A hard-disk error occurred.

L1084 **cannot create temporary file**

One of the following may have occurred:

- The disk or the root directory was full.
- The directory specified in the TMP environment variable did not exist.

L1085 **cannot open temporary file—***message*

LINK could not open a temporary file for the given reason.

One of the following may have occurred:

- The disk or the root directory was full.
- The directory specified in the TMP environment variable did not exist.

L1086 **temporary file missing**

An internal error has occurred.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

L1087 **unexpected end-of-file on temporary file**

A problem occurred with the temporary linker-output file.

One of the following may have occurred:

- The disk that holds the temporary file was removed.
- The disk or directory specified in the TMP environment variable was full.

L1088 **out of space for list file**

The disk or the root directory was full.

Delete or move files to make space.

L1089 *filename* **: cannot open response file**

LINK could not find the given response file.

One of the following may have occurred:

- The response file does not exist.
- The name of the response file was incorrectly specified.
- An old version of LINK was used. Check your path. To see the version number of LINK, run LINK with the /? option.

L1090 **cannot reopen list file**

The original floppy disk was not replaced at the prompt.

Restart the LINK session.

L1091 **unexpected end-of-file on library**

The floppy disk containing the library was probably removed.

Replace the disk containing the library and run LINK again.

L1092 **cannot open module-definition file**

LINK could not find the specified module-definition (.DEF) file.

Check that the name of the .DEF file is spelled correctly.

L1093 *filename* **: object file not found**

LINK could not find the given object file.

Check that the name of the object file is spelled correctly.

L1094 *filename* **: cannot open file for writing**

LINK was unable to open the given file with write permission.

Check the attributes for the file.

L1095 *filename* **: out of space for file**

LINK ran out of disk space for the specified output file.

Delete or move files to make space.

L1096 **unexpected end-of-file in response file**

LINK encountered a problem while reading the response file.

One of the following may be a cause:

- The response file is corrupt.
- The file was deleted between reads.

L1097 **I/O error—***message*

LINK encountered the given input or output error.

L1098 **cannot open include file** *filename—message*

LINK could not open the given include file for the given reason.

An include file is specified in an **INCLUDE** statement in the module-definition (.DEF) file.

L1100 **stub .EXE file invalid**

The file specified in the **STUB** statement in the module-definition (.DEF) file is not a valid DOS executable file.

L1101 **invalid object module**

LINK could not link one of the object files.

Check that the correct version of LINK is being used.

If the error persists after recompiling, note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

L1102 **unexpected end-of-file**

The given library or object file had an invalid format.

L1103 **attempt to access data outside segment bounds**

A data record in an object file specified data extending beyond the end of a segment. This is a translator error.

Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

L1104 *filename* **: invalid library**

The given file had an invalid format for a library.

L1105 **invalid object due to interrupted incremental compile**

Delete the object file, recompile the program, and relink.

L1106 **unknown COMDAT allocation type for** *symbol***; record ignored**

This is a translator error. The given symbol is either a routine or a data item.

Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

L1107 **unknown COMDAT selection type for** *symbol***; record ignored**

This is a translator error. The given symbol is either a routine or a data item.

Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

L1108 **invalid format of debugging information**

This is a translator error.

Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

L1113 **unresolved COMDEF; internal error**

This is a translator error.

Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

L1114 **unresolved COMDAT** *symbol*; **internal error**

This is a translator error. The given symbol is either a routine or a data item.

Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

L1115 *option* **: option incompatible with overlays**

The given option cannot be used when linking an overlaid program.

L1117 **unallocated COMDAT** *symbol*; **internal error**

This is a translator error. The given symbol is either a routine or a data item.

Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

L1123 *segment* **: segment defined both 16-bit and 32-bit**

Define the segment as either 16-bit or 32-bit.

L1127 **far segment references not allowed with /TINY**

The /TINY option for producing a .COM file was used in a program that has a far segment reference.

Far segment references are not compatible with the .COM file format. High-level-language programs cause this error unless the language supports the tiny memory model. An assembly-language program that references a segment address also causes this error.

For example, the following causes this error:

```
mov     ax, seg mydata
```

L1128 **too many nested include files in module-definition file**

Nesting of **INCLUDE** statements in a module-definition (.DEF) file is limited to 10 levels.

L1129 **missing or invalid include file name**

The file specification in an **INCLUDE** statement in the module-definition (.DEF) file was missing or was not a valid filename.

LINK Error Messages

Number **LINK Error Message**

L2000 **imported starting address**

The program starting address as specified in the **END** statement in an assembly-language file is an imported routine. This is not supported by Windows.

L2002 **fixup overflow at** *number* **in segment** *segment*

This error message is followed by one of these strings:

- **target external** *symbol*
- **frm seg** *name1*, **tgt seg** *name2*, **tgt offset** *number*

A fixup overflow is an attempted reference to code or data that is impossible because the source location (where the reference is made "from") and the target address (where the reference is made "to") are too far apart. Usually the problem is corrected by examining the source location.

For information about frame and target segments, see the *Microsoft MS-DOS Programmer's Reference.*

L2003 **near reference to far target at** *offset* **in segment** *segment*
pos: *offset* **target external** *name*

The program issued a near call or jump to a label in a different segment.

This error occurs most often when specifically declaring an external procedure as near that should be declared as far.

This error can be caused by compiling a small-model C program with CL's /NT option.

L2005 **fixup type unsupported at** *number* **in segment** *segment*

A fixup type occurred that is not supported by LINK. This is probably a translator error.

Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

L2010 **too many fixups in LIDATA record**

The number of far relocations (pointer- or base-type) in an LIDATA record exceeds the limit imposed by LINK.

The cause is usually a **DUP** statement in an assembly-language program. The limit is dynamic: a 1,024-byte buffer is shared by relocations and the contents of the LIDATA record. There are 8 bytes per relocation.

Reduce the number of far relocations in the **DUP** statement.

L2011 *identifier* **: NEAR/HUGE conflict**

Conflicting **NEAR** and **HUGE** attributes were given for a communal variable. This error can occur only with programs produced by the Microsoft FORTRAN Compiler or other compilers that support communal variables.

L2012 *arrayname* **: array-element size mismatch**

A far communal array was declared with two or more different array-element sizes (for instance, an array was declared once as an array of characters and once as an array of real numbers). This error occurs only with the Microsoft FORTRAN Compiler and any other compiler that supports far communal arrays.

L2013 **LIDATA record too large**

An LIDATA record contained more than 512 bytes. This is probably a translator error.

Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

L2022 *entry* **(alias** *internalname***) : export undefined**

The internal name of the given exported routine or data item is undefined.

L2023 *entry* **(alias** *internalname***) : export imported**

The internal name of the given exported routine or data item conflicts with the internal name of a previously imported routine or data item.

L2024 *symbol* **: special symbol already defined**

The program defined a symbol name already used by LINK for one of its own low-level symbols. For example, LINK generates special symbols used in overlay support and other operations.

Choose another name for the symbol to avoid conflict.

L2025 *symbol* **: symbol defined more than once**

The same symbol has been found in two different object files.

L2026 **entry ordinal** *number*, **name** *name* **: multiple definitions for same ordinal**

The given exported name with the given ordinal number conflicted with a different exported name previously assigned to the same ordinal. Only one name can be associated with a particular ordinal.

L2027 *name* **: ordinal too large for export**

The given exported name was assigned an ordinal that exceeded the limit of 65,535 (64K–1).

L2028 **automatic data segment plus heap exceed 64K**

The size of the sum of the following exceeds 64K:

- Data declared in DGROUP
- The size of the heap specified in the **HEAPSIZE** statement in the module-definition (.DEF) file
- The size of the stack specified in either the /STACK option or the **STACKSIZE** statement in the .DEF file

Reduce near-data allocation, HEAPSIZE, or stack.

L2029 *symbol* **: unresolved external**

A symbol was declared to be external in one or more modules, but it was not publicly defined in any module or library.

The name of the unresolved external symbol is given, followed by a list of object modules that contain references to this symbol. This message and the list of object modules are written to the map file, if one exists.

One cause of this error is using the /NOI option for files that use case inconsistently in identifiers.

This error can also occur when a program compiled with C/C++ version 7.0 (or later) is linked using /NOD. The /NOD option tells LINK to ignore all default libraries named in object files. C/C++ 7.0 embeds in an object file both the name of the default run-time library and OLDNAMES.LIB. To avoid this error, either specify OLDNAMES.LIB in the *libraries* field or specify /NOD:*library* where *library* is the name of the default run-time library to be excluded from the search.

L2030 **starting address not code (use class 'CODE')**

The program starting address, as specified in the **END** statement of an .ASM file, should be in a code segment. Code segments are recognized if their class name ends in "CODE". This is an error in a segmented executable file.

The error message can be disabled by including the **REALMODE** statement in the module-definition (.DEF) file.

L2041 **stack plus data exceed 64K**

If the total of near data and requested stack size exceeds 64K, the program will not run correctly. LINK checks for this condition only when /DOSSEG is enabled, which is the case in the library startup module for Microsoft language libraries.

For object modules compiled with the Microsoft C or FORTRAN optimizing compilers, recompile with the /Gt command-line option to set the data-size threshold to a smaller number.

This is a fatal LINK error.

L2043 **Quick library support module missing**

The required file QUICKLIB.OBJ was missing. QUICKLIB.OBJ must be linked in when creating a Quick library.

L2044 *symbol* **: symbol multiply defined, use /NOE**

LINK found what it interprets as a public-symbol redefinition, probably because a symbol defined in a library was redefined.

Relink with the /NOE option. If error L2025 results for the same symbol, then this is a genuine symbol-redefinition error.

L2046 **share attribute conflict—segment** *segment* **in group** *group*

The given segment has a different sharing attribute than other segments that are assigned to the given group.

All segments assigned to a group must have the same attribute, either **SHARED** or **NONSHARED**. The attributes cannot be mixed.

L2047 **IOPL attribute conflict—segment** *segment* **in group** *group*

The specified segment is a member of the specified group but has an **IOPL** attribute that is different from other segments in the group.

L2048 **Microsoft Overlay Manager module not found**

Overlays were designated, but an overlay manager was missing.

By default, the overlay manager is the Microsoft Overlay Virtual Environment (MOVE). This is provided in MOVE.LIB, which is a component library of the default combined libraries provided with Microsoft C/C++ version 7.0. The error occurs when LINK cannot find the **_moveinit** routine.

If the /OLDOVERLAY option is specified, the overlay manager is the Microsoft Static Overlay Manager, which is also provided in the default combined libraries.

