FORBAS CONTROL LANGUAGE

PROGRAMMING MANUAL

TABLE OF CONTENTS

1. INTRODUCTION

The FORBAS Control Language provides a software tool for manipulating FORTRAN and BASIC programs, complementing the hardware controls built into the teletypewriter (TTY) and described in the Console User's Guide. The tasks performed by FORBAS include system monitoring, file input/output, program utility functions, and line and program editing.

FORBAS commands differ from routine programming commands insofar as they do not require line numbers, are not stored, are executed immediately, and can be issued at any time. Like program commands, however, they are terminated with a carriage return (RETURN key). For the sake of smooth reading, no "CR" or "RET" is shown in examples, but the statements do not execute without it.

Another manual convention is to underline what the user keys into the TTY to avoid confusion with system-generated printout.

Rather than simply list commands alphabetically, which is done adequately enough by the index, this manual attempts to teach the language in the approximate order of need. To wit:

Chapter 2: LOGGING ON/OFF,

Chapter 3: BUILDING FILES,

Chapter 4: LISTING AND CHANGING FILES,

Chapter 5: COMBINING FILES.

You will find that the FORBAS commands become second nature with a little practice. The most instructive approach is to experiment on a small FORTRAN or BASIC program, changing statements slightly, listing by various methods, etc. Do not be frustrated by initial difficulties. As the computer would say, "to err is only human."

Appendix A lists a few technicalities to be considered when using FORBAS. Although they will probably be a bit incomprehensible at this juncture, you should at least skim them before going on to the next chapter.

2. LOGGING ON/OFF

LOGGING ON

The procedure for calling the computer depends on whether your TTY is hard-wired to it or whether you are using a telephone link. In the first case, simply turn the TTY ON (or to LINE or ON-LINE as your specific model requires). In the case of the telephone link, turn on the TTY, set the HALF/FULL DUPLEX switch to HALF, dial the appropriate phone number, wait for an answering tone, and finally place the telephone receiver in the acoustic coupler. If you are using a local line, you may simply have to dial a single digit on a dial built into the TTY keyboard.

Having contacted the computer, you are asked to supply your user number (four alphanumeric characters) and user password (four characters - no restrictions). The latter is disguised (typed over "MMMM?") to protect your account. End both numbers with a carriage return.

If you type in the wrong information, the TTY responds with INVALID USER NUMBER. Should you still want to log on, just call the computer and try again, and again...

OLD and NEW Commands

You are next asked to "TYPE OLD OR NEW", that is, state whether you want the computer to load a previously stored program or to accept a program you are about to create.

After you reply, the computer requests the name of the old or new program file.

SYSTEM Command

Once you have been given the "READY" indication, you should use the SYSTEM command to tell the computer whether your new program is to be written in FORTRAN or BASIC. You may type SYSTEM or SYS, to which the TTY responds with SYSTEM NAME--and waits for FOR or BAS. Alternatively, you may type SYS:BAS or SYS:FOR and the TTY will skip the formalities.

Once a program has been saved (i.e., is an OLD program), the system identification is not needed to reload it. The computer remembers the creation language.

The entire transaction to this point looks like the following (user type underscored):

```
            BCC TIME SHARING SYSTEM
    ON AT - 16:Ø1    PORT: 33
    USER NUMBER -- A123
    PASSWORD
    MMMM ?
    TYPE OLD OR NEW:NEW
    PROBLEM NAME:EXMPL1
    READY
    SYSTEM:BASIC
    READY
```

And now begin typing your program!

LOGGING OFF

BYE and GOODBYE Commands

Logging off the system is relatively uncomplicated.

Type BYE (or GOODBYE, according to how your mother raised you).

Turn off the TTY.

HELLO and RESTART Commands

If a brother (or sister) programmer is waiting to use your TTY, do not type BYE. Instead, type HELLO or RESTART and the TTY will cut back to the user number request sequence.

SYSTEM MONITORS

STATUS and TTY Commands

You can monitor the system at any time by typing STATUS or TTY. STATUS will cause the teletypewriter to type out the current state of the system (e.g., idle, run, list, save, etc.)

Typing TTY will give you the number of the current user, the name and system identification of the current program, and the amount of computer time used in the current run.

Example:

```
STATUS
IDLE
TTY
USER:A123   PROB:EXMPL1   SYS:BAS   RUNNING TIME:ØØ.7 SEC
```

READING PAPER TAPE

The procedures for punching and reading paper tape are described in the Console User's Guide. Because reading a tape also requires two FORBAS commands, this procedure is repeated here. See the Console User's Guide for details about keys mentioned, etc.

When reading paper tape be sure the tape surface is facing upward (small tape feed holes to the left).

If applicable press TD ON and set the tape read unit to either T, KT, or TTS mode.

TAPE Command

Type the command TAPE and set the tape read switch to START.

The tape read stops automatically when the last character (or CTRL/X OFF) is read. Reading can be halted at any time by hitting the read unit's STOP switch.

KEY Command

When finished, press TD OFF and type the command KEY to reset the computer to normal (keyboard input) mode.

3. BUILDING FILES

WHAT IS A FILE?

The term "file" is applied to several types of related information (user program, system utility program, input data, etc.). It may be only temporary, such as a program to perform a one-shot calculation, or it may be saved permanently on a disk storage unit.

