VII. Full Screen Processing, Summary, and Examples

IBM

Learning System/23 BASIC

VII. Full Screen Processing, Summary, and Examples

IBM

Learning System/23 BASIC

First Edition (January 1981)

Use this publication only for the purpose stated in the Preface.

Changes are periodically made to the information herein; any such changes will be reported in subsequent revisions or Technical Newsletters.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Publications are not stocked at the address given below. Requests for copies of IBM publications should be made to your IBM representative or the IBM branch office serving your locality.

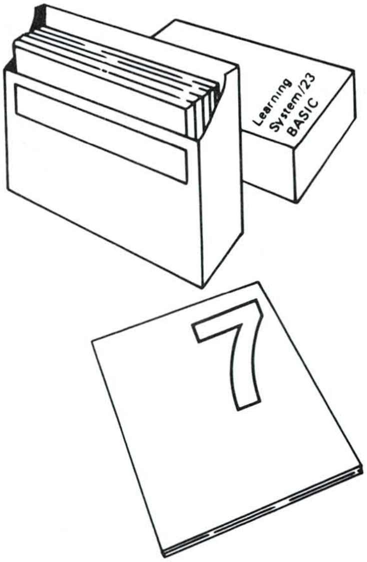
This publication could contain technical inaccuracies or typographical errors. A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to IBM Corporation, Systems Publications, Department 27T, P.O. Box 1328, Boca Raton, Florida 33432. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

VII. Full screen processing, summary and examples

Contents

VII. Full screen processing, summary and examples

About this book



This is the seventh in your series of seven books on *Learning System/23 BASIC*. By now, you should be familiar with the fundamental statements and commands that are used to program your System/23.

In Chapter 1 of this book, we will show you one more feature that is available on your System/23. You will learn how to use *full screen processing*. Full screen processing allows you to input data from or output data to specific areas on the screen

In the remaining chapters of this book, we will review what you have learned. Chapter 2 is about organizing a program and solving a problem. It includes *flowcharts*, which are diagrams showing the solution to a problem.

Chapter 3 contains three example programs. These programs use the BASIC statements you have studied. Notes are included to explain the programs.

Chapter 4 contains a flowchart and example program. This program is an inventory control program. Notes are included to explain each section of the program.

Chapter 1. Full screen processing

Introduction

Up to this point, you have used only one line of the screen at a time in your programs. You know how to display data on the screen and input data from the screen, one line at a time.

In this chapter, you will learn how to use the entire screen to input or output data. You will learn how to place the cursor in any position on the screen.

If you have run any of the IBM supplied application programs, you have already used full screen processing. The Customer Support Functions also use full screen processing. Now you will learn how to use it in your programs, too.

Objectives

Upon completion of this chapter, you should be able to do the following:

- Display data in a specific area on the screen by using the PRINT FIELDS statement.
- Input data from a specific area on the screen by using the INPUT FIELDS statement.

If you are familiar with these tasks, try the exercises located at the end of this chapter. If not, read through the chapter before going on to the exercises.

Full screen processing

Displaying or inputting one line of data

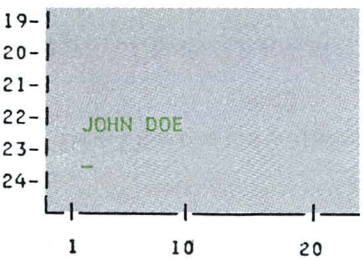
You already know how to display information on your screen. Remembering what you have already learned about PRINT, enter a 2-line program to display your name. Use line numbers 10 and 20, and don't forget to enter CLEAR.

Answer: ______________________________

If your name is John Doe, your answer should look like this:

```
CLEAR
10 PRINT "JOHN DOE"
20 END
```

Now run your program:




```
RUN
```

Your name should be displayed at the bottom of the screen on line 23. Your name moves up, and the cursor appears. Then the words READY INPUT should appear beneath the cursor.

If you had wanted to display your name on a clear screen, your answer would have looked like this:

```
CLEAR
10 PRINT NEWPAGE;"JOHN DOE"
20 END
```

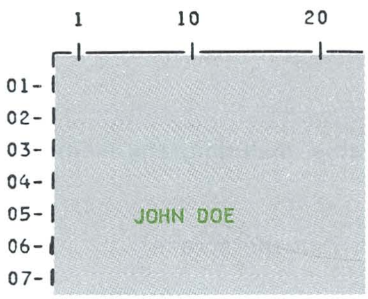
With either program, your name is always displayed on line 23, because the PRINT statement always causes data to be displayed there. Then your name moves up, and the cursor appears.

Now we'll show you how to display your name in a specific location on the screen. Enter the following program. (Don't forget to enter your name in place of John Doe.)

```
CLEAR
10 A$="5,7,C 18"
20 NAME$="JOHN DOE"
30 PRINT NEWPAGE
40 PRINT FIELDS A$:NAME$
50 END
```

Remember that what you enter for NAME$ can not have more than 18 characters. Now run the program:

```
RUN
```



Your name should be displayed on the fifth line of the screen, starting in column seven. This placement of data on the screen is called *full screen processing*.