L2050 **USE16/USE32 attribute conflict—segment** *segment* **in group** *group*

You cannot group 16-bit segments with 32-bit segments.

L2052 *symbol* **: unresolved external; possible calling convention mismatch**

A symbol was declared to be external in one or more modules, but LINK could not find it publicly defined in any module or library.

The name of the unresolved external symbol is given, followed by a list of object modules that contain references to this symbol. The error message and the list of object modules are written to the map file, if one exists.

This error occurs in a C-language program when a prototype for an externally defined function is omitted and the program is compiled with CL's /Gr option. The calling convention for __**fastcall** does not match the assumptions that are made when a prototype is not included for an external function.

Either include a prototype for the function, or compile without the /Gr option.

L2057 **duplicate of** *function* **with different size found; record ignored**

An inconsistent class definition was found.

Check the include files and recompile.

L2058 **different duplicate of** *function* **found; record ignored**

An inconsistent class definition was found.

Check the include files and recompile.

L2060 **size of data block associated with** *symbol* **(16-bit segment) exceeds 64K**

A class had too many virtual functions. The given symbol is the v-table for the class, in the form of a decorated name.

L2061 **no space for data block associated with** *function* **inside** *segment* **segment**

The given function was allocated to the given segment, but the segment was full.

L2062 **continuation of COMDAT** *function* **has conflicting attributes; record ignored**

This is a translator error.

Note which translator (compiler or assembler) produced the incorrect object module and the circumstances in which it was produced. Please report this error to Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

L2063 *function* **is allocated in undefined segment**

The given function was allocated to a nonexistent segment.

L2064 **starting address not in the root overlay**

The segment or object file that contains the starting address for the program was placed into an overlay.

The starting address in a C-language program is provided by the **main** function.

LINK Warning Messages

Number **LINK Warning Message**

L4000 **segment displacement included near** *offset* **in segment** *segment*

This is the warning generated by the /W option.

L4001 **frame-relative fixup, frame ignored near** *offset* **in segment** *segment*

A reference was made relative to a segment or group that is different from the target segment of the reference.

For example, if `_id1` is defined in segment `_TEXT`, the instruction call `DGROUP:_id1` produces this warning. The frame `DGROUP` is ignored, so LINK treats the call as if it were call `_TEXT:_id1`.

L4002 **frame-relative absolute fixup near** *offset* **in segment** *segment*

A reference was made relative to a segment or group that was different from the target segment of the reference, and both segments are absolute (defined with AT).

LINK assumed that the executable file will be run only under DOS.

L4004 **possible fixup overflow at** *offset* **in segment** *segment*

A near call or jump was made to another segment that was not a member of the same group as the segment from which the call or jump was made.

This can cause an incorrect real-mode address calculation when the distance between the paragraph address (frame number) of the segment group and the target segment is greater than 64K, even though the distance between the segment where the call or jump was actually made and the target segment is less than 64K.

L4010 **invalid alignment specification**

The number specified in the /ALIGN option must be a power of 2 in the range 2–32,768.

L4011 **/PACKC value exceeding 64K–36 unreliable**

The packing limit specified with the /PACKC option was in the range 65,501–65,536 bytes. Code segments with a size in this range are unreliable on some versions of the 80286 processor.

L4012 **/HIGH disables /EXEPACK**

The /HIGH and /EXEPACK options cannot be used at the same time.

L4013 *option* **: option ignored for segmented executable file**

The given option is not allowed for segmented executable programs.

L4014 *option* **: option ignored for DOS executable file**

The given option is not allowed for DOS programs.

L4015 **/CO disables /DSALLOC**

The /CO and /DSALLOC options cannot be used at the same time.

L4016 **/CO disables /EXEPACK**

The /CO and /EXEPACK options cannot be used at the same time.

L4017 *option* **: unrecognized option name; option ignored**

The given option was not a valid LINK option. LINK ignored the option specification.

One of the following may be a cause:

- An obsolete option was specified to the current version of LINK. For example, the /INCR option is obsolete in LINK version 5.30. The current options are described in the manual and in online Help. To see a list of options, run LINK with the /? option.
- An old version of LINK was used. Check your path. To see the version number of LINK, run LINK with the /? option.
- The name was incorrectly specified. For example, the option specification /NODEFAULTLIBSEARCH is an invalid abbreviation of the /NODEFAULTLIBRARYSEARCH option. Option names can be shortened by removing letters only from the end of the name.

L4018 **missing or unrecognized application type; option** *option* **ignored**

The /PM option accepts only the keywords **PM**, **VIO**, and **NOVIO**.

L4020 *segment* **: code-segment size exceeds 64K–36**

Code segments that are 65,501 through 65,536 bytes in length may be unreliable on some versions of the 80286 processor.

L4021 **no stack segment**

The program did not contain a stack segment defined with the STACK combine type.

Normally, every program should have a stack segment with the combine type specified as STACK. You can ignore this message if you have a specific reason for not defining a stack or for defining one without the STACK combine type. Linking with versions of LINK earlier than version 2.40 might cause this message since these linkers search libraries only once.

L4022 *group1*, *group2* **: groups overlap**

The given groups overlap. Since a group is assigned to a physical segment, groups cannot overlap in segmented executable files.

Reorganize segments and group definitions so the groups do not overlap. Refer to the map file.

L4023 *entry*(*internalname*) **: export internal name conflict**

The internal name of the given exported function or data item conflicted with the internal name of a previous import definition or export definition.

L4024 *name* **: multiple definitions for export name**

The given name was exported more than once, an action that is not allowed.

L4025 *modulename.entry*(*internalname*) **: import internal name conflict**

The internal name of the given imported function or data item conflicted with the internal name of a previous export or import. (The given entry is either a name or an ordinal number.)

L4026 *modulename.entry*(*internalname*) **: self-imported**

The given function or data item was imported from the module being linked. This error can occur if a module tries to import a function or data item from itself or from another source (such as a DLL) that has the same name.

L4027 *name* **: multiple definitions for import internal name**

The given internal name was imported more than once. Previous import definitions are ignored.

L4028 *segment* **: segment already defined**

The given segment was defined more than once in a **SEGMENTS** statement of the module-definition (.DEF) file.

L4029 *segment* **: DGROUP segment converted to type DATA**

The given logical segment in the group DGROUP was defined as a code segment.

DGROUP cannot contain code segments because LINK always considers DGROUP to be a data segment. The name DGROUP is predefined as the automatic (or default) data segment.

LINK converted the named segment to type DATA.

L4030 *segment* **: segment attributes changed to conform with automatic data segment**

The given logical segment in the group DGROUP was given sharing attributes (**SHARED/NONSHARED**) that differed from the automatic data attributes as declared by the DATA instance specification (**SINGLE/MULTIPLE**). The attributes are converted to conform to those of DGROUP.

The name DGROUP is predefined as the automatic (or default) data segment. DGROUP cannot contain code segments because LINK always considers DGROUP to be a data segment.

L4031 *segment* **: segment declared in more than one group**

A segment was declared to be a member of two different groups.

L4032 *segment* **: code-group size exceeds 64K–36**

The given code group has a size in the range 65,501–65,536 bytes, a size that is unreliable on some versions of the 80286 processor.

L4033 **first segment in mixed group** *group* **is a USE32 segment**

A 16-bit segment must be first in a group created with both USE16 and USE32 segments.

LINK continued to build the executable file, but the resulting file may not run correctly.

L4034 **more than 1024 overlay segments; extra put in root**

The limit on the number of segments that can go into overlays is 1024. Segments starting with the 1025th segment are assigned to the permanently resident portion of the program (the root).

L4036 **no automatic data segment**

The application did not define a group named DGROUP.

DGROUP has special meaning to LINK, which uses it to identify the automatic (or default) data segment used by the operating system. Most segmented executable applications require DGROUP.

This warning will not be issued if **DATA NONE** is declared or if the executable file is a dynamic-link library.

L4037 *group* **: both USE16 and USE32 segments in group; assuming USE32**

The given group was allocated contributions from both 16-bit segments and 32-bit segments.

L4038 **program has no starting address**

The segmented executable application had no starting address. A missing starting address will usually cause the program to fail.

High-level languages automatically specify a starting address. In a C-language program, this is provided by the **main** function.

If you are writing an assembly-language program, specify a starting address with the **END** statement.

DOS programs and dynamic-link libraries should never receive this message, regardless of whether they have starting addresses.

L4040 **stack size ignored for /TINY**

LINK ignores stack size if the /TINY option is used and if the stack segment has been defined in front of the code segment.

L4042 **cannot open old version**

The file specified in the **OLD** statement in the module-definition (.DEF) file could not be opened.

L4043 **old version not segmented executable format**

The file specified in the **OLD** statement in the module-definition (.DEF) file was not a valid segmented executable file.

L4045 **name of output file is** *filename*

LINK used the given filename for the output file.

If the output filename is specified without an extension, LINK assumes the default extension .EXE. Creating a Quick library, DLL, or .COM file forces LINK to use a different extension. In the following cases, if either .EXE or no extension is specified, LINK assumes the appropriate extension:

/TINY option	.COM
/Q option	.QLB
LIBRARY statement	.DLL

This warning also occurs if the name specified in the **LIBRARY** statement in the module-definition (.DEF) file does not match the name specified in the *exefile* field.

L4050 **file not suitable for /EXEPACK; relink without**

The size of the packed load image plus packing overhead was larger than it would be for the unpacked load image. There is no advantage to packing this program.

Remove /EXEPACK from the LINK command line. In PWB, turn off the Pack EXE File check box in the Additional Debug/Release Options dialog box under Link Options.

L4051 *filename* **: cannot find library**

LINK could not find the given library file.

One of the following may be a cause:

- The specified file does not exist. Enter the name or full path specification of a library file.
- The LIB environment variable is not set correctly. Check for incorrect directory specifications, mistyping, or a space, semicolon, or hidden character at the end of the line.
- An earlier version of LINK is being run. Check the path environment variable and delete or rename earlier linkers.

L4053 **VM.TMP : illegal filename; ignored**

VM.TMP appeared as an object-file name.

Rename the file and rerun LINK.

L4054 *filename* **: cannot find file**

LINK could not find the specified file.

Enter a new filename, a new path specification, or both.

L4055 **start address not equal to 0x100 for /TINY**

The starting address for a .COM file must be 100 hexadecimal.