Each user is assigned a file storage area. You are also assigned a user catalog that lists all your files by file name together with the location of the file in your specific user area. Your files may be accessed only through your own catalog. Each time you save or delete a file, your catalog is updated automatically.

Libraries

Two different types of library can be accessed under BCC file operation: the BCC system library and user libraries. The first contains utility packages developed at the BCC main installation. Although any customer may use these files, changes may be instituted only by BCC systems programmers.

Your group of programmers might also wish to establish a private library of frequently called routines or data. The group identification and the establishment of your library is handled at your own installation. User library names are limited to three alphanumeric characters. Maintenance and changes to the library are the responsibility of a designated librarian.

NAMING FILES

Commands described in this section:

NEW	RENAME
SCRATCH	RENAME:file

File names may contain up to six alphanumeric characters. Most symbols and non-printing elements are verboten, but you may use control (CTRL)A-Z. (Hold down the CTRL key while typing the desired letter key.)

```
     Acceptable:   MAXMIN
                   123$# ($# only legal symbols)
                   B^C C^C C^C 123
     Rejectable:   BCC 12 (space illegal)
                   SEXINAME (too many characters)
                   A12@ (@ illegal)
```

NEW Command

The name of your file is established during the log-in procedure using the NEW command.

```
     TYPE OLD OR NEW:NEW
     PROBLEM NAME:EXMPL1
```

Had you typed NEW:EXMPL1, the PROBLEM NAME line would have been omitted.

You may change the name of your file without changing its content using one of the RENAME commands, or conversely, you may delete its content while retaining the name using the SCRATCH command.

RENAME Command

Suppose you had stored the file named EXMPL1 initiated above. You might wish to play with a copy of this file without disturbing the stored version.

```
     TYPE OLD OR NEW:OLD
     PROBLEM NAME:EXMPL1
     READY
     RENAME
     PROBLEM NAME:EXMPL2
```

EXMPL1 is still intact in storage, and EXMPL2 is a new temporary file with the same content. The same operation could have been accomplished more directly with the command format

```
     RENAME:EXMPL2
```

In this instance the current file is renamed without the TTY asking again for PROBLEM NAME. The renamed file can also be saved if you wish.

SCRATCH Command

If you want to delete all the data in the current file without deleting the name, simply type SCRATCH.

```
     RENAME:EXMPL2
     READY
     SCRATCH
```

If you then ask for a file listing (with heading) you will receive a message like the following:

```
     EXMPL2   13:22   BCC   Ø8/14/7Ø   FRI.
```

```
NO PROGRAM

READY
```

The SCRATCH command doesn't affect stored versions of a program.

File Password

As an added file protection feature, you can add a password to your file identification. (This password is not to be confused with the four-character user password requested by the TTY as part of the log-on procedure.) A file thus secured can be loaded only if both the file name and password are supplied.

You should keep a personal list of your passwords and associated file names because only the file names are shown in a catalog listing.

The assignment of passwords is described in the next section as part of the SAVE command. In general, though, a password is limited to six characters, which may be alphabetic, numeric, control characters (CTRL) A-Z, plus internal codes 100-137 octal (lower case letters and other funnies).

Legal:	MYCODE
	$M^C I^C N^C E^C$
	$\{1|23\}$
Illegal:	SEXINAME (still too long)
	@ $1.25 (illegal symbols)

STORING/RETRIEVING FILES

Commands described in this section:

SAVE	OLD
UNSAVE	EDIT PACK

SAVE Command

Storing a file is an easy matter using the "straight" SAVE command. For the sake of example, let's define a short BASIC program to calculate the hypotenuse of a right triangle, remembering the formula,

(hypotenuse) $A^2 = B^2 + C^2$

$$A = \sqrt{B^2 + C^2}$$

This program is to be called BASEX1, a mnemonic for "BASIC example one." The BASIC statements themselves are ignored in this discussion as they are the subject of a separate BASIC Programming Manual.

```
TYPE OLD OR NEW:NEW

PROBLEM NAME:BASEX1

READY
SYS:BAS

READY
5 REM HYPOTENUSE
2Ø READ B,C
25 LET A = SQR (B↑2 + C↑2)
3Ø PRINT A
4Ø GOTO 2Ø
5Ø DATA 4,3, 6.23, 7.1, 2.1E6, 31ØØ
6Ø END
SAVE

READY
BYE
```

And like that you have a stored program! If a file named BASEX1 already exists in your catalog, it is deleted and the catalog entry changed to show the location of the new BASEX1 (let the namer beware!). Note that the straight SAVE command includes no allowance for a password or for access privileges for programmers with a different user name. These options are included in the SAVE variations.

SAVE:password	This saves the current file with a password. This format is also used when replacing a copy of an OLD file protected by the specified password. SAVE:POCUS
SAVE:new pwd, old pwd	Use this format to change the password of the current file. SAVE:HOCUS, POCUS
SAVE: ,old pwd	This command allows you to delete the password from the current file with no replacement. SAVE: , POCUS

You may want to let programmers with differing user numbers access your files. This is done using the SAVE command with the three "privilege codes" described below. All three codes may be specified in the same SAVE. You may also drop the file's password as in the last example shown above.