Full screen processing allows you to use specific areas of the screen to input or display data. You specify the location of the data by defining *fields*. A *field* is one or more consecutive positions on the screen.

When defining a field, you specify the line and column of the first position. In our example,

```
10 A$="5,7,C 18"
```

Line Column

This field begins in line 5, column 7 of the screen.

Displaying or inputting one line of data (continued)

The fields of the screen in full screen processing are similar to the fields of a record in a record I/O file. They both describe the location of data.

Line 40 of your program could also look like this:

```
40 PRINT #0,FIELDS A$:NAME$
```

Because the #0 is not necessary, we will not include it in any of the examples in this book.

Let's look again at the statement that describes the field.

```
      10 A$="5,7,C 18"
```

Character Line Column Type of variable data

The A$ can be any character variable, including any element of a character array.

This field begins in line 5, column 7 of the screen.

The data item to be displayed is a character string of up to 18 characters. As with data specifications in a FORM statement, C specifies a character string.

You can also use N or PIC in a field definition. They have the same meaning that they have in a FORM statement.

What field would this statement define?

```
60 FULL$(1)="11,24,N 7.2"
```

It defines a field beginning in line 11, column 24. The data item would be a number with up to seven digits, with two digits to the right of the decimal point.

Your turn!

Where would the field defined by B$ begin?

```
80 B$="23,1,C 10"
```

Answer: ____________________________________

It would begin in line 23, column 1, the same place where the output from a regular PRINT statement begins.

Let's see what happens when we input your name from the field beginning in line 5, column 7. Enter the following:

```
DEL 20
40 INPUT FIELDS A$:NAME$
```

List your new program:

```
LIST
```

```
00010 LET A$="5,7,C 18"
00030 PRINT NEWPAGE
00040 INPUT FIELDS A$:NAME$
00050 END
_
```

Now run your program:

```
RUN
```

Full screen processing

Displaying or inputting one line of data (continued)

The NEWPAGE should clear the screen, and the cursor should now be positioned at line 5, column 7. Go ahead and enter your name.

Note: The way you enter data with full screen processing is different from the way you normally enter data. With full screen processing, you press either the Field Exit key, the New Line key, or the Field Advance key after each input field to get to the next input field. You press the Enter key after the last input field on the screen.

```
       1       10        20
      ┌|--------|---------|---
  01- |
  02- |
  03- |
  04- |
  05- |     JOHN DOE
  06- |
  07- |
```

In this program, you are entering data from only one input field. Therefore, you can press the Enter key after you type your name.

Now enter:

```
PRINT NAME
```

to see that your name really did get input into the variable NAME.

```
PRINT NAME$
JOHN DOE
```

Later in this chapter, we will show you how to write a program that uses more than one input field. Then, you will need to use both the Field Exit or Field Advance key and the Enter key.

Displaying or inputting several lines of data

Now let's look at an example that displays data on more than one line. Enter the following program. (Enter your name and address in lines 40-60. Notice that you are limited to a maximum of 18 characters for each variable.)

```
CLEAR
10 OPTION BASE 1
20 REM DISPLAY NAME AND ADDRESS
30 DIM FS$(3)*8
40 N$="JOHN DOE"
50 S$="125 1ST ST."
60 C$="CHICAGO, IL"
70 FS$(1)="3,4,C 25"
80 FS$(2)="4,4,C 25"
90 FS$(3)="5,4,C 25"
100 PRINT NEWPAGE
110 PRINT FIELDS MAT FS$:N$,S$,C$
120 END
```

The statement in line 110 causes three lines of data to be displayed. The three fields are defined in the three elements of the FS$ array.

FIELDS MAT FS$ tells your System/23 to use the elements of the FS$ array to define the fields for the items being displayed. N$ uses FS$(1). S$ uses FS$(2). C$ uses FS$(3).

The first item to be displayed uses the first field defined in the array. The second item uses the second field, etc. If an array has more elements than the number of items being displayed, the extra elements are ignored.