Put the following line of assembly source code in front of the code segment:

```
ORG 100h
```

L4056 **/EXEPACK valid only for OS/2 and real-mode DOS; ignored**

The /EXEPACK option is incompatible with Windows programs.

L4057 **stack specified for DLL; ignored**

A stack was specified for a dynamic-link library (DLL). Either the /STACK option was used on the command line or the **STACKSIZE** statement was used in the module-definition (.DEF) file. LINK ignored the specification and did not create a stack.

A DLL does not have a stack.

L4058 **ignoring alias for already defined symbol** *symbol*

The specified symbol was redefined in the program. However, it is an identifier from a C run-time library that has an alias to a new name in OLDNAMES.LIB. LINK ignored the alias for the symbol.

This warning appears only when the /INFO option is specified.

L4067 **changing default resolution for weak external** *symbol*
from *oldresolution* **to** *newresolution*

LINK found conflicting default resolutions for a weak external. It ignored the first resolution and used the second.

L4068 **ignoring stack size greater than 64K**

A stack was defined with an invalid size. LINK assumed 64K.

L4069 **filename truncated to** *filename*

A filename specification exceeded the length allowed. LINK assumed the given filename.

L4070 **too many public symbols for sorting**

LINK uses the stack and all available memory in the near heap to sort public symbols for the /MAP option. This warning is issued if the number of public symbols exceeds the space available for them. In addition, the symbols are not sorted in the map file but are listed in an arbitrary order.

L4076 **no segments defined**

There was no code in the program.

This warning can occur if the file contains only resources.

L4077 **symbol** *function* **not defined; ordered allocation ignored**

The given function was specified in a **FUNCTIONS** statement in the module-definition (.DEF) file, but the function was not defined.

L4079 **symbol** *function* **already defined for ordered allocation; duplicate ignored**

The given function was specified twice in **FUNCTIONS** statements in the module-definition (.DEF) file.

L4080 **changing substitute name for alias** *symbol*
from *oldalias* **to** *newalias*

LINK found conflicting alias names. It ignored the first alias and used the second.

L4081 **cannot execute** *program arguments—message*

LINK could not run the given program (with the given arguments) for the given reason.

L4082 **changing overlay assigment for segment** *segment* **from** *oldnumber* **to** *newnumber*

The given segment was assigned to two overlays, represented by *oldnumber* and *newnumber*. LINK assumed the *newnumber* overlay.

Probably a command-line overlay specification with parentheses conflicted with an overlay specification in the module-definition (.DEF) file.

L4083 **changing overlay assigment for symbol** *symbol* **from** *oldnumber* **to** *newnumber*

The given symbol was assigned to two overlays, represented by *oldnumber* and *newnumber*. LINK assumed the *newnumber* overlay.

Probably a command-line overlay specification with parentheses conflicted with an overlay specification in the module-definition (.DEF) file.

L4084 *option* **: argument missing; option ignored**

The given option requires an argument, but none was specified.

For example, the following option specification causes this error:

```
/ONERROR
```

L4085 *option* **: argument invalid; assuming** *argument*

The given option was specified with a numeric argument that was out of range for the option. LINK assumed the given argument.

For example, the option specification /DYNAMIC:11000 causes the following error:

```
/DYNAMIC:11000 : argument invalid; assuming 10922
```

Floating-Point Math Error Messages

The error messages listed below correspond to exceptions generated by the math coprocessor hardware. Refer to the manufacturer's documentation for your processor for a detailed discussion of hardware exceptions. These errors may also be detected by the floating-point emulator or alternate math library.

Using the Microsoft C/C++ default math coprocessor control-word settings, the following exceptions are masked and do not occur:

Exception	Default Masked Action
Denormal	Exception masked
Underflow	Result goes to 0.0
Inexact	Exception masked

For information on how to change the floating-point control word, see the _control87 function in the *Run-Time Library Reference.*

The following exceptions do not occur with code generated by the Microsoft C/C++ Compiler or code provided in the standard Microsoft C/C++ run-time libraries or the MFC class libraries:

```
square root
stack underflow
unemulated
```

The floating-point error messages have the following format:

```
run-time error M6xxx : MATH
- floating-point error : messagetext
```

Number **Floating-Point Math Error Message**

M6101 **invalid**

An invalid operation occurred. This error usually occurs when the operand is NAN (not a number) or infinity.

This error terminates the program with exit code 129.

M6102 **denormal**

A very small floating-point number was generated, which may no longer be valid because of a loss of significance. Denormal floating-point exceptions are usually masked, causing them to be trapped and operated upon.

This error terminates the program with exit code 130.

M6103 **divide by 0**

A floating-point operation attempted to divide by zero.

This error terminates the program with exit code 131.

M6104 **overflow**

An overflow occurred in a floating-point operation.

This error terminates the program with exit code 132.

M6105 **underflow**

An underflow occurred in a floating-point operation. Underflow floating-point exceptions are usually masked, causing the underflowing value to be replaced by 0.0.

This error terminates the program with exit code 133.

M6106 **inexact**

Loss of precision occurred in a floating-point operation. This exception is usually masked. Many floating-point operations cause a loss of precision.

This error terminates the program with exit code 134.

M6107 **unemulated**

An attempt was made to execute a coprocessor instruction that is invalid or is not supported by the emulator.

This error terminates the program with exit code 135.

M6108 **square root**

The operand in a square-root operation was negative.

This error terminates the program with exit code 136.

Note The **sqrt** function in the C run-time library and the FORTRAN intrinsic function **SQRT** do not generate this error. The C **sqrt** function checks the argument before performing the operation and returns an error value if the operand is negative. The FORTRAN **SQRT** function generates the DOMAIN error M6201 instead of this error.

M6110 **stack overflow**

A floating-point expression caused a stack overflow on the 8087/80287/80387 coprocessor or the emulator.

Stack-overflow floating-point exceptions are trapped up to a limit of seven levels in addition to the eight levels usually supported by the 8087/80287/80387 coprocessor.

This error terminates the program with exit code 138.

M6111 **stack underflow**

A floating-point operation resulted in a stack underflow on the 8087/80287/80387 coprocessor or the emulator.

This error is often caused by a call to a **long double** function that does not return a value. For example, the following generates this error when compiled and run:

```
long double ld() {};

main ()
{
    ld();
}
```

This error terminates the program with exit code 139.

M6201 *function* **: _DOMAIN error**

An argument to the given function was outside the domain of legal input values for that function.

For example, the following statements generate this error:

```
result = sqrt(-1.0)   // C statement
result = SQRT(-1.0)   !  FORTRAN statement
```

This error calls the **_matherr** function with the function name, its arguments, and the error type. You can rewrite the **_matherr** function to customize the handling of certain run-time floating-point math errors.

M6202 *function* **: _SING error**

An argument to the given function was a singularity value for this function. The function is not defined for that argument.

For example, in FORTRAN the following statement generates this error:

```
result = LOG10(0.0)
```

This error calls the **_matherr** function with the function name, its arguments, and the error type. You can rewrite the **_matherr** function to customize the handling of certain run-time floating-point math errors.

M6203 *function* **: _OVERFLOW error**

The given function result was too large to be represented.

This error calls the **_matherr** function with the function name, its arguments, and the error type. You can rewrite the **_matherr** function to customize the handling of certain run-time floating-point math errors.

M6205 *function* **: _TLOSS error**

A total loss of significance (precision) occurred.

This error may be caused by giving a very large number as the operand of **sin**, **cos**, or **tan** because the operand must be reduced to a number between 0 and 2π.

MPC Error Messages

Microsoft Make P-Code Utility (MPC) generates the following error messages:

- Fatal errors (MP1*xxx*) cause MPC to stop execution.
- Errors (MP2*xxx*) do not stop execution but prevent MPC from creating an executable file.
- Warnings (MP4*xxx*) indicate possible problems in the p-code executable file being created.

MPC Fatal Error Messages

Number **MPC Fatal Error Message**

MP1001 **cannot open file :** *filename*

The given file either did not exist, could not be opened, or was not found.

One or more of the following can cause this error:

- The file was read-only or was being used by another process.
- An invalid filename or path was specified.
- MPC ran out of file handles. To increase the number of available file handles, change the FILES setting in CONFIG.SYS. FILES=50 is the recommended setting.
- The environment settings were invalid.
- There was a disk media error or an open floppy disk drive door.

MP1002 **file I/O error :** *filename*

There was an error when working with the given file.

One or more of the following can cause this error:

- There was a disk media error or an open floppy disk drive door.
- The file was read-only or was being used by another process.
- There was insufficient space on the specified drive.
- The file was corrupt.

MP1010 *filename* **is not a segmented executable file**

The given file was not a valid segmented executable file. MPC can only operate on segmented executable files.

Make sure that the given filename was compiled and linked with the correct command-line options to generate a segmented executable file.

P-code object modules contain special-purpose records that force the generation of a segmented executable file. When building an MS-DOS p-code program, MPC converts the segmented executable file to an unsegmented executable file.

MP1040 out of memory

The MPC program ran out of memory and was unable to recover.

One of the following may be a solution:

- Remove other programs or drivers running in the system, which could be consuming significant amounts of memory.
- Recompile some of the p-code functions as native code and relink.
- Run MPC directly from the command line, instead of from CL, LINK, NMAKE, or PWB.

MP1041 unsupported fixup; source type *type* **flags** *flags*

MPC has encountered an unsupported relocation fixup in the executable file.

This warning can be generated by dynamic-link library (DLL) fixups in an MS-DOS executable file. This warning can also be caused by using 32-bit code in a p-code executable file. P-code supports only 16-bit programs.

MP1042 too many p-code segments

The p-code limit of 255 segments was exceeded.

Each physical segment containing p-code functions or functions referenced by p-code routines is divided into one or more p-code logical segments. There is a limit of 255 p-code logical segments per executable file.

To reduce the number of physical p-code segments, select segments that reference or are referenced by many functions and compile them into native code by using **#pragma optimize("q", off)**. The number of p-code segments can also be reduced by combining physical segments.

MP1043 too many DLL functions referenced

The maximum number of dynamic-link library (DLL) functions referenced from p-code was exceeded. A maximum of 255 DLL entry points can be referenced from p-code in each executable file.

To eliminate this error, reduce the DLL reference count by recompiling some p-code functions that call DLL functions into native code.

MP1044 too many functions referenced by p-code function at *address*

The maximum number of unique functions referenced by a p-code function was exceeded. A maximum of 255 unique functions can be referenced by a p-code function.