Code R allows others only to read and list your file.

Code W allows them to write over your file or add information.

Code E permits others to execute the file, but in no way to modify it.

Examples:

```
SAVE:  , POCUS, R, E
SAVE:  , ,W
SAVE:  , ,R,W,E
```

UNSAVE Command

The command to delete a saved file is, logically enough, UNSAVE, and comes in roughly the same flexibile varieties.

UNSAVE	Deletes the current file and updates your catalog and storage area accordingly.
UNSAVE: /password	Does the same for a current file protected by a password.
UNSAVE: file name	Deletes the named file rather than the current one.
UNSAVE: file name/pwd	I think you have the hang of it by now.

Cleaning up files that are no longer in use is a good practice---economical from the viewpoint of access time saved, storage space saved, and dollars saved (which is frequently what it's all about).

EDIT PACK Command

Before leaving the subject of economics and file storage, one more handy tool should be introduced. The EDIT PACK instruction removes all fill characters and extraneous blanks from the current file, allowing about a ten percent saving in storage requirements. Before using this command you would be wise to specify your SYSTEM. Otherwise weird and not very wonderful things may happen.

Applying EDIT PACK to the BASEX1 file saved earlier would give you the following:

```
BASEX1

5REMHYPOTENUSE
2ØREADB,C
25LETA=SQR(B↑2+C↑2)
3ØPRINTA
4ØGOTO2Ø
5ØDATA4,3,6.23,7.1,2.1E6,31ØØ
6ØEND
```

Note that there must be a space between EDIT and PACK. EDITPACK will cause a "NO SUCH FUNCTION" error message.

OLD Command

Retrieving and loading a saved file is usually just a matter of typing OLD and the file name in answer to the TTY's queries (or the short form OLD:file name). But as usual there are exceptions if your file has a password, or if you're accessing someone else's file or a library file.

For password-protected files use the form

```
PROBLEM NAME:BASEX1/POCUS
```

If another programmer has saved a file with read privileges, you may access it by giving the file name and his (or her) user number.

```
     PROBLEM NAME:BASEX1, B232
```

Three (3) asterisks (*) following the file name indicate that you're accessing a BCC installation library file.

```
     PROBLEM NAME:VECTOR***
```

Finally, if you are loading a file from your private library, list the file name and your library name separated by an asterisk.

```
     PROBLEM NAME:LOG1Ø*LIB
```

COMPILE/EXECUTE/STOP

Commands described in this section:

RUN	RUNBIG
RUNNH	STOP

RUN Command

The FORTRAN and BASIC compilers convert your program and data (source data) into a form understandable to the machine (object code). It can then be executed by the computer. In the BCC 500 system both the compilation and execution of the current program are initiated by one command - RUN.

While the program is actually executing, certain other commands are still recognized. These include STATUS, TTY, TAPE, KEY (see chapter 2), and the halt commands described later in this section. If you repeat the RUN command during execution, the TTY responds that the job is in "run" status and informs you of the elapsed running time since the job began.

Running the BASEX1 problem yields a console log sheet like the following:

```
TYPE OLD OR NEW:OLD

PROBLEM NAME:BASEX1

READY
RUN

BASEX1   17:38   BCC   Ø8/14/7Ø   FRI.

5
9.44579
2.1E+6
OUT OF DATA LINE # 2Ø

RUNNING TIME:  ØØ.7 SECS.

READY
BYE
```

As you recall, we fed the problem three sets of numbers in our DATA line (5Ø) of the program. The RUN statement causes the three answers to be calculated and printed as specified by the BASEX1 instructions.

RUNNH Command

If you wish to suppress printing of the heading,

```
BASEX1   17:38   etc.
```

use the command form RUNNH (RUN No Heading).

RUNBIG Command

If your program uses between 10K and 20K words of core, type RUNBIG instead of RUN. This forewarns the computer and keeps it from blowing its memory. For programs in the borderline area, bear in mind that RUNBIG jobs receive lower priority than RUN jobs. The BCC system actually handles programs up to 128K, but those larger than 20K should be negotiated with the company.

Suppose now that you are performing a vector analysis using a current program called VECTOR and have just run out of data. Another program in storage, named POINTS, will supply additional input. This program can be executed immediately (bypassing the OLD routing) using one of these formats:

```
RUN:program name
RUNNH:program name
RUNBIG:program name
```

The "program name" can actually be extended to include the program name, (other) user number, library name, or password.

```
RUNNH:POINTS, A123/HOCUS
```

After the POINTS program is executed, there is no current program. At this juncture you would probably type RUNNH:VECTOR and feed in your new data.

Console Halts and STOP

The RUN command causes a program to execute until the job is completed. Execution can be stopped at any point, however, using any of several convenient escapes.

As noted in the Console User's Guide, you may hit either the CTRL/SHIFT/@ key combination or the BREAK and BRK-RLS key combination at any time, depending on your particular TTY console. Print output can be stopped by hitting the "S" key (although this key is interpreted as normal input at other times). When no I/O operation is in progress, use the FORBAS terminator STOP (which is also considered input if typed in answer to an input request).