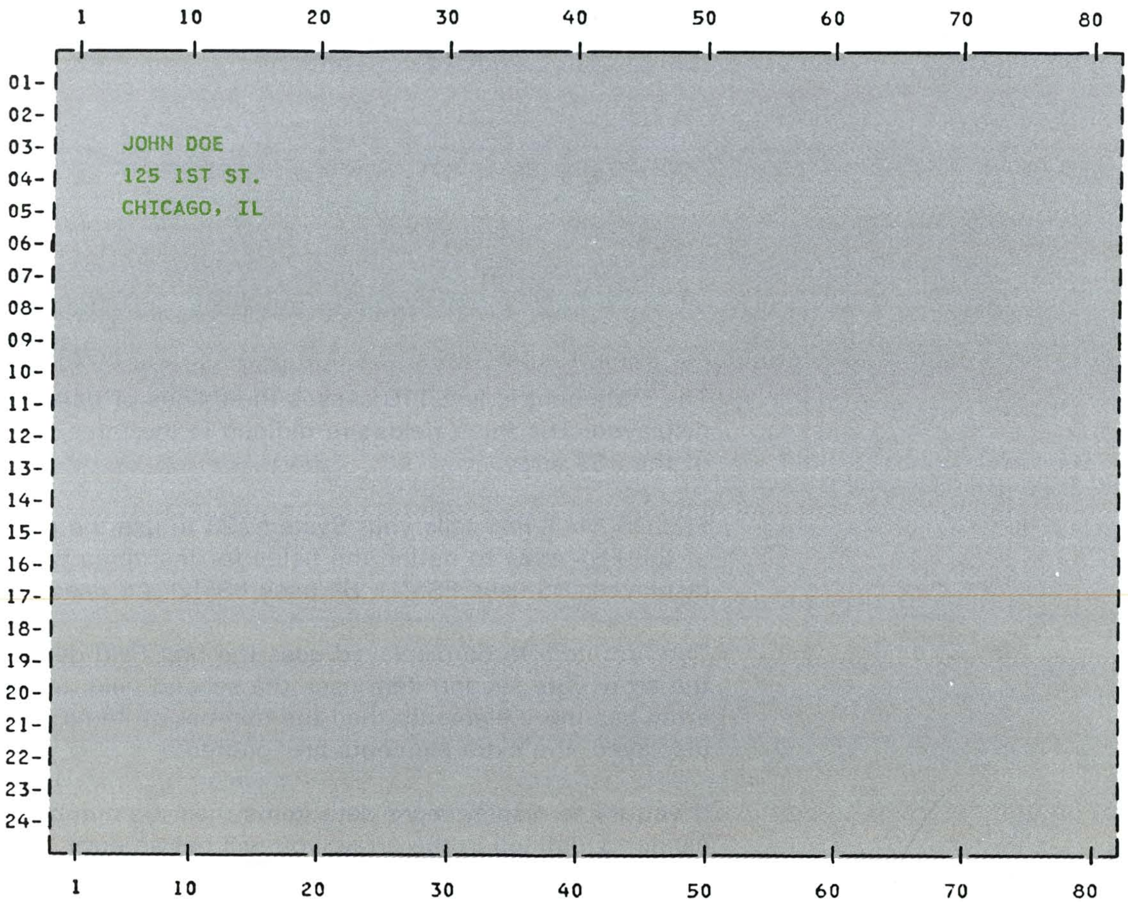
If you try to display more data items than the number of elements defined in the array, you will get an error.

Displaying or inputting several lines of data (continued)

Now run the program:

```
RUN
```

Using our example name and address, the screen should look like this:



You've seen how to display data and how to input data by using full screen processing. Now let's look at a program that does both. Enter the following:

```
CLEAR
10 OPTION BASE 1
20 DIM A$(4),B$(3)
30 A$(1)="5,10,C 8,U,N"
40 A$(2)="10,4,C 5"
50 A$(3)="13,4,C 7"
60 A$(4)="16,4,C 5"
70 PRINT NEWPAGE
80 PRINT FIELDS MAT A$:"EMPLOYEE","NAME:","STREET:","CITY:"
90 B$(1)="10,12,C 18,U,N"
100 B$(2)="13,12,C 18,U,N"
110 B$(3)="16,12,C 18,U,N"
120 INPUT FIELDS MAT B$:NAME$,STREET$,CITY$
130 END
```

You may have noticed that each of our full screen processing examples included a PRINT NEWPAGE. Use this statement to clear the screen before displaying data. Otherwise, the new data will be mixed up with whatever was on the screen before.

When a normal PRINT statement is executed, the data is displayed on line 23. Then lines 2 through 23 are moved up up into lines 1 through 22, and line 23 is cleared. This allows the new data to be entered on line 23.

When a PRINT FIELDS statement is executed, data is displayed on the screen in the specified location, without moving any data that is currently on the screen.

Therefore, when you write a program that uses both PRINT and PRINT FIELDS, you may want to use NEWPAGE to separate non-full screen from full screen processing.

Displaying or inputting several lines of data (continued)

When you run this program, four lines of prompts will be displayed. Remember that a prompt is a word or words on the screen that tell you what input is requested.

For example, NAME:__ tells you to enter your name. STREET:__ tells you to enter your street address.

The cursor appears where the next character will be displayed. When we tell you to run the program, enter your responses in the indicated fields.

Remember to press either the Field Exit key, the New Line key, or the Field Advance key after each input field except the last. Press the Enter key after the last field.

Before we run the program, let's look at one more thing. We have added something to the field definition in line 30.