To find the function that caused this error, look up the given map address in the .MAP file generated from the executable file. To generate a .MAP file, use the CL /Fm command-line option or the LINK /MAP command-line option.

To correct this error, either recompile the routine at the specified map address to native code or split the function into subfunctions that call fewer functions.

MP1045 **too many globals referenced by p-code function at** *address*

The maximum number of unique global variables referenced by a p-code function was exceeded. A maximum of 255 unique global variables can be referenced by a p-code function.

To find the function that caused this error, look up the given map address in the .MAP file generated from the executable file. To generate a .MAP file, use the CL /Fm command-line option or the LINK /MAP command-line option.

To correct this error, either recompile the routine at the specified map address to native code or split the function into subfunctions that reference fewer global variables.

MP1046 **unused entry** *number*

MPC has detected an unused entry in the module entry table. Unused entries are not currently supported by MPC.

MP1047 **entry table overflow; attempted to extend segment table to** *number*

The 64K limit of the segment table was exceeded.

MPC adds an entry to the segment table for each movable p-code segment and dynamic-link library (DLL) entry point.

To correct this situation, either reduce the number of p-code segments or mark some of the p-code segments as **FIXED**.

MP1048 **too many imported functions :** *number*

Too many imported functions were referenced by p-code functions. The limit is 4680 imported functions.

Reduce the number of imported functions referenced by p-code functions by recompiling p-code functions into native code or by moving references to imported function to native code.

MP1049 **invalid executable file :** *filename*

The given executable file contains errors. MPC will not process invalid executable files.

Relink the program so that no errors are generated.

MP1051 imported names table overflow

There were too many imported names in the executable file, causing the MPC buffer to overflow. The buffer is 64K bytes.

Reduce the number of imported names in the executable file.

MP1080 p-code interpreter not found : *filename*

MPC could not find the p-code interpreter in the given file.

Relink the executable file, making sure that no link errors occur. This error can also be caused by adding information to the reserved p-code interpreter segment.

MP1081 MPC fixup segment not found: *filename*

MPC could not find any p-code fixup segments in the given executable file.

The compiler emits special p-code fixup data to reserved data segments. These segments could not be found.

MP1082 invalid data found in executable file at *segment:offset*

MPC detected corrupted code or data in the executable file at the given address.

Recompile the source files containing p-code and relink the executable file. If the error persists, compile the function at the given address into native code.

MP1083 segment overflow writing p-code tables in segment *segment*

The writing of p-code tables at the end of the given segment caused the segment to exceed 64K. This segment contains or is referenced by p-code functions.

Reduce the size of the given segment and relink. If the LINK /PACKC option is being used, specify an amount to reserve for the p-code tables (512 bytes is the recommended amount.)

MP1084 nonsegmented executable file cannot contain imported routines

MPC attempted to generate a nonsegmented executable file, but encountered references to imported functions. This indicates a protected-mode executable file.

Relink the executable file using real-mode libraries.

MP1085 Windows application does not reference module KERNEL

When generating a real-mode Windows executable file, MPC requires a reference to the Windows KERNEL module in the executable file. This allows MPC to add the appropriate memory-management information to the p-code executable file.

This error is usually caused by a corrupted library.

Relink the executable file, making sure that no errors occur in the linking process.

MPC Error Messages

Number MPC Error Message

MP2081 missing native entry code for function at *segment:offset*

MPC detected a p-code function that was called from native code but does not have a valid native entry-code sequence.

To find the function that caused this error, look up the given map address in the .MAP file generated from the executable file. To generate a .MAP file, use the CL /Fm command-line option or the LINK /MAP command-line option.

This error can be caused by using **#pragma native_caller(off)** or the CL /Gn command-line option when defining functions that are exported.

Recompile and relink the module that contains the function at the specified map address, making sure that native entry code is generated, then relink the executable file.

MP2082 missing native entry code for exported function at *segment:offset*

MPC detected an exported p-code function that does not have a valid native entry code sequence. All exported p-code functions must have native entry code sequences.

To find the function that caused this error, look up the given map address in the .MAP file generated from the executable file. To generate a .MAP file, use the CL /Fm command-line option or the LINK /MAP command-line option.

This error can be caused by using **#pragma native_caller(off)** or the CL /Gn command-line option when defining functions that are exported.

Recompile and relink the module that contains the function at the specified map address, making sure that native entry code is generated, then relink the executable file.

MP2083 calling convention inconsistency: source *segment:offset*, **target** *segment:offset*

The calling convention of the function reference at the given source map address was inconsistent with the calling convention in the function defined at the target map address.

To find the functions that caused this error, look up the given map addresses in the .MAP file generated from the executable file. To generate a .MAP file, use the CL /Fm command-line option or the LINK /MAP command-line option.

Check the consistency of the function prototypes used in the given functions. Recompile the appropriate modules and relink.

MP2084 parameter length inconsistency: source *segment:offset*, **target** *segment:offset*

The length of the actual parameters of the function reference at the given source map address was inconsistent with the length of the formal parameters of the function defined at the target map address.

To find the functions that caused this error, look up the given map addresses in the .MAP file generated from the executable file. To generate a .MAP file, use the CL /Fm command-line option or the LINK /MAP command-line option.

Check the consistency of the function prototypes used in the given functions. Recompile the appropriate modules and relink.

MP2085 inconsistent reference made to the function at *segment:offset*

One or more inconsistent references were made to the function at the specified map address. Either the calling convention or parameter length was inconsistent.

To find the function that caused this error, look up the given map address in the .MAP file generated from the executable file. To generate a .MAP file, use the CL /Fm command-line option or the LINK /MAP command-line option.

Check the consistency of the function prototypes for the function at the given map address and all references to that function. Recompile the appropriate modules and relink.

MP2086 p-code version mismatch: compiler *version1*, **MPC** *version2*

The versions of the compiler and MPC that were used are incompatible. Make sure that compatible versions of the compiler and MPC are used when generating p-code programs.

This error can occur when there are multiple versions of the compiler or MPC in the path. Check the path to make sure that the correct versions are being used.

Either recompile the modules using the appropriate compiler or rerun the appropriate version of MPC.

MP2087 p-code version mismatch: p-code interpreter *version1*, **MPC** *version2*

The versions of PCD.LIB and MPC that were used are incompatible. Make sure that compatible versions of the p-code interpreter library and MPC are used when generating p-code programs.

This error can occur when there are multiple versions of the p-code interpreter or MPC in the path. Check the path to make sure that the correct versions are being used.

Either relink using the appropriate version of PCD.LIB or rerun the appropriate version of MPC.

MPC Warning Messages

Number MPC Warning Message

MP4001 interpreter segment not FIXED

The segment containing the p-code interpreter must be fixed when generating real-mode Windows executable files.

MPC considered the p-code interpreter segment to be **FIXED**.

The following lines in a LINK module-definition (.DEF) file will eliminate this warning:

```
SEGMENTS
    $$PCD_INTERP   FIXED
```

MP4002 executable file contains no p-code : *filename*

The given .EXE or .DLL file did not contain any p-code.

MPC copied the given file to the output filename without modification.

MP4003 file already processed by MPC : *filename*

The given .EXE or .DLL file has already been processed by MPC.

MPC copied the given file to the output filename without modification.

MP4004 far data reference to movable segment: source *segment:????*, **target** *segment:offset*

The source segment contained a far data reference to the target map address, which is in a movable segment. This is illegal when creating a real-mode Windows executable file.

These references are dangerous when running Windows in real mode because data segments can move, causing the references to be invalid.

To find the function that caused this warning, look up the given map address in the .MAP file generated from the executable file. To generate a .MAP file, use the CL /Fm command-line option or the LINK /MAP command-line option.

Relink the executable file using a module-definition (.DEF) file to mark the target segment as **FIXED** or to mark the executable file as **PROTMODE**.

MP4005 interpreter debug data segment not FIXED

The segment containing the p-code interpreter debug data must be **FIXED** when when creating a real-mode Windows executable file.

MPC considered the interpreter debug data segment to be **FIXED**.

MP4006 CVPACK has not been run on this executable file

The executable file contains Microsoft symbolic debugging information, but CVPACK has not been run.

MPC requires CVPACK-processed debugging information.

Run CVPACK on the executable file, then rerun MPC.

MP4007 MAP file parse failed at line *number* **in file** *filename*

MPC could not read the map (.MAP) file for this executable file.

The map file was ignored.

Either relink the executable file to generate a new map file or delete the current corrupt map file.

MP4008 internal optimization error; turn off quoting for *segment:offset*

A physical segment used more than 256 global variables and functions or contained more than 256 functions. This can cause problems when using p-code quoting optimization.

To find the function that caused this warning, look up the given map address in the .MAP file generated from the executable file. To generate a .MAP file, use the CL /Fm command-line option or the LINK /MAP command-line option.

To turn off the p-code quoting optimization, use **#pragma optimize("f", off)** or the /Of- command-line option.

PWB Error Messages

PWB displays an error message whenever it detects a command it cannot execute. Most errors terminate the command that is in error, but do not terminate PWB.

For most errors, PWB displays a message box with only the text of the message. The error number does not appear. With these messages, press F1 or click Help when the message box is displayed for Help on the error. Some errors terminate PWB. PWB displays these fatal errors on the command line after returning to the operating system.

This section lists only the fatal PWB errors.

Number PWB Error Message

PWB3089 Out of local memory. Unable to recover.

PWB has run out of memory and cannot recover. This is a fatal PWB condition. However, PWB is able to save your files, and you can restart PWB to continue.

This can happen when using PWB continuously for a long time.

This can also happen when creating a project with a very large number of files or adding files to a large project. To make the largest amount of memory available to PWB for creating a very large project, load only the PWBUTILS extension and only the language extensions you need for the project. Start PWB with the /DS option, and create the project before doing any other work.

If the project is too large for PWB to handle as a PWB project, you can use a non-PWB makefile for your project.

PWB3090 Out of virtual memory space. Unable to recover.

PWB has run out of virtual memory and cannot recover. This is a fatal PWB condition. However, PWB is able to save your files, and you can restart PWB to continue.

PWB3096 Unsupported video mode. Please change modes and restart.

A request was made to start PWB with the **Savescreen** switch set to yes (the default), but PWB does not support the current operating-system video mode.

Change the video mode and restart PWB.

PWB3178 Cannot start: unable to open swapping file

PWB is unable to create its virtual-memory file on disk.

PWB creates this file in the directory pointed to by the TMP environment variable. If no TMP environment variable is set, PWB creates the file in the current directory.