SUMMARY

The following console log illustrates many of the commands discussed in this chapter. It also introduces the LIST command, which is discussed in greater detail in the next chapter.

```
USER NUMBER -- A123

PASSWORD
MMMM ?

TYPE OLD OR NEW:OLD:BASEX1/MYCODE

READY
RENAME:BASEX2

READY
LIST

BASEX2    22:Ø8    BCC    Ø8/14/7Ø    FRI.

5 REM HYPOTENUSE
2Ø READ B,C
25 LET A = SQR (B↑2 + C↑2)
```

```
30 PRINT A
40 GOTO 20
50 DATA 4,3, 6.23, 7.1, 2.1E6, 3100
60 END

READY
RUNNH

5
9.44579
2.1E+6
OUT OF DATA LINE # 20

RUNNING TIME:  00.6 SECS.

READY
SCRATCH

READY
LIST

BASEX2    22:09    BCC    08/14/70    FRI.

NO PROGRAM

READY
BYE
```

4. LISTING AND CHANGING FILES

The functions described in chapters 2 and 3 are used to process entire files. Those discussed here are primarily line functions, although some categories run the spectrum from single-character to entire-catalog manipulation.

LIST COMMANDS

Commands described in this section:

LIST	CATALOG
LISTNH	EDIT PAGE
EDIT LIST	EDIT TEXT

LIST and LISTNH Commands

List commands are a good illustration of the range of flexibility just mentioned. LIST is used primarily to produce an immediate printout of the current file, as was done in the summary example of chapter 3. Like RUN, it also has a heading suppressing option, LISTNH. Should you want to see only a specific portion of a file, however, you can use one of the following formats:

```
LIST:line number
LISTNH:line number
```

Executing this instruction, the printout begins at the indicated line number and continues to the end of the file or until one of the print halt options (see chapter 3) is exercised.

<u>EDIT LIST Command</u>

The EDIT LIST command allows even more leeway in listing specific lines. Telling the computer to

```
     EDIT LIST 1Ø, 3Ø, 5Ø, 8Ø - 13Ø
```

prints out the three individual lines referenced, plus the block of lines 8Ø through 130 inclusive. <u>Again note that there must always be a space between each word of a two-word command and between the last word and first argument</u>. The only limitation on the number of arguments permitted is the length of the input line.

The arguments shown above are listed in ascending order, but this is not necessary. In fact, 130-80 would have printed the block in reverse order. An EDIT LIST command with no arguments causes the entire file to be printed in reverse order.

To get a feel for the various listing possibilities, study the factorial example below. First the entire program is listed, then the subroutine only, next the entire program in reverse, the numbers being factored plus the subroutine, and finally a couple of stray lines plus the subroutine in reverse order.

```
READY
LIST

BASEX2    16:48    BCC    Ø8/14/7Ø    FRI.

5  REM PRINT FACTORIAL N
1Ø LET N = 1Ø
2Ø GOSUB 8Ø
3Ø LET N = 15
4Ø GOSUB 8Ø
5Ø LET N = 2Ø
```

```
6Ø GOSUB 8Ø
7Ø STOP
8Ø    LET F = 1
9Ø    FOR G = 1 TO N STEP 1
1ØØ   LET F = F * G
11Ø   NEXT G
12Ø   PRINT N;F
13Ø   RETURN
14Ø   END

READY
LISTNH:8Ø

8Ø    LET F = 1
9Ø    FOR G = 1 TO N STEP 1
1ØØ   LET F = F * G
11Ø   NEXT G
12Ø   PRINT N;F
13Ø   RETURN
14Ø   END

READY
EDIT LIST

14Ø   END
13Ø   RETURN
12Ø   PRINT N;F
11Ø   NEXT G
1ØØ   LET F = F * G
9Ø    FOR G = 1 TO N STEP 1
8Ø    LET F = 1
7Ø    STOP
6Ø    GOSUB 8Ø
5Ø    LET N = 2Ø
4Ø    GOSUB 8Ø
3Ø    LET N = 15
2Ø    GOSUB 8Ø
1Ø    LET N = 1Ø
5     REM PRINT FACTORIAL N

READY
EDIT LIST 1Ø, 3Ø, 5Ø, 8Ø-13Ø

1Ø    LET N = 1Ø
3Ø    LET N = 15
5Ø    LET N = 2Ø
8Ø    LET F = 1
9Ø    FOR G = 1 TO N STEP 1
1ØØ   LET F = F * G
11Ø   NEXT G
12Ø   PRINT N;F
13Ø   RETURN
```

```
READY
EDIT LIST 5, 13Ø-8Ø, 140

5     REM PRINT FACTORIAL N
13Ø   RETURN
12Ø   PRINT N;F
11Ø   NEXT G
1ØØ   LET F = F * G
9Ø    FOR G = 1 TO N STEP 1
8Ø    LET F = 1
14Ø   END
```

CATALOG Command

The content of your catalog may be listed also. If you only want to see the names of the files in your catalog, type CATALOG. Passwords attached to a file name will not be shown.

To list library file names, use one of these:

CATALOG*** CAT***	BCC System Library
CATALOG:user library name CAT:user library name	Specified User Library

EDIT PAGE and EDIT TEXT

To see the content of your cataloged files or BCC system files, as well as their names, type in either the EDIT PAGE or EDIT TEXT command. Both use the same format, but EDIT TEXT suppresses program line numbers in the printout.

```
EDIT PAGE progl-page, prog2,... prog9

EDIT TEXT progl-page, prog2,...prog9
```

As the form implies, as many as nine programs can be printed out with one command. The "page" indication tells the computer what to number the first page of the list. If you prefer not to specify this option, the default (assumed) value is one.