Position 1 2 3 4 5

```
30 A$(1)="5,10,C 10,U,N"
```

This is different

The U in the fourth position of the field definition tells the System/23 to underline the field. The N in the fifth position returns the screen to normal (no underline) where the field ends. This N is different from an N in the third position:

```
A$(1)= "10,12,N 7.2"
```

The N in the third position indicates a numeric data item. Remember that N 7.2 has up to seven digits, with two digits to the right of the decimal point.

Now run the program:

RUN

When you finish entering your address, your screen should look something like this:

```
         1       10        20        30        40        50        60        70        80
01-
02-
03-
04-
05-
06-           EMPLOYEE
07-
08-
09-
10-
11-      NAME:   JOHN DOE__________
12-
13-
14-      STREET: 125 1ST ST._______
15-
16-
17-      CITY:   CHICAGO, IL_______
18-
19-
20-
21-
22-
23-
24-
         1       10        20        30        40        50        60        70        80
```

Full screen processing

Displaying or inputting several lines of data (continued)

Let's look at one of the input fields.

```
90 B$(1)="10,12,C 18,U,N"
```

Line Column Type of Underline Normal
data

This field begins in line 10, column 12. It will contain a character string of up to 18 characters. The U specifies underline, and the N returns the display to normal.

The first data item entered is NAME$. It is entered from the first field that is defined, B$(1). STREET$ uses B$(2), and CITY$ uses B$(3).

Your turn!

Where does your street address begin on your screen?

Answer: ______________________________

It should begin in line 13, column 12.

Other full screen characteristics, in addition to U and N, are available on your System/23. Refer to ''Full screen processing'' in your *Basic Language Reference* manual for more information.

Chapter summary

Full screen processing allows you to display data and input data in specific areas of the screen. Data items are positioned in fields, which you define in your program.

When you define a field, you must include the line and column in which the field starts. You must also include the type of data item and any special characteristics.

To display data with full screen processing, you enter PRINT FIELDS. To input data with full screen processing, you enter INPUT FIELDS.

To input data with full screen processing, press either the the Field Exit key after each input field on the screen except the last. Press the Enter key after the last input field.

Full screen processing

Exercises

Question 1

What would you enter on line 50 of the following program to display ENTER OPTION NUMBER beginning in line 4, column 12?

```
10 DIM MSG$*19
20 ABC$="4,12,C 19"
30 MSG$="ENTER OPTION NUMBER"
40 PRINT NEWPAGE
60 END
```

Answer: ______________________________

Question 2

What would you enter on line 20 of the following program to display FEBRUARY 1 beginning in line 8, column 6?

```
10 DATE$="FEBRUARY 1"
30 PRINT NEWPAGE
40 PRINT FIELDS FS$:DATE$
50 END
```

Answer: ______________________________

Question 3

Using the numeric variable X, what would you enter on line 30 to be able to input the value 100.05 from line 10, column 2?

```
10 FS$="10,2,N 6.2"
20 PRINT NEWPAGE
40 END
```

Answer: ____________________

Question 4

What will be displayed on line 11 of the screen if you run the following program?

```
10 OPTION BASE 1
20 DIM B$(2)*8
30 DATA "11,3,C 5","11,9,C 6"
40 READ MAT B$
50 D1$="DATE:"
60 D2$="JULY 1"
70 PRINT NEWPAGE
80 PRINT FIELDS MAT B$:D1$,D2$
90 END
```

Answer: ____________________

Exercises (continued)

Question 5

Using the character variable D$, what should you enter on line 60 to be able to input the value JULY 1 from line 5, column 8?

```
10 OPTION BASE 1
20 DIM B$(2)*7
30 READ MAT B$
40 DATA "5,2,C 5","5,8,C 6"
50 PRINT FIELDS B$(1):"DATE:"
70 END
```

Answer: ______________________________

Answers

Question 1

```
50 PRINT FIELDS ABC$:MSG$
```

Question 2

```
20 FS$="8,6,C 10"
```

Question 3

```
30 INPUT FIELDS FS$:X
```

Question 4

```
DATE: JULY 1
```

Question 5

```
60 INPUT FIELDS B$(2):D$
```

Chapter 2. Organizing a program

Introduction

All of the programs you have written in this course have been solutions to problems. Without the programs, your System/23 can't do much more than a pocket calculator can.

In this chapter, you will learn how to use your System/23 and the BASIC language to solve problems. You will see the importance of breaking problems down into manageable parts and then organizing those parts in a program to produce a solution.

One way to organize these parts is to use a *flowchart*. A *flowchart* is a diagram of a solution to a problem. A flowchart can be a very useful tool, because it helps you organize your thoughts in a logical fashion.

Once your thoughts are organized, it is then much easier to write a program.

Objectives

Upon completion of this chapter, you should be able to do the following:

- Identify the standard symbols used in a flowchart.
- Draw a flowchart of a solution to a problem.
- Write a program by translating a flowchart.
- Recognize common programming methods.

If you are familiar with these tasks, try the exercises located at the end of this chapter. If not, read through the chapter before going on to the exercises.