Check that the disk has at least 2 free megabytes and that the directory can be accessed with permissions to create a file. Check that the TMP environment variable lists a single existing directory.

PWB3180 Cannot start: not enough far memory

PWB ran out of memory while starting up.

Make more memory available to PWB and restart PWB.

PWB3181 Cannot initialize

PWB cannot initialize itself.

Check that there is enough memory available for PWB. Also, check that there is no conflict with a TSR (terminate-and-stay-resident) program.

PWB3901 RE: error *number*, line *line*

PWB has encountered an error while processing a regular expression. The expression may be malformed or too complex.

Check that the syntax of the regular expression is correct.

PWB3909 RemoveFile can't find file

PWB has encountered an internal error.

Contact Microsoft Product Support Services.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

PWB3912 Internal VM Error

PWB has encountered an internal error.

Contact Microsoft Product Support Services.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

PWB12078 Cannot access *file*: *reason*

PWB cannot access the given file for the stated reason.

Correct the situation and restart PWB.

PWB12086 Cannot access TMP directory: *reason*

PWB cannot access the directory listed in the TMP environment variable for the stated reason.

Correct the situation and restart PWB.

Run-Time Error Messages

The following messages indicate general problems that may occur during program startup, termination, or execution. These error messages have the following format:

```
run-time error R6xxx
- messagetext
```

Number Run-Time Error Message

R6000 stack overflow

The program ran out of stack space. This can occur when a program uses a large amount of local data or is heavily recursive.

The use of p-code can cause a program to require more stack space than it would with native code.

There are several ways to allocate a larger stack:

- Recompile using the /F compiler option.
- Relink using LINK's /STACK option.
- Run EXEHDR on the program using the /STACK option.

R6001 **null pointer assignment**

The contents of the NULL segment changed during the course of program execution. The program wrote to this area, possibly due to an inadvertent assignment through a null pointer.

The NULL segment is a location in low memory that is not normally used. The contents of the NULL segment are checked upon program termination. If a change is detected, this error is generated.

This error appears only when the program writes to memory through a null pointer; the run-time library does not check reads from null pointers.

Although a program that produces this run-time error may appear to operate correctly, references to null pointers may cause problems when executed under a different operating environment.

R6002 **floating-point support not loaded**

The program needs the floating-point library, but the library was not linked to the program.

One of the following may have occurred:

- The program was compiled or linked with an option (such as /FPi87) that required a coprocessor, but the program was run on a machine that did not have a coprocessor installed.
- A format string for a **printf** or **scanf** function contained a floating-point format specification, and the program did not contain any floating-point values or variables.

  The compiler minimizes a program's size by loading floating-point support only when necessary. The compiler cannot detect floating-point format specifications in format strings, so it does not load the necessary floating-point routines.

  Use a floating-point argument to correspond to the floating-point format specification, or perform a floating-point assignment elsewhere in the program. This causes floating-point support to be loaded.
- In a mixed-language program, a C library was specified before a FORTRAN library when the program was linked. Relink and specify the C library last.

R6003 **integer divide by 0**

An attempt was made to divide an integer by zero, which produces an undefined result.

R6005 **not enough memory on exec**

Not enough memory was available to load the process being spawned.

This error occurs when a child process that was spawned by one of the **exec** library routines fails and the operating system cannot return control to the parent process.

R6006 **invalid format on exec**

The file to be executed by one of the **exec** functions was not in the correct format for an executable file.

This error occurs when a child process that was spawned by one of the **exec** library routines fails and the operating system cannot return control to the parent process.

R6007 **invalid environment on exec**

During a call to an **exec** function, the operating system found that the child process was given an invalid environment block.

This error occurs when a child process that was spawned by one of the **exec** library routines fails and the operating system cannot return control to the parent process.

R6008 **not enough space for arguments**

There was enough memory to load the program but not enough memory to create the *argv* array.

One of the following may be a solution:

- Increase the amount of memory available to the program.
- Reduce the number and size of command-line arguments.
- Reduce the environment size by removing unnecessary variables.

R6009 **not enough space for environment**

There was enough memory to load the program but not enough memory to create the *envp* array.

One of the following may be a solution:

- Increase the amount of memory available to the program.
- Reduce the number and size of command-line arguments.
- Reduce the environment size by removing unnecessary variables.

If your program uses the compact, large, or huge memory model, this error may be avoided by using LINK's /CPARM:1 command-line option. This option causes unused near heap space to be allocated to the far heap.

R6010 **abnormal program termination**

The **abort** function was called.

This error is generated by the **abort** function. The program terminates with exit code 3, unless an abort signal handler has been defined by using the **signal** function.

R6012 **illegal near-pointer use**

A null near pointer was used in the program.

This error only occurs if pointer checking is in effect. You can enable pointer checking with either the /Zr compiler option or the **check_pointer** pragma.

R6013 **illegal far-pointer use**

An out-of-range far pointer was used in the program.

This error only occurs if pointer checking is in effect. You can enable pointer checking with either the /Zr compiler option or the **check_pointer** pragma.

R6016 **not enough space for thread data**

The program did not receive enough memory from the operating system to complete a **_beginthread** call.

When a new thread is started, the library must create an internal database for the thread. If the database cannot be expanded with memory provided by the operating system, the thread will not begin and the calling process will stop.

R6017 **unexpected multithread lock error**

The process received an unexpected error while trying to access a C run-time multithread lock on a system resource.

This error usually occurs if the process inadvertently alters the run-time heap data. However, it can also be caused by an internal error in the run-time or operating-system code.

R6018 **unexpected heap error**

The program encountered an unexpected error while performing a memory-management operation.

This error usually occurs if the program inadvertently alters the run-time heap data. However, it can also be caused by an internal error in the run-time or operating-system code.

If your compiler provides a library containing **_heapchk** and **_heapwalk**, you can use these functions to diagnose this error.

R6019 **unable to open console device**

The program called a console function declared in CONIO.H, but the operating system did not grant access to the console.

R6020 **unexpected QuickWin error**

The program encountered an unexpected QuickWin error.

One of the following may have occurred:

- The program tried to access QuickWin, but the program was built without QuickWin libraries.
- A QuickWin operation had an unrecoverable error.

R6021 **no main procedure**

The program does not have a procedure called **main**.

Make sure that all object and library modules have been linked into the executable.

R6800 **internal error**

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

R6801 **overlay manager stack overflow**

The program exceeded the limit on recursive overlay calls.

Overlay calls can be nested up to 64 levels.

R6802 **requires DOS 3.0 or higher**

A program that uses the MOVE overlay manager can run only under DOS version 3.0 or higher.

R6803 **DOS memory error**

MOVE was unable to allocate memory from DOS.

Probably the available memory was corrupted.

R6804 **not enough conventional memory**

There was insufficient conventional memory (the lower 640K of memory) to load the program. The required memory equals the size of the root plus the largest overlay plus MOVE's overhead.

R6805 **cannot open file**

MOVE could not find the program's executable file, which it needs when loading a new overlay.

R6806 **cannot read file**

An error occurred while MOVE was reading the program's executable file to load a new overlay.

R6807 **invalid executable file**

The program's executable file was not in the format recognized by the operating system.

R6808 **error accessing expanded memory**

MOVE encountered an error while writing to or reading from expanded memory. The error could be a MOVE problem or a problem with the expanded memory manager (EMM).

R6809 **error accessing extended memory**

MOVE encountered an error while writing to or reading from extended memory. The error could be a MOVE problem or a problem with the extended memory manager.

R6810 **overlay manager was reentered**

The program reentered the overlay manager, which is not reentrant.

Possibly a signal handler was called in an overlay or the handler called a routine that was in an overlay.

R6900 **internal error**

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

R6901 **This is a protected-mode application and requires a DPMI host.**

DPMI (DOS Protected Mode Interface) is a standard that provides services required by protected-mode programs. Examples of DPMI hosts include:

- A DOS session under Windows (386 enhanced mode)
- MSDPMI.EXE, provided in Microsoft C/C++ version 7.0

R6902 **DPMI host not 32-bit**

The installed DPMI host is a 16-bit server.

R6903 **DPMI host does not have 32-bit interrupts**

The installed DPMI host does not support 32-bit interrupt services. The DOS Extender requires 32-bit interrupt services, which are an extension to the DPMI specification.

Servers that provide these services include:

- A DOS session under Windows
- MSDPMI.EXE, provided in Microsoft C/C++ version 7.0

R6904 **requires DOS 3.0 or higher**

The program uses the 32-bit DOS Extender and must be run under DOS version 3.0 or higher.

R6905 **requires 80386 or higher**

The program uses the 32-bit DOS Extender and must be run on a machine that has an 80386 processor or higher version.

R6906 **not enough conventional memory on exec**

There was insufficient conventional memory (the lower 640K of memory) to load the program.

Remove TSR (terminate-and-stay-resident) programs. Under MS-DOS version 5.0, you can load TSRs using either a LOADHIGH command in AUTOEXEC.BAT or a DEVICEHIGH command in CONFIG.SYS.

R6907 **not enough extended memory on exec [: server]**

There was insufficient expanded or extended memory to load the program.

Use the DOS CHKDSK command to see the current amount of extended or expanded memory.

The error message may be followed by the type of server that is providing the memory, which can be one of the following:

- If the server is VCPI (for example, Microsoft's EMM386.EXE), it is not configured to provide enough expanded memory.
- If the server is DPMI (for example, a DOS session under Windows, or MSDPMI.EXE), there is not enough extended or expanded memory.
- If the server is XMS (for example, Microsoft's HIMEM.SYS), it is not configured to provide enough extended memory.

R6908 **cannot find file :** *filename*

The DOS Extender couldn't find the given file.

If the given filename is MS32KRNL.DLL or MS32EM87.DLL, set the SYSTEM environment variable to the directory that contains this file.

R6909 **cannot read file :** *filename*

The DOS Extender couldn't open the given file.

Probably the file is locked by another process.

R6910 **invalid executable file [:** *filename***]**

The file to be executed was not in the correct format for an executable file. The filename may be displayed following the error message.

Possibly the file was corrupted or overwritten.

R6911 **invalid environment on exec**

The DOS Extender found a problem in the DOS environment.

One of the following may have occurred:

- An invalid environment block was passed to a spawned DOS-extended program.
- The DOS environment was corrupted.

R6912 **out of selectors**

The program requested more segments than the existing number of selectors.