The format of the printed output is as follows:

- 8-1/2 x 11 pages, divided by dashed lines;
- Page numbers centered at top;
- Up to 50 lines per page;
- Ten blank lines between programs (but if there are less than 20 lines left on a page, the new program begins on the following page.

The following example shows how the outputs from EDIT PAGE and EDIT TEXT differ. Because only user files are listed, there is no need to distinguish between BASIC and FORTRAN programs.

```
READY
EDIT PAGE BASEX1, BASEX2, FOREX1
---------------------------------------------------------------------


                               - 1 -

BASEX1

5  REM HYPOTENUSE
2Ø READ B,C
25 LET A = SQR (B↑2 + C↑2)
3Ø PRINT A
4Ø GOTO 2Ø
5Ø DATA 4,3, 6.23, 7.1, 2.1E6, 31ØØ
6Ø END
```

BASEX2

```
  5  REM PRINT FACTORIAL N
 1Ø LET N = 1Ø
 2Ø GOSUB 8Ø
 3Ø LET N = 15
 4Ø GOSUB 8Ø
 5Ø LET N = 20
 6Ø GOSUB 8Ø
 7Ø STOP
 8Ø LET F = 1
 9Ø FOR G = 1 TO N STEP 1
1ØØ LET F = F * G
11Ø NEXT G
12Ø PRINT N;F
13Ø RETURN
14Ø END
```

FOREX1

```
1ØC         FIND THE LARGEST OF THREE NUMBERS
2Ø          READ 12, L, M, N
3Ø       12 FORMAT (3I7)
4Ø          IF (L.GT.M) GOTO 3
5Ø          IT = M
6Ø        2 IF (IT.LT.N) IT = N
7Ø          PRINT 1Ø,IT
8Ø          CALL EXIT
```

- 2 -

FOREX1 CONTINUED

```
9Ø       3 IT = L
1ØØ        GO TO 2
11Ø     1Ø FORMAT ("THE LARGEST NUMBER IS", I7)
12Ø        END
```

(HIT "S" KEY AT THIS POINT TO STOP LINE SKIPPING)

```
READY
EDIT TEXT BASEX2-3,FOREX1
```

BASEX2

```
REM PRINT FACTORIAL N
LET N = 1Ø
GOSUB 8Ø
LET N = 15
GOSUB 8Ø
LET N = 2Ø
GOSUB 8Ø
STOP
LET F = 1
FOR G = 1 TO N STEP 1
LET F = F * G
NEXT G
PRINT N;F
RETURN
END
```

FOREX1

```
C          FIND THE LARGEST OF THREE NUMBERS
           READ 12, L, M, N
      12   FORMAT (3I7)
           IF (L.GT.M) GOTO 3
           IT = M
       2   IF (IT.LT.N) IT = N
           PRINT 1Ø,IT
           CALL EXIT
       3   IT = L
           GO TO 2
      10   FORMAT ("THE LARGEST NUMBER IS", I7)
           END
```

RESEQUENCING FILES

Two commands are available for rearranging and renumbering your files: EDIT MOVE and EDIT RESEQUENCE.

EDIT MOVE Command

EDIT MOVE lets you move a single line or block of lines to a new position in your program.

Examples:

<u>EDIT MOVE   7Ø-11Ø, 15Ø</u>

<u>EDIT MOVE   7Ø, 15Ø</u>

In the first case the block of lines 7Ø through 11Ø are moved and inserted following line 15Ø. The secon moves only line 7Ø to the position following line 15Ø. This command will not execute if the new location (15Ø in this case) falls within the specified block being moved. EDIT MOVE 7Ø-11Ø, 1ØØ will hang you up. Trying to visualize such a move will hang you up too.

Following the EDIT MOVE, lines 7Ø-11Ø in the first example would be renumbered in increments of one, 151, 152, 153, etc. If the number of lines in the block overlaps the instruction(s) originally following line 15Ø, the latter is bumped up and resequenced and the message BLOCK TOO LARGE is issued (for your information only).

EDIT RESEQUENCE Command

EDIT RESEQUENCE provides you with a neat answer to messes created by many insertions or moves. It looks tricky at first because it requires all of three arguments (aaargh!), but these can be mastered with a bit of concentration and a few applications.

EDIT RESEQUENCE arg1, arg2, arg3

arg1 - This number will be the first line of the resequenced file (or file portion);

arg2 - May be a single line number or block of lines (e.g., 7Ø-11Ø); represents the present first line (or entire block) where resequencing is to start; "Ø" alone will resequence the entire file;

arg3 - This number is the increment between the resequenced line numbers.

Easy Out - You may simply specify EDIT RESEQUENCE and default values 1ØØ, Ø, 1Ø are assumed. But easy outs add no flair to your life style!

When using EDIT RESEQUENCE, you should specify which system you're working in. Line number references within BASIC programs or FORTRAN statement numbers may be incorrectly adjusted otherwise.