Organizing a program

Solving a problem

You have learned the fundamentals of System/23 BASIC. It is time now to advance from *knowing* BASIC to *using* BASIC to solve problems. The solution to a problem can be broken down into three main parts:

- The *input*, or information required to produce the results. In the accumulated savings program, the input included the principal, the interest rate, and the number of years.

- The *processing*, or manipulation of data to produce the results. This can include initializing variables and performing calculations. Processing turns input into output. In the accumulated savings program, the processing was the calculation of A=P*(1+I)**N.

- The *output*, or results. The primary reason for a program to exist is the output. In the accumulated savings program, the output was the amount of accumulated savings.

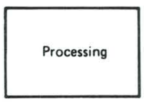
Each of these three parts may consist of one or more statements. And, some short programs may have only one part, such as the output, in:

```
10 PRINT "System/23 BASIC"
20 END
```

In recent years, another form of program organization has been designed. The chart used is called an "N-S" chart or a "star" chart, and it was designed by I. Nassi and B. Schneiderman. For more information, you can refer to "Flowchart Techniques for Structured Programming" in Volume 8 of the *Association for Computing Machines*, August 1973.

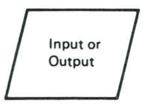
To organize the parts of a program, we use a flowchart. Here are the standard flowcharting symbols we will use in this course. These symbols are common throughout the computer industry.

This is used to show where a program begins or ends.

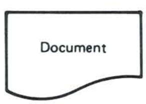


This shows a program instruction to manipulate data or initialize values.

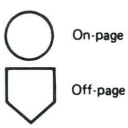
This shows a decision being made. Arrowheads will point in different directions depending on the results.



This shows data to be input or output. It applies to both the display and data files.



This shows printed output, such as a report.



These two symbols show a connection between two separate sections in a flowchart.

This shows direction from one symbol to the next.

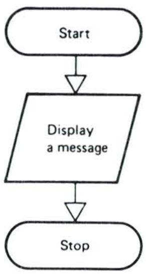
Solving a problem (continued)

Let's look at a few examples of flowcharts. First, consider the first program we wrote in this course. It was a short, simple program to display THIS IS EASY.

If you had been given this problem, how would you solve it? The first thing you do when you solve a problem is break the problem down into manageable parts. In this problem, you are only concerned with output. You want to display a message.

Let's organize the problem in a flowchart. This flowchart is short and simple. It looks like this:

Flowchart



Program

```
10 PRINT "THIS IS EASY"

20 END
```

Some people prefer to draw flowcharts that contain actual program statements and formulas. This flowchart could also look like this:

Flowchart

Program

```
10 PRINT "THIS IS EASY"

20 END
```

You can use whichever method you prefer: general instructions or specific statements. Just make sure that you draw flowcharts that you can understand and use.

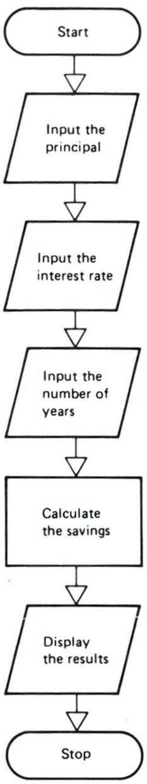
This problem was a simple problem with a simple solution. Most of the problems you will be solving with your System/23 will not be so simple. Let's look at a more complicated example.

Organizing a program

Solving a problem (continued)

Do you remember the accumulated savings problem? This problem has *input* (principal, interest rate, and number of years). It has *output* (accumulated amount). And it has processing (calculation of accumulated amount). What does the flowchart look like for this problem?

Flowchart

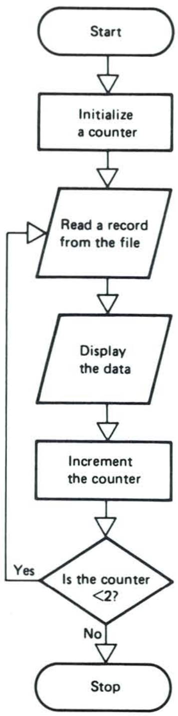


Program

```
10 P=100
   or
10 PRINT "ENTER PRINCIPAL"
15 INPUT P

20 I=.08
   or
20 PRINT "ENTER INTEREST RATE"
25 INPUT I

30 N=2
   or
30 PRINT "ENTER YEARS"
35 INPUT N

40 A=P*(1+I)**N

50 PRINT A

60 END
```

We have shown two programs for this problem. The only difference between them is the method used to input your values. There is no *set* solution to any problem. You must use the statements and methods that seem best to you. We could have written a program that reads a file in two different ways:

Flowchart

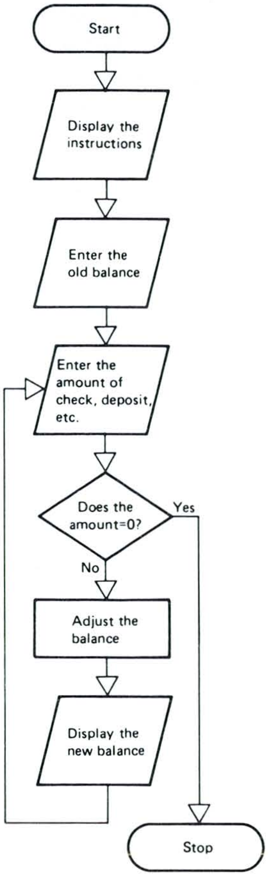


Program

```
10 OPTION BASE 1
20 DIM NAME$*25,ADDRESS$*65
30 OPEN #1:"NAME=CUST",INTERNAL,OUTIN

40 FOR I=1 TO 2
   or
40 I=0

50 READ #1,USING 60:NAME$,ADDRESS$
60 FORM POS 1,C 25,POS 26,C 65


70 PRINT USING 60:NAME$,ADDRESS$


80 NEXT I
   or
80 I= I+1
82 IF I<2 THEN GOTO 50


90 END
```

Organizing a program

Going from a flowchart to a program



Suppose you were asked to write a program to balance a checkbook. Where would you start? The first thing you should do is draw a flowchart.

In the first part of your program you should include a remark that tells what the program does.

Next, you should include instructions for when you run the program. These instructions should be displayed on the screen.

The first data item required is the old account balance.

Now you are ready to adjust the balance for any deposits or withdrawals. Set up a loop to process each transaction.

Display a prompt to enter the amount of each check or deposit or withdrawal.Branch out of the loop when you enter 0 for the amount.

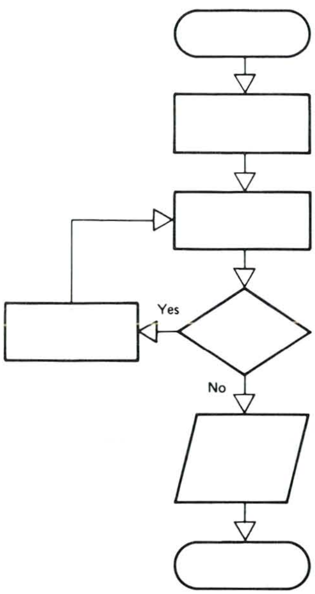
If you enter any amount other than 0, add the amount to the balance.

Display each new balance on the screen, and then ask for the next transaction.

The hard part is done. You have organized each part of the program in a flowchart. Now all you have to do is translate the flowchart into BASIC. One possible solution would be as follows:

```
CLEAR
10 ! PROGRAM TO BALANCE A CHECKBOOK
20 PRINT "ENTER DEPOSITS AS POSITIVE NUMBERS"
30 PRINT "ENTER CHECKS, CHARGES, WITHDRAWALS AS NEGATIVE"
40 PRINT "ENTER AN AMOUNT OF 0 TO END THE PROGRAM"
50 PRINT
60 PRINT "ENTER OLD ACCOUNT BALANCE"
70 INPUT BALANCE
80 PRINT "ENTER CHECK OR DEPOSIT"
90 INPUT X
100 IF X=0 THEN STOP
110 BALANCE=BALANCE+X
120 PRINT "NEW BALANCE= ";BALANCE
130 PRINT
140 GOTO 80
150 END
```

Going from a flowchart to a program (continued)

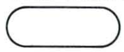


Your turn!

Now we want you to try it. Draw a flowchart for this problem.

Add the odd numbers from 1 through 15. Display the total.

We will help you with this problem by supplying the required flowchart symbols. You will also find helpful notes beside each symbol. Just fill in the instructions in each symbol.

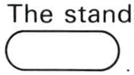
The standard symbol to show the beginning of a program is .

The first number to be added is 1. You should initialize some variable to 1. Use the variable NUMBER.

To add numbers one at a time, you can use the formula SUM=SUM+NUMBER.

You want to keep adding until after NUMBER equals 15. Test the value of NUMBER. If it is less than 15, add 2 to NUMBER and go back to SUM=SUM+NUMBER.

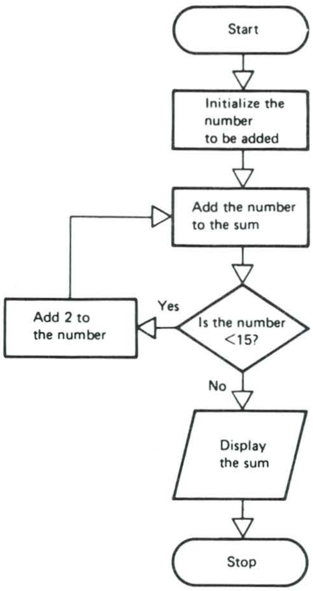
If NUMBER is greater than 15, stop adding and display the sum.