R6913 **incompatible file version :** *filename*

The version of the given file does not match the versions of related files. All product files must be updated simultaneously. The required files may include the following:

- CL.EXE
- MS32EM87.DLL
- MS32KRNL.DLL
- VMCPD.386
- VPFD.386

R6914 **cannot initialize device :** *filename*

The DOS Extender could not find or initialize the given device driver.

The initialization file must contain commands to load the device driver. For Windows, this file is SYSTEM.INI; for MSDPMI, the file is MSDPMI.INI. The commands must be in a section marked with the label [386Enh]. The required commands are as follows:

- For MSDPMI and for Windows 3.0, use:

  ```
  device=[path\]vmcpd.386
  device=[path\]vpfd.386
  ```

  If the command `device=*vmcpd` appears, either delete it or change it to a comment by preceding it with a semicolon (;).

- For later versions of Windows, use:

  ```
  device=*vmcpd
  device=[path\]vpfd.386
  ```

R6915 **unhandled exception [:** *number***]**

The DOS-extended program caused a protected-mode fault. Probably an invalid memory access operation occurred. The message may be followed by a number that represents an exception code defined by the operating system.

If the program is a Microsoft product, note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

R6916 **requires 80286 or higher**

The program uses the 16-bit DOS Extender and must be run on a machine that has an 80286 processor or higher version.

R6917 **CPU already in protected or virtual mode**

The DOS Extender cannot enter protected mode because an operating system was already running and had entered protected mode.

R6918 **DPMI, VCPI, or XMS host required**

The program is DOS-extended and requires extended or expanded memory, which must be provided by one of the following:

- A DPMI server, such as MSDPMI.EXE or the server provided in a DOS session under Windows enhanced mode. This server provides extended or expanded memory.

- A VCPI server, such as Microsoft's EMM386.EXE. This server provides expanded memory.
- An XMS server, such as Microsoft's HIMEM.SYS. This server provides extended memory.

R6919 **unexpected initialization error**

An unknown error occurred that prevented loading the program.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

R6920 **invalid XMS host**

The XMS server cannot provide any extended memory. This error occurs in the following environments:

- Windows 3.0 standard mode
- MS-DOS Shell in MS-DOS version 5.0

R6921 **no expanded memory under VCPI host**

The VCPI driver was loaded but no expanded memory was allocated.

Possibly the CONFIG.SYS file contained a line such as:

```
DEVICE=C:DOSEMM386.EXE NOEMS
```

In this case, replace **NOEMS** with another argument. Specify a decimal number in kilobytes. You can specify **RAM** (or leave the argument blank) to allocate 256K (the default). If error R6907 subsequently occurs, increase the specified value.

It is recommended that you run this program under a DPMI host such as MSDPMI.EXE or the DOS box in Windows enhanced mode.

SBRPACK Error Messages

This section lists error messages generated by the Microsoft Browse Information Compactor (SBRPACK). SBRPACK errors (SB *xxx*) are always fatal.

Number **SBRPACK Fatal Error Message**

SB1000 **UNKNOWN ERROR**
Contact Microsoft Product Support Services

SBRPACK detected an unknown error condition.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

This error ends SBRPACK with exit code 1.

SB1001 *option* **: unknown option**

SBRPACK did not recognize the given option.

This error ends SBRPACK with exit code 1.

SB1002 *sbrfile* **: corrupt file**

The given .SBR file is corrupt or does not have the expected format.

Recompile to regenerate the .SBR file.

This error ends SBRPACK with exit code 2.

SB1003 *sbrfile* **: invalid .SBR file**

SBRPACK did not recognize the given file as an .SBR file.

One of the following may be a solution:

- Check the spelling of the specified file.
- Recompile to regenerate the .SBR file.

This error ends SBRPACK with exit code 2.

SB1004 *sbrfile* **: incompatible .SBR version**

The given .SBR file cannot be packed by this version of SBRPACK.

One of the following may be a cause:

- The .SBR file was created by a compiler that is not compatible with this version of SBRPACK.
- The .SBR file is corrupt.

This error ends SBRPACK with exit code 2.

SB1005 *sbrfile* **: cannot open file**

SBRPACK cannot open the given .SBR file.

One of the following may be a cause:

- The .SBR file does not exist. Check the spelling.
- The .SBR file was locked by another process.

This error ends SBRPACK with exit code 3.

SB1006 cannot create temporary .SBR file

SBRPACK could not open a temporary file.

One of the following may have occurred:

- No more file handles were available. Increase the number of file handles by changing the FILES setting in CONFIG.SYS to allow a larger number of open files. FILES=50 is recommended.
- The disk was full.

This error ends SBRPACK with exit code 4.

NMAKE, EXEHDR, and LIB Error Messages

This section lists error messages generated by the NMAKE, EXEHDR, and LIB utilities.

NMAKE Error Messages

Microsoft Program Maintenance Utility (NMAKE) generates the following error messages:

- Fatal errors (U1000 through U1099) cause NMAKE to stop execution.
- Errors (U2001) do not stop execution but prevent NMAKE from completing the make process.
- Warnings (U4001 through U4011) indicate possible problems in the make process.

EXEHDR Error Messages

This section includes error messages generated by the Microsoft EXE File Header Utility (EXEHDR). EXEHDR errors (U1100 through U1140) are always fatal.

LIB Error Messages

Microsoft Library Manager (LIB) generates the following error messages:

- Fatal errors (U1150 through U1203) cause LIB to stop execution.
- Errors (U2152 through U2159) do not stop execution but prevent LIB from creating a library.
- Warnings (U4150 through U4158) indicate possible problems in the library being created.

NMAKE, EXEHDR, and LIB Fatal Error Messages

Number	NMAKE, EXEHDR, and LIB Fatal Error Message

U1000 **syntax error : ')' missing in macro invocation**

A left parenthesis, (, appeared without a matching right parenthesis,), in a macro invocation. The correct form is $(*name*), and $*n* is allowed for one-character names.

U1001 **syntax error : illegal character** *character* **in macro**

The given character appeared in a macro but was not a letter, number, or underscore (_).

If the colon (:) is omitted in a macro expansion, the following error occurs:

```
syntax error : illegal character '=' in macro
```

U1002 **syntax error : invalid macro invocation '$'**

A single dollar sign ($) appeared without a macro name associated with it.

The correct form is $(*name*). To specify a dollar sign, use a double dollar sign ($$) or precede it with a caret (^).

U1003 **syntax error : '=' missing in macro substitution**

A macro invocation contained a colon (:), which begins a substitution, but it did not contain an equal sign (=).

The correct form is:

```
$(macroname:oldstring=newstring)
```

U1004 **syntax error : macro name missing**

One of the following occurred:

- The name of a macro being defined was itself a macro invocation that expanded to nothing. For example, if the macro named `ONE` is undefined or has a null value, the following macro definition causes this error:

```
$(ONE)=TWO
```

- A macro invocation did not specify a name in the parentheses. The following specification causes this error:

```
$()
```

The correct form is:

```
$(name)
```

U1005 **syntax error : text must follow ':' in macro**

A string substitution was specified for a macro, but the string to be changed in the macro was not specified.

U1006 **syntax error : missing closing double quotation mark**

An opening double quotation mark (") appeared without a closing double quotation mark.

U1007 **double quotation mark not allowed in name**

The specified target name or filename contained a double quotation mark (").

Double quotation marks can surround a filename but cannot be contained within it.

U1017 **unknown directive** *!directive*

The specified directive is not one of the recognized directives.

U1018 **directive and/or expression part missing**

The directive was incompletely specified.

The expression part of the directive is required.

U1019 **too many nested !IF blocks**

The limit on nesting of **!IF** directives was exceeded.

The **!IF** preprocessing directives include **!IF**, **!IFDEF**, **!IFNDEF**, **!ELSE IF**, **!ELSE IFDEF**, and **!ELSE IFNDEF**.

U1020 **end-of-file found before next directive**

An expected directive was missing.

For example, an **!IF** was not followed by an **!ENDIF**.

U1021 **syntax error : !ELSE unexpected**

An **!ELSE** directive was found that was not preceded by an **!IF** directive, or the directive was placed in a syntactically incorrect place.

The **!IF** preprocessing directives include **!IF**, **!IFDEF**, **!IFNDEF**, **!ELSE IF**, **!ELSE IFDEF**, and **!ELSE IFNDEF**.

U1022 **missing terminating character for string/program invocation :** *char*

The closing double quotation mark (") in a string comparison in a directive was missing, or the closing bracket (]) in a program invocation in a directive was missing.

U1023 **syntax error in expression**

An expression was invalid.

Check the allowed operators and operator precedence.

U1024 **illegal argument to !CMDSWITCHES**

An unrecognized command switch was specified.

U1031 **filename missing (or macro is null)**

An **!INCLUDE** directive was found, but the name of the file to be included was missing or a macro representing the filename expanded to nothing.

U1033 **syntax error :** *string* **unexpected**

The given string is not part of the valid syntax for a makefile.

The following are examples of causes and results of this error:

- If the closing set of angle brackets for an inline file are not at the beginning of a line, the following error occurs:

```
syntax error : 'EOF' unexpected
```

- If a macro definition in the makefile contained an equal sign (=) without a preceding name or if the name being defined is a macro that expands to nothing, the following error occurs:

```
syntax error : '=' unexpected
```

- If the semicolon (;) in a comment line in TOOLS.INI is not at the beginning of the line, the following error occurs:

```
syntax error : ';' unexpected
```

- If the makefile has been formatted by a word processor, the following error can occur:

```
syntax error : ':' unexpected
```

U1034 **syntax error : separator missing**

The colon (:) that separates targets and dependents is missing.

U1035 **syntax error : expected ':' or '=' separator**

Either a colon (:) or an equal sign (=) was expected.

Possible causes include the following:

- A target was not followed by a colon.
- A single-letter target was followed by a colon and no space (such as `a:`). NMAKE interpreted it as a drive specification.
- An inference rule was not followed by a colon.
- A macro definition was not followed by an equal sign.
- A character followed a backslash (\) that was used to continue a command to a new line.
- A string appeared that did not follow any NMAKE syntax rule.
- The makefile was formatted by a word processor.

U1036 **syntax error : too many names to left of '='**

Only one string is allowed to the left of a macro definition.

U1037 **syntax error : target name missing**

A colon (:) was found before a target name was found.

At least one target is required.

U1038 **internal error : lexer**

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

U1039 **internal error : parser**

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

U1040 **internal error : macro expansion**

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

U1041 **internal error : target building**

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

U1042 **internal error : expression stack overflow**

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

U1043 **internal error : temp file limit exceeded**

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

U1045 **spawn failed :** *message*

A program or command, called by NMAKE, failed for the given reason.