In the following examples, OLD friend BASEX2 has been loaded and a remark added at line 75.

```
READY
SYSTEM:BASIC

READY
75   REM CALCULATE FACTORIAL N
LISTNH
5    REM PRINT FACTORIAL N
1Ø   LET N = 1Ø
2Ø   GOSUB 8Ø
3Ø   LET N = 15
4Ø   GOSUB 8Ø
5Ø   LET N = 2Ø
6Ø   GOSUB 8Ø
7Ø   STOP
75   REM CALCULATE FACTORIAL N
8Ø   LET F = 1
9Ø   FOR G = 1 TO N STEP 1
1ØØ  LET F = F * G
11Ø  NEXT G
```

```
120    PRINT N;F
130    RETURN
140    END
```

Being all-wise, we decide the added remark would be more appropriate at the head of the program.

```
READY
EDIT MOVE 75,0

READY
LISTNH
1    REM CALCULATE FACTORIAL N
5     REM PRINT FACTORIAL N
10    LET N = 10
20    GOSUB 80
30    LET N = 15
40    GOSUB 80
50    LET N = 20
60    GOSUB 80
70    STOP
80    LET F = 1
90    FOR G = 1 TO N STEP 1
100   LET F = F * G
110   NEXT G
120   PRINT N;F
130   RETURN
140   END
```

But now, being fastidious by nature, we're displeased with the irregular line numbering system.

```
READY
EDIT RESEQUENCE 10,1,10

READY
LISTNH
10    REM CALCULATE FACTORIAL N
20     REM PRINT FACTORIAL N
30    LET N = 10
40    GOSUB 100
50    LET N = 15
60    GOSUB 100
70    LET N = 20
80    GOSUB 100
90    STOP
100    LET F = 1
110    FOR G = 1 TO N STEP 1
120   LET F = F * G
130   NEXT G
140   PRINT N;F
150   RETURN
160   END
```

In the resequenced version lines 4Ø, 6Ø, and 8Ø read GOSUB 1ØØ to allow for the changed location of the subroutine.

INSERTIONS

Insertions and deletions are to some extent interwoven. An inserted line, for example, could be a replacement for another line with the same number. EDIT REPLACE performs the same function with characters.

Line Insertion

The simplest method of line insertion involves no command at all. In the example ending the last section, the line 75 REM statement was simply typed and the computer automatically placed it between lines 7Ø and 8Ø. As noted, we could also use this form to change a line. For example,

```
30  LET N = 5
```

could be typed to replace the current

```
30  LET N = 10
```

EDIT DUPLICATE Command

If a line is repeated in a program, the EDIT DUPLICATE command may be used to save retyping the statement each time it appears. All duplicated lines are preserved in their original positions as well as appearing in the specified new location(s).

In the present version of BASEX2, we could have typed

```
EDIT DUPLICATE 4Ø, 5Ø, 7Ø
```

instead of typing GOSUB 1ØØ three times. Here we are telling the computer to duplicate line 4Ø following lines 5Ø and 7Ø. Insert line numbers may appear in any order and as many may be specified as fit in the input line. The only requirement is that the line being duplicated be listed first.

To duplicate a <u>block</u> of lines, use the same format.

<u>EDIT DUPLICATE 1Ø-7Ø, 1ØØ, 15Ø, 125, etc.</u>

After the EDIT DUPLICATE is executed, your program is resequenced using the EDIT RESEQUENCE default options (1ØØ, Ø, 1Ø). Again, indicate the system you are using to preserve consistency in your internal references.

DELETIONS

Deletions fall into several groups, i.e., deletions of entire files, of lines or blocks of lines, or of individual characters. For the first category see chapter 3, where the SCRATCH and UNSAVE commands are discussed.

EDIT EXTRACT Command

Lines and/or blocks of lines can be deleted using the EDIT EXTRACT or EDIT DELETE commands. EDIT EXTRACT specifies the numbers of lines to be saved in a program and the rest of the file is deleted. Simply designate as many lines or blocks of lines as you wish, in any order, up to the limit of the input line length.

<u>EDIT EXTRACT 10-30, 70, 100-160, 90</u>

EDIT DELETE Command

EDIT DELETE is essentially a cameo of EDIT EXTRACT. Instead of specifying lines to be saved and "cutting away" the rest of the program, EDIT DELETE indicates which lines are to be removed and the remainder of the program stays intact. The same formatting rules apply as for EDIT EXTRACT.

```
EDIT DELETE  1Ø-4Ø, 8Ø, 1ØØ-12Ø, 7Ø
```

In the following example, file BASEX2 is copied as BASEX3 and chopped up using the EDIT EXTRACT and EDIT DELETE commands. Compare the printout to the last version of the program listed in the EDIT RESEQUENCE example.

```
RENAME:BASEX3

READY
EDIT DXTRACT 1Ø, 6Ø, 8Ø-16Ø, 4Ø

READY
LISTNH
1Ø    REM CALCULATE FACTORIAL N
4Ø    GOSUB 1ØØ
6Ø    GOSUB 1ØØ
8Ø    GOSUB 1ØØ
9Ø    STOP
1ØØ   LET F = 1
11Ø   FOR G = 1 TO N STEP 1
12Ø   LET F = F * G
13Ø   NEXT G
14Ø   PRINT N;F
15Ø   RETURN
16Ø   END

READY
EDIT DELETE 60-90, 40

READY
LISTNH
1Ø    REM CALCULATE FACTORIAL N
1ØØ   LET F = 1
```

```
110  FOR G = 1 TO N STEP 1
120  LET F = F * G
130  NEXT G
140  PRINT N;F
150  RETURN
160  END
```

CTRL/X Keys

To delete the line currently being typed, hold down the CTRL key and type X (or use a corresponding console control, depending on your teletypewriter model--see function table in Console User's Guide, chapter 2).