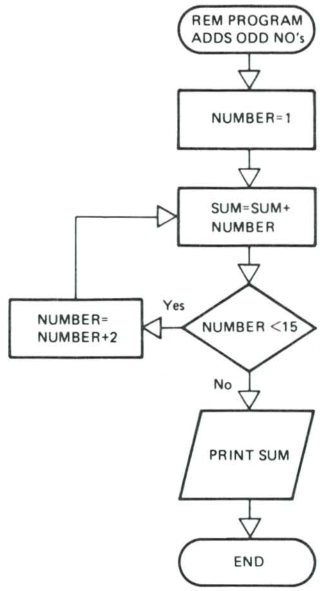
The standard symbol to show the end of a program is .

Going from a flowchart to a program

Here's our solution and a program that performs the indicated instructions.

General instruction flowchart **or** **Specific statement flowchart**






```
10 ! PROGRAM ADDS ODD NUMBERS 1-15
20 NUMBER=1
30 SUM=SUM+NUMBER
40 IF NUMBER>=15 THEN GOTO 70
50 NUMBER=NUMBER+2
60 GOTO 30
70 PRINT SUM
80 END
```

Organizing a program

Inputting data

You have already seen that one problem can have more than one solution. How do you decide which programming method to use in a program? There is no set answer. But here are some suggestions to keep in mind.

Whenever a program requires input data, you need some way to get that data into the computer. The most common methods are:

- Make the data a part of the program. You can do this with LET statements or READ and DATA statements, like this:

  ```
  5 DIM M$(12)
  10 LET X=10
  20 DATA "JANUARY","FEBRUARY","MARCH"
  30 READ M$(1),M$(2),M$(3)
  ```

  Either of these methods can be used when you know the data values when you are writing your program, and the values are not changing.

- Enter the data from the keyboard as you run the program. You can do this with INPUT statements, like this:

  ```
  100 PRINT "ENTER NAME"
  200 INPUT N$
  300 Z$="12,5,C 15"
  400 PRINT FIELDS Z$: "ENTER ADDRESS"
  500 INPUT FIELDS "13,5,C 18,U,N":A$
  ```

  This method can be used when the data may change each time you run the program. Remember to use prompts on the screen to tell you what input is required.

Read the data from a data file as you run the program. You can do this with READ statements, like this:

```
1000 READ #1,USING 50:X,Y,Z
2000 READ #3,REC=4:A$
```

This method can be used when you have a lot of data, and when the same data may be required more than once or by more than one program.

Organizing a program

Outputting data

After a program produces results, you must have some way to access those results. Your program must produce output so you can access those results. Three common methods to output data are:

- Display the results on the screen. You can do this with PRINT statements or PRINT FIELDS statements, like this:

  ```
  100 PRINT "THE NAME IS ";NAME$
  200 PRINT FIELDS MAT A$:X,Y,Z
  ```

  This is a good method when you do not need a copy of the results on paper. The results are lost when the screen is cleared.

- Print the results with your printer. You can do this with PRINT #255 statements, like this:

  ```
  10 PRINT #255:COST,NUMBER,TOTAL
  20 PRINT #255:NEWPAGE,HEADING$
  ```

  This is a good method to use when you want a copy of the results on paper, such as in a report.

- Write the results to a data file. You can do this with WRITE statements, like this:

```
10 WRITE #1:A,B,C
20 WRITE #3,USING 50:N$,A$
```

This is a good method to use when you want to keep the data and update it later. You cannot see the output. Instead, it is stored in a file on your diskette. Some typical uses include customer lists, account balances, and inventories.

Organizing a program

Processing data

The processing part of a program is usually the most complicated part. It includes statements that initialize counters, perform calculations, test values, and direct program control.

Often you need to perform the same job several times in a program. This is especially true when you want to produce a chart or report. It was also true when we wrote the program to balance a checkbook. Here are three common methods to repeat operations in a program:

- Use a function. You can use a function to perform the same calculation on different variables, like this:

  ```
  10 DEF FNT(X)=(1+.06)*X
  20 DEF FNM(X)=(1+.40)*X
  ```

  These functions, or similar functions, can be used to find the cost of an item plus sales tax, or to find the value of an item with a percentage markup.

- Use a FOR/NEXT or IF/THEN loop, like this:

  ```
  10 FOR X=1 TO 90
  20 IF LINES <30 THEN GOTO 50
  30 PRINT #255:NEWPAGE
  40 LINES=0
  50 PRINT #255:A(X),P(X)
  60 LINES=LINES+1
  70 NEXT X
  ```

  This example uses a FOR/NEXT loop to print 90 lines of output. An IF/THEN loop controls paging, allowing only 30 lines per page.

- Use a subroutine. You can use a short program section, like this:

```
10 GOSUB PAGEOUT
20 PRINT A$,A(X)
30 L1=L1+1
.
.
.
700 GOSUB PAGEOUT
710 PRINT B$,B(X)
.
.
.
8000 PAGEOUT: IF L1<30 THEN GOTO 8040
8010 PRINT #255:"PAGE NUMBER ";P1
8020 P1=P1+1
8030 L1=1
8040 RETURN
```

This example controls paging with a simple subroutine. If you place a subroutine near the end of a program, it is easier to trace actual calculations in a program listing.