U1047 **argument before ’)’ expands to nothing**

The parentheses following the preprocessing operator **DEFINED** or **EXIST** either were empty or contained an argument that evaluated to a null string.

U1048 **cannot write to file** *filename*

NMAKE could not write to the given file.

One cause of this error is a read-only file specified with /X.

U1049 **macro or inline file too long (maximum : 64K)**

An inline file or a macro exceeded the limit of 64K.

U1050 *user-specified text*

The message specified with the **!ERROR** directive was displayed.

U1051 **out of memory**

The makefile was too large or complex for available memory.

U1052 **file** *filename* **not found**

NMAKE could not find the given file, which was specified with one of the following:

- The /F option
- The **!INCLUDE** preprocessing directive
- The at sign (@) specifier for a response file

Check that the file exists and the filename is spelled correctly.

U1053 **file** *filename* **unreadable**

The file cannot be read.

One of the following may be a cause:

- The file is in use by another process.
- A bad area exists on disk.
- A bad file-allocation table exists.

U1054 **cannot create inline file** *filename*

NMAKE failed to create the given inline file.

One of the following may be a cause:

- A file by that name exists with a read-only attribute.
- The disk is full.

U1055 **out of environment space**

The operating system ran out of room for environment variables.

Either increase the environment space or set fewer environment variables.

U1056 **cannot find command processor**

The command processor was not in the path specified in the COMSPEC or PATH environment variables.

NMAKE uses COMMAND.COM or CMD.EXE as a command processor when executing commands. It looks for the command processor first in the path set in COMSPEC. If COMSPEC does not exist, NMAKE searches the directories specified in PATH.

U1057 **cannot delete temporary file** *filename*

NMAKE failed to delete the temporary inline file.

U1058 **terminated by user**

NMAKE was halted by CTRL+C or CTRL+BREAK.

U1060 **unable to close file :** *filename*

NMAKE encountered an error while closing a file.

One of the following may be a cause:

- The file is a read-only file.
- There is a locking or sharing violation.
- The disk is full.

U1061 /F option requires a filename

The /F command-line option must be followed by either a makefile name or a dash (–), which represents standard input.

U1062 missing filename with /X option

The /X command-line option requires the name of the file to which diagnostic error output should be redirected.

To use standard output, specify '-' as the output filename.

U1063 missing macro name before '='

A macro definition on the NMAKE command line contained an equal sign (=) without a preceding name.

This error can occur if the macro name being defined is itself a macro that expands to nothing.

U1064 MAKEFILE not found and no target specified

The NMAKE command line did not specify a makefile or a target, and the current directory did not contain a file named MAKEFILE.

NMAKE requires either a makefile or a command-line target. To make a makefile available to NMAKE, either specify the /F option or place a file named MAKEFILE in the current directory. NMAKE can create a command-line target by using an inference rule if a makefile is not provided.

U1065 invalid option *option*

The specified option is not a valid option for NMAKE.

U1069 no match found for wildcard *filename*

There is no file that matches the given filename, which was specified using one or more wildcards (* and **?**).

A target file specified using a wildcard must exist on disk.

U1070 cycle in macro definition *macroname*

The given macro definition contained a macro whose definition contained the given macro. Circular macro definitions are invalid.

For example, the following macro definitions:

```
ONE=$(TWO)
TWO=$(ONE)
```

cause the following error:

```
cycle in macro definition 'TWO'
```

U1071 **cycle in dependency tree for target** *targetname*

A circular dependency exists in the dependency tree for the given target. The given target is a dependent of one of the dependents of the given target. Circular dependencies are invalid.

U1072 **cycle in include files :** *filename*

The given file includes a file that eventually includes the given file. Inclusions (using the **!INCLUDE** preprocessing directive) cannot be circular.

U1073 **don't know how to make** *targetname*

The specified target does not exist, and there is no command to execute or inference rule to apply.

One of the following may be a solution:

- Check the spelling of the target name.
- If *targetname* is a pseudotarget, specify it as a target in another description block.
- If *targetname* is a macro invocation, be sure it does not expand to a null string.

U1076 **name too long**

A string exceeded one of the following limits:

- A macro name cannot exceed 1024 characters.
- A target name (including path) cannot exceed 256 characters.
- A command cannot exceed 2048 characters.

U1077 *program* **: return code** *value*

The given command or program called by NMAKE failed and returned the given exit code.

To suppress this error and continue the NMAKE session, use the /I option, the **.IGNORE** dot directive, or the dash (–) command modifier. To continue the NMAKE session for unrelated parts of the dependency tree, use the /K option.

U1078 **constant overflow at** *expression*

The given expression contained a constant that exceeded the range –2,147,483,648 to 2,147,483,647. The constant appeared in one of the following situations:

- An expression specified with a preprocessing directive
- An error level specified with the dash (–) command modifier

U1079 **illegal expression : divide by zero**

An expression tried to divide by zero.

U1080 **operator and/or operand usage illegal**

The expression incorrectly used an operator or operand.

Check the allowed set of operators and their order of precedence.

U1081 *filename* **: program not found**

NMAKE could not find the given program in order to run it.

Make sure that the program is in a directory specified in the PATH environment variable and is not misspelled.

U1082 *command* **: cannot execute command; out of memory**

There is not enough memory to execute the given command.

U1083 **target macro** *target* **expands to nothing**

The given target is an invocation of a macro that has not been defined or has a null value. NMAKE cannot process a null target.

U1084 **cannot create temporary file** *filename*

NMAKE was unable to create the temporary file it needs when it processes the makefile.

One of the following may be a cause:

- The file already exists with a read-only attribute.
- There is insufficient disk space to create the file.
- The directory specified in the TMP environment variable does not exist.

U1085 **cannot mix implicit and explicit rules**

A target and a pair of inference-rule extensions were specified on the same line. Targets cannot be named in inference rules.

U1086 **inference rule cannot have dependents**

The colon (**:**) in an inference rule must be followed by one of the following:

- A newline character
- A semicolon (**;**), which can be followed by a command
- A number sign (**#**), which can be followed by a comment

U1087 **cannot have : and :: dependents for same target**

A target cannot be specified in both a single-colon (**:**) and a double-colon (**::**) dependency.

To specify a target in multiple description blocks, use **::** in each dependency line.

U1088 **invalid separator '::' on inference rule**

An inference rule must be followed by a single colon (**:**).

U1089 **cannot have build commands for directive** *targetname*

Dot directives cannot be followed by commands. The dot directives are **.IGNORE**, **.PRECIOUS**, **.SILENT**, and **.SUFFIXES**.

U1090 **cannot have dependents for directive** *targetname*

Dot directives cannot be followed by dependents. The dot directives are **.IGNORE**, **.PRECIOUS**, **.SILENT**, and **.SUFFIXES**.

U1092 **too many names in rule**

An inference rule cannot specify more than two extensions.

U1093 **cannot mix dot directives**

Multiple dot directives cannot be specified on one line. The dot directives are **.IGNORE**, **.PRECIOUS**, **.SILENT**, and **.SUFFIXES**.

U1094 **syntax error : only (NO)KEEP allowed here**

Something other than **KEEP** or **NOKEEP** appeared after the closing set of angle brackets (<<) specifying an inline file. Only **KEEP**, **NOKEEP**, or a newline character may follow the angle brackets. No spaces, tabs, or other characters may appear.

KEEP preserves the inline file on disk. **NOKEEP** deletes the file after the NMAKE session. The default is **NOKEEP**.

U1095 **expanded command line** *commandline* **too long**

After macro expansion, the given command line exceeded the limit on length of command lines for the operating system.

DOS permits up to 128 characters on a command line.

If the command is for a program that can accept command-line input from a file, change the command and supply input from either a file on disk or an inline file. For example, LINK and LIB accept input from a response file.

U1096 **cannot open inline file** *filename*

NMAKE could not create the given inline file.

One of the following occurred:

- The disk was full.
- A file with that name exists as a read-only file.

U1097 **filename-parts syntax requires dependent**

The current dependency does not have either an explicit dependent or an implicit dependent. Filename-parts syntax, which uses the percent (%) specifier, represents components of the first dependent of the current target.

U1098 **illegal filename-parts syntax in** *string*

The given string does not contain valid filename-parts syntax.

U1099 **stack overflow**

The makefile being processed was too complex for the current stack allocation in NMAKE. NMAKE has an allocation of 0x3000 (12K).

To increase NMAKE's stack allocation, run the EXEHDR utility with a larger stack option:

```
EXEHDR /STACK:stacksize
```

where `stacksize` is a number greater than the current stack allocation in NMAKE.

U1100 **invalid magic number** *number*

EXEHDR discovered an unknown signature in the header for the file.

The signature in the header for a file identifies the operating system under which the executable file will run.

U1101 **automatic data segment greater than 64K;
correcting heap size**

There was not enough space in the automatic, or default, data segment (DGROUP) to accommodate the requested new heap size. EXEHDR adjusted the heap size to the maximum available space.

This error applies only to segmented executable files.

U1102 **automatic data segment greater than 64K;
correcting stack size**

There was not enough space in the automatic, or default, data segment (DGROUP) to accommodate the requested new stack size. EXEHDR adjusted the stack size to the maximum available space.

This error applies only to segmented executable files.

U1103 **invalid .EXE file : actual length less than reported**

The second and third fields in the input-file header indicate a file size greater than the actual size.

U1104 **cannot change load-high program**

When the minimum allocation and the maximum allocation are both 0, the file cannot be modified.

U1105 **minimum allocation less than stack; correcting minimum**

If the minimum allocation is not enough to accommodate the stack (either as originally requested or as modified with EXEHDR), the minimum allocation is adjusted.

This error applies only to DOS programs.

U1106 **minimum allocation greater than maximum; correcting maximum**

If the minimum allocation is greater than the maximum allocation, the maximum allocation value is adjusted.

If a display of DOS header values is requested, the values shown will be the values after the packed file is expanded.

This error applies only to DOS programs.

U1107 **unexpected end of resident/nonresident name table**

While decoding run-time relocation records, EXEHDR found the end of either the resident names table or the nonresident names table. The executable file is probably corrupted.

This error applies only to segmented executable files.

U1108 **unknown format of relocation records**

EXEHDR cannot decode the information in the file header because the header is not in a standard format.

U1109 **illegal value** *argument*

The given argument was invalid for the EXEHDR option it was specified with.