SHIFT/← Keys

To delete the last character typed, hold down the SHIFT key and type ← (over letter O). Refer to the Console User's Guide function table for variations. To delete several consecutive characters, hit the SHIFT/← for each one to be erased.

EDIT REPLACE COMMAND

EDIT REPLACE stands somewhat apart from the other EDIT commands, both in function and format. Hence, its separate treatment here at the end of the chapter. It could be classified as an "advanced" instruction, like the EDIT MERGE and EDIT WEAVE commands introduced in the next chapter.

EDIT REPLACE lets you specify the character(s) in a line, including blanks, that you wish to supplant and what the replacement is to be. You may enter several such instructions in a single command line, or you may use EDIT REPLACE to execute a file made up of many replacement commands.

In its simplest form this command might be used to correct a few characters in a line.

```
10  REM CALCALATE FACTORIAL N
EDIT REPLACE #LCAL#LCUL#10
LISTNH:10
10  REM CALCULATE FACTORIAL N
```

Line scans begin with the first character following the line number. Be sure you identify the replaced characters uniquely. Should you type only #CAL# as the character string to be replaced with CUL in this example, the first CAL of "CALCALATE" would be changed also.

The general format rule for EDIT REPLACE is as follows:

EDIT REPLACE arg1#replace#replacement#arg2, arg3

where

arg1 represents the maximum number of replacements in each line (default value is "unlimited");

# acts as a separator (you may also use characters $, !, ", %, ', or &);

"replace" identifies the current character string to be removed;

"replacement" is the new character string;

arg2 is the line number where the replacement search begins (default value is zero);

arg3 is the number of the final line to be scanned (default value is 999999).

If both "arg2" and "arg3" are omitted, the entire file is scanned. If only "arg2" is given, as in the example above (1Ø), then only that line is scanned and the default for "arg3" is ignored.

This format can be repeated in the same line to provide multiple replacements, with each repetition divided by a colon.

```
EDIT REPLACE 1#BIT#BYTE#1Ø,1ØØ:#HEAD#READ#3Ø
```

Multiple commands are executed sequentially. Be careful that you do not unintentionally wipe out a replacement performed earlier in the same command sequence. A refinement of the EDIT REPLACE format also lets you perform "non-specific" character replacement, using "?" to indicate unknown characters.

```
EDIT REPLACE #?OCUS#HOCUS#
```

HOCUS replaces any five-character string ending in OCUS.

```
EDIT REPLACE#PRINT???#WRITE(OUT,???)#
```

Here we replace all PRINT statements with WRITE(OUT,   ) while retaining any data following the statements. The point to remember here is that the number of question marks in the "replace" and "replacement" sections must be the same.

Earlier we said EDIT REPLACE could be used to execute entire files of replacements as well as performing single line assignments. This ability comes in very handy for extensive conversion work. In this case we use the form

```
     EDIT REPLACE BASOUT
```

where BASOUT is a routine for converting BASIC format programs to FORTRAN format.

```
     1Ø    #IF#IF  (#
     2Ø    #THEN#)    GOTO#
     3Ø    #LET##
           etc.
```

Note that statement 3Ø completely deletes, rather than replaces, every occurance of LET.

5. COMBINING FILES

Program and subroutine files can be combined using either the EDIT WEAVE or EDIT MERGE command. EDIT WEAVE combines files in the sequence of existing line numbers. EDIT MERGE allows you to combine subprograms with a main program in any order you wish, and then resequences the resultant file.

EDIT WEAVE COMMAND

EDIT WEAVE lets you combine as many as nine saved programs. Programs are woven together and existing line numbers retained. If two lines have the same number, one will be lost (the last mentioned is retained). Take care that you do not end up with more than one END statement or an END statement stuck in the heart of your new program. Not only does this create a painful image, it also frustrates the compiler.

Suppose we wanted to combine our BASEX1 hypotenuse calculation function and BASEX2 factorial problem. Remembering that line 6Ø contained the END statement in BASEX1, we could load BASEX2, resequence it to overlay line 6Ø, rename and save the resequenced program, and perform the weave.

```
READY
OLD:BASEX2

READY
EDIT RESEQUENCE 6Ø,Ø,1Ø

READY
RENAME:BASEX4
```

```
READY
SAVE

READY
EDIT WEAVE BASEX1,BASEX4

READY
LISTNH

5 REM HYPOTENUSE
2Ø READ B,C
25 LET A = SQR (B↑2 + C↑2)
3Ø PRINT A
4Ø GOTO 2Ø
5Ø DATA 4,3, 6.23, 7.1, 2.1E6, 31ØØ
6Ø REM CALCULATE FACTORIAL N
7Ø REM PRINT FACTORIAL N
8Ø LET N = 1Ø
9Ø GOSUB 15Ø
1ØØ LET N = 15
11Ø GOSUB 15Ø
12Ø LET N = 2Ø
13Ø GOSUB 15Ø
14Ø STOP
15Ø LET F = 1
16Ø FOR G = 1 TO N STEP 1
17Ø LET F = F * G
18Ø NEXT G
19Ø PRINT N;F
2ØØ RETURN
21Ø END
```

EDIT MERGE COMMAND

EDIT MERGE also allows you to combine as many as nine saved files. With this command, however, you may specify where the subprograms being merged should be inserted into the main program. After the merger, the new program is resequenced (1ØØ,Ø,1Ø), and as usual this means you should tell the computer what system you're in.

The command format is

```
EDIT MERGE main, sub1-n, sub2-n...
```

where

"main" represents the main program name;

"sub1, sub2" are the subprograms being merged
into the main program;

"n" represents the line number in "main" that
this particular subprogram is to follow; if
no line numbers are specified, the subprograms
are appended sequentially after the last
statement in "main."

Again, be certain there is no more than one END statement and that it is properly placed in your program.

In the following example two data files are created and <u>saved</u> to supply more input for our original BASEX1 program.

DATA 1

```
10  REM DATA STATEMENTS FOLLOW
20  DATA 1, 2, 5, 6
30  DATA 10, 11, 12, 13, 14, 15
```

DATA 2

```
10  REM MORE DATA
20  DATA 20, 21, 22, 23
30  DATA 30, 31, 32, 33
```

Note that neither file contains an END statement, and that we're not conerned with lines having the same number.

Now we can EDIT MERGE the files, first with no insertion line specification, then in a more logical construction.

<u>EDIT MERGE BASEX1,DATA1,DATA2</u>

```
READY
LISTNH
1ØØ REM HYPOTENUSE
11Ø READ B,C
12Ø LET A = SQR (B↑2 + C↑2)
13Ø PRINT A
14Ø GOTO 2Ø
15Ø DATA 4,3, 6.23, 7.1, 2.1E6, 31ØØ
16Ø END
17Ø REM DATA STATEMENTS FOLLOW
18Ø DATA 1,2,5,6
19Ø DATA 1Ø,11,12,13,14,15
2ØØ REM MORE DATA
21Ø DATA 2Ø,21,22,23
22Ø DATA 3Ø,31,32,33
```

This new file will not compile because the END statement is not the highest numbered line.

```
EDIT MERGE BASEX1,DATA1-4Ø,DATA2-5Ø

READY
LISTNH
1ØØ REM HYPOTENUSE
11Ø READ B,C
12Ø LET A = SQR (B↑2 + C↑2)
13Ø PRINT A
14Ø GOTO 2Ø
15Ø REM DATA STATEMENTS FOLLOW
16Ø DATA 1,2,5,6
17Ø DATA 1Ø,11,12,13,14,15
18Ø DATA 4,3, 6.23, 7.1, 2.1E6, 31ØØ
19Ø REM MORE DATA
2ØØ DATA 2Ø,21,22,23
21Ø DATA 3Ø,31,32,33
22Ø END
```

Neither EDIT WEAVE nor EDIT MERGE affects the saved versions of the combined files. If you want to keep both the new and old programs, rename the new one before saving it.

APPENDIX A: GENERAL FORBAS CONSIDERATIONS

1. Numbers appearing in commands may be preceded or followed by any number of blanks. Command names, however, must be separated from each other and from their arguments by at least one blank (except for commands followed by a colon (:) -- OLD, NEW, RENAME, etc.).

2. The only restriction on the number of intervals, arguments, etc., is the length of the input line.

3. Every command changes your program, but only SAVE and UNSAVE affect the permanent copy.

4. SYSTEM should be specified whenever a resequencing command is invoked (RESEQUENCE, DUPLICATE, PACK, MOVE, MERGE).

5. While files without line numbers can be created by programs, they cannot be modified with the editing commands. They can be printed out using LIST, LISTNH, EDIT TEXT, or EDIT LIST (without arguments). EDIT MERGE also accepts unnumbered files and reads them correctly. After resequencing they do, of course, have numbers.

6. Unless the end letters of a command are vital (e.g., LISTNH, RUNNH), the command can be invoked by typing only the first three letters and the carriage return (e.g., EDI LIS for EDIT LIST, GOO for GOODBYE).

7. Files must be saved before they can be used as parameters by EDIT MERGE and EDIT WEAVE. These commands ignore the current program.

8. Syntax Specifications:

User numbers:	Four characters,
	Ø→9 A→Z

User passwords:	Four characters, no restrictions
User libraries:	Three characters, $Ø \rightarrow 9$ $A \rightarrow Z$
File names:	Six characters, #$ $Ø \rightarrow 9$ $A \rightarrow Z$ $A^C \rightarrow Z^C$
File passwords:	Six characters, $Ø \rightarrow 9$ $A \rightarrow Z$ $A^C \rightarrow Z^C$, plus internal codes 1ØØB--137B

9. All statements in this manual are true forever, except for those to be modified at some future date.

PEACE

APPENDIX B: BCC LIBRARY FILES

Not available

yet

INDEX