U1110 **malformed number** *number*

A command-line option for EXEHDR required a value, but the given number was mistyped.

U1111 **option requires value**

A command-line option for EXEHDR required a value, but no value was specified or the specified value was in an illegal format for the given option.

U1112 **value out of legal range** *lower – upper*

A command-line option for EXEHDR required a value, but the specified number did not fall in the required decimal range.

U1113 **value out of legal range** *lower – upper*

A command-line option for EXEHDR required a value, but the specified number did not fall in the required hexadecimal range.

U1114 **missing option value; option** *option* **ignored**

The given command-line option for EXEHDR required a value, but nothing was specified. EXEHDR ignored the option.

U1115 **option** *option* **ignored**

The given command-line option for EXEHDR was ignored. This error usually occurs with error U1116, unrecognized option.

U1116 **unrecognized option:** *option*

A command-line option for EXEHDR was not recognized. This error usually occurs with either U1115, option ignored, or U1111, option requires value.

U1120 **input file missing**

No input file was specified on the EXEHDR command line.

U1121 **command line too long:** *commandline*

The given EXEHDR command line exceeded the limit of 512 characters.

U1130 **cannot read** *filename*

EXEHDR could not read the input file. Either the file is missing or the file attribute is set to prevent reading.

U1131 **invalid .EXE file**

The input file specified on the EXEHDR command line was not recognized as an executable file.

U1132 **unexpected end-of-file**

EXEHDR found an unexpected end-of-file condition while reading the executable file. The file is probably corrupt.

U1140 **out of memory**

There was not enough memory for EXEHDR to decode the header of the executable file.

U1150 **page size too small; use option /PAGE:n to increase it**

The page size of an input library was too small, indicating an invalid input .LIB file.

U1151 **syntax error : illegal file specification**

A command operator was not followed by a module name or filename.

One possible cause of this error is an option specified with a dash (–) instead of a forward slash (/).

U1152 **syntax error : option name missing**

A forward slash (/) appeared on the command line without an option name after it.

U1153 **syntax error : option value missing**

The /PAGE option was given without a value following it.

U1154 **unrecognized option**

An unrecognized name followed the option indicator (/).

An option is specified by a forward slash (/) and a name. The name can be specified by a legal abbreviation of the full name.

U1155 **syntax error : illegal input**

A specified command did not follow correct LIB syntax.

U1156 **syntax error**

A specified command did not follow correct LIB syntax.

U1157 **comma or newline missing**

A comma or newline character was expected in the command line but did not appear.

One cause of this error is an incorrectly placed comma, as in the following command line:

```
LIB math.lib, -mod1 +mod2;
```

The line must be entered as follows:

```
LIB math.lib -mod1 +mod2;
```

U1158 **terminator missing**

The last line of the response file supplied to LIB did not end with a newline character.

U1161 **cannot rename old library**

LIB could not rename the old library with a .BAK extension because the .BAK version already existed with read-only protection.

Change the protection attribute on the .BAK file.

U1162 **cannot reopen library**

The old library could not be reopened after it was renamed with a .BAK extension.

One of the following may have occurred:

- Another process deleted the file or changed it to read-only.
- The floppy disk containing the file was removed.
- A hard-disk error occurred.

U1163 **error writing to cross-reference file**

The disk or root directory was full.

Delete or move files to make space.

U1164 **name length exceeds 255 characters**

A filename specified on the command line exceeded the LIB limit of 255 characters. Reduce the number of characters in the name.

U1170 **too many symbols**

The number of symbols in all object files and libraries exceeded the capacity of the dictionary created by LIB.

Create two or more smaller libraries.

U1171 **insufficient memory**

LIB did not have enough memory to run.

Remove any shells or resident programs, or add more memory.

U1172 **no more virtual memory**

The LIB session required more memory than the 1-megabyte limit imposed by LIB.

Try using the /NOE option or reducing the number of object modules.

U1173 **internal failure**

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

U1174 **mark : not allocated**

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

U1175 **free : not allocated**

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

U1180 **write to extract file failed**

The disk or root directory was full.

Delete or move files to make space.

U1181 **write to library file failed**

The disk or root directory was full.

Delete or move files to make space.

U1182 *filename* **: cannot create extract file**

The disk or root directory was full, or the given extract file already existed with read-only protection.

Make space on the disk or change the protection of the extract file.

U1183 **cannot open response file**

The response file was not found.

U1184 **unexpected end-of-file on command input**

An end-of-file character was received prematurely in response to a prompt.

U1185 **cannot create new library**

The disk or root directory was full, or the library file already existed with read-only protection.

Make space on the disk or change the protection of the library file.

U1186 **error writing to new library**

The disk or root directory was full.

Delete or move files to make space.

U1187 **cannot open temporary file VM.TMP**

The disk or root directory was full.

Delete or move files to make space.

U1188 **insufficient disk space for temporary file**

The library manager cannot write to the virtual memory.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

U1189 **cannot read from temporary file**

The library manager cannot read the virtual memory.

Note the circumstances of the error and notify Microsoft Corporation by following the instructions in the Microsoft Product Assistance Request form at the back of one of your manuals.

U1190 **interrupted by user**

LIB was interrupted with either CTRL+C or CTRL+BREAK.

U1191 *libraryname* **: cannot write to read-only file**

Operations cannot be performed on the given library because it is marked as a read-only file.

Change the protection attribute on the library.

U1200 *filename* **: invalid library header**

The input library file had an invalid format.

Either it was not a library file or it was corrupted.

U1203 *filename* **: invalid object file near** *location*

The given file was not a valid object file or was corrupted at the given location.

NMAKE and LIB Error Messages

Number	NMAKE and LIB Error Message

U2001 **no more file handles (too many files open)**

NMAKE could not find a free file handle.

One of the following may be a solution:

- Reduce recursion in the build procedures.
- In DOS, increase the number of file handles by changing the FILES setting in CONFIG.SYS to allow a larger number of open files. FILES=50 is the recommended setting.

U2152 *filename* **: cannot create listing**

One of the following may have occurred:

- The directory or disk was full.
- The cross-reference-listing file already existed with read-only protection.

U2155 *module* **: module not in library; ignored**

The specified module was not found in the input library.

One cause of this error is a filename or directory containing a a hyphen or dash (–). LIB interprets the dash as the operator for the delete command. This error occurs if you install a Microsoft language product in a directory that has a dash in its pathname, such as C:\MS-C. The SETUP program for a language calls LIB to create combined libraries, but the dash in the command line passed to LIB causes the library-building session to fail.

Another possible cause of this error is an option specified with a dash (–) instead of a forward slash (/).

U2157 *filename* **: cannot access file**

LIB was unable to open the specified file, probably because the file did not exist.

Check the path and filename.

U2158 *library* **: invalid library header; file ignored**

The given library had an incorrect format and was not combined.

U2159 *filename* **: invalid format (***number***); file ignored**

The given file was not recognized as a XENIX archive and was not combined.

NMAKE and LIB Warning Messages

Number **NMAKE and LIB Warning Message**

U4001 **command file can be invoked only from command line**

A command file, which is invoked by the at sign (@) specifier, cannot contain a specification for another command file. Such nesting is not allowed. The specification was ignored.

U4002 **resetting value of special macro** *macroname*

The given predefined macro was redefined.

U4004 **too many rules for target** *targetname*

More than one description block was specified for the given target using single colons (**:**) as separators. NMAKE executed the commands in the first description block and ignored later blocks.

To specify the same target in multiple dependencies, use double colons (**::**) as the separator in each dependency line.

U4005 **ignoring rule** *rule* **(extension not in .SUFFIXES)**

The given rule contained a suffix that is not specified in the **.SUFFIXES** list. NMAKE ignored the rule.

This warning appears only when the /P option is used.

U4006 **special macro undefined :** *macroname*

The given special macro name is undefined and expands to nothing.

U4007 **filename** *filename* **too long; truncating to 8.3**

The base name of the given file has more than eight characters, or the extension has more than three characters. NMAKE truncated the name to an eight-character base and a three-character extension.

If long filenames are supported by your file system, enclose the name in double quotation marks (**"**).

U4008 **removed target** *target*

NMAKE was interrupted while trying to build the given target, and the target file was incomplete. Because the target was not specified in the **.PRECIOUS** list, NMAKE deleted the file.

U4010 *target* **: build failed; /K specified, continuing ...**

A command in the commands block for the given target returned a nonzero exit code. The /K option told NMAKE to continue processing unrelated parts of the build and to issue an exit code 1 when the NMAKE session is finished.

If the given target is itself a dependent for another target, NMAKE issues warning U4011 after this warning.

U4011 *target* **: not all dependents available; target not built**

A dependent of the given target either did not exist or was out-of-date, and a command for updating the dependent returned a nonzero exit code. The /K option told NMAKE to continue processing unrelated parts of the build and to issue an exit code 1 when the NMAKE session is finished.

This warning is preceded by warning U4010 for each dependent that failed to be created or updated.

U4150 *module* **: module redefinition ignored**

A module was specified with the add operator (+) to be added to a library, but a module having that name was already in the library.

One cause of this error is an incorrect specification of the replace operator (– +).

U4151 *symbol* **: symbol defined in module** *module***; redefinition ignored**

The given symbol was defined in more than one module.

U4153 *option* : *value* **: page size invalid; ignored**

The argument specified with the /PAGE option was not valid for that option. The value must be an integer power of 2 between 16 and 32,768. LIB assumed an existing page size from a library that is being combined.

U4155 *modulename* **: module not in library**

The given module specified with a command operator does not exist in the library.

If the replacement command (– +) was specified, LIB added the file anyway. If the delete (–), copy (*), or move (– *) command was specified, LIB ignored the command.

U4156 *library* **: output-library specification ignored**

A new library was created because the filename specified in the *oldlibrary* field did not exist. However, a filename was also specified in the *newlibrary* field. LIB ignored the *newlibrary* specification.

For example, both of the following command lines cause this error if PROJECT.LIB does not already exist:

```
LIB project.lib +one.obj, new.lst, project.lib
LIB project.lib +one.obj, new.lst, new.lib
```

U4157 **insufficient memory, extended dictionary not created**

Insufficient memory prevented LIB from creating an extended dictionary.

The library is still valid, but the linker cannot take advantage of the extended dictionary to speed linking.

U4158 **internal error, extended dictionary not created**

An internal error prevented LIB from creating an extended dictionary.

The library is still valid, but the linker cannot take advantage of the extended dictionary to speed linking.

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

Microsoft®

1191 Part No. 24779