Here's another example of how you can use a subroutine. The following program reads names and addresses from the CUST file we created in Book VI. It prints the names and addresses on labels with pre-printed return addresses.

This program is designed for 3-inch forms. These forms are designed to be printed with the printer set for six lines per inch. You can order forms from a computer supply store, or you can run the program with plain paper.

If you want to print these labels, enter this program:

Processing data (continued)

```
CLEAR
10 OPEN #1:"NAME=CUST",INTERNAL,INPUT
20 DIM N$*20,S$*20,C$*20
30 FORM POS 1,C 20,C 20,C 20
40 FORM SKIP 8,POS 7,C 20
50 FORM POS 7,C 20
60 FORM POS 7,C 20,SKIP 7
70 PRINT BELL
80 PAUSE
90 PRINT #255:HEX$("2B0205000A1042")
100 FOR X=1 TO 4
110 READ #1,USING 30:N$,S$,C$ IOERR 190
120 GOSUB 150
130 NEXT X
140 STOP
150 PRINT #255,USING 40:N1$
160 PRINT #255,USING 50:S1$
170 PRINT #255,USING 60:C1$
180 RETURN
190 PRINT "ERROR IN DATA FILE"
200 PRINT "FREE CUST FILE AND"
210 PRINT "RERUN PROGRAM TO CREATE FILE"
220 END
```

Before you run this program, notice the IOERR in line 110. If your CUST file was incorrectly entered, you will get an error. If so, free your CUST file and rerun the program that creates the file. Change the paper in your printer, and then enter RUN:

```
RUN
```

When the asterisks appear on the screen, set the paper in the printer to top of form, and enter:

```
GO
```

IBM

P. O. Box 1328
Boca Raton, FL 33432

M05-4110-5

1

First Class
Mail

IBM

P. O. Box 1328
Boca Raton, FL 33432

M05-4110-6

1

First Class
Mail

Organizing a program

Chapter summary

The solution for a problem can be broken down into three main parts:

- The input, or information required to produce the results
- The processing, or manipulation of data to produce the results
- The output, or results

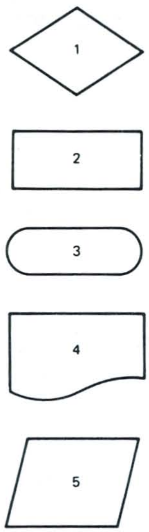
The three parts of a program must be organized to form a solution. A flowchart is a diagram of the solution to a problem.

The most common methods of entering data are to enter the data from the keyboard, to make the data a part of the program, or to read the data from a file.

Processing in a program includes initializing variables, performing calculations, testing values, and directing program control. Some common processing methods for repeating operations include loops, subroutines, and functions

The most common methods of outputting data are to display the data on the screen, to print the data with the printer, or to write the data to a file.

Exercises



Question 1

Match the following flowchart symbols with their meanings.

a. Processing of data

b. Input or output data using display or data file

c. Making a decision

d. The beginning or end of a program

e. Printed output

Question 2

Using three symbols, draw a flowchart for this problem: Print the letter A with your printer.

Answer:

Exercises (continued)

Question 3

Write a program with line numbers 10 and 20 by using the flowchart you drew in Question 2.

Answer: ______________________________

____

Question 4

Match the following programming statements with their common uses. Each letter can be used more than once.

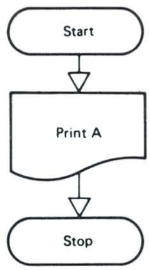
__LET	a. input data
__INPUT	b. output data
__PRINT #255	c. repeat an operation
__FOR and NEXT	
__DEF	
__READ #5	
__IF and THEN	
__WRITE #10	

Answers

Question 1

2 a.
5 b.
1 c.
3 d.
4 e.

Question 2



Answers (continued)

Question 3

```
10 PRINT #255:"A"
20 END
```

Question 4

a LET
a INPUT
b PRINT #255
c FOR and NEXT
c DEF
a READ #5
c IF and THEN
b WRITE #10

Chapter 3. Programming examples

Introduction

In this chapter we present three example programs. These programs use all of the information you have studied in this course. Therefore, you should find them useful for review and future reference.

We think you will find it useful to enter and run these programs. Entering these programs on your System/23 and working with them will give you the practice and experience all beginning programmers need.

The first program is the easiest. The second program builds on the first, and the third program builds on the first two programs and is the most complex. If you enter and run these programs as they are presented, you will have a good foundation to build upon in Chapter 4.

Programming examples

Example 1

Program example 1 records names and addresses and stores them in an array. After you enter the last name and address, a customer information list is printed.

The program uses full screen processing. It displays six prompts for you to enter data. If you enter and run this program, the first screen will look like this:

```
Enter NAME (last name first): ____________________________
               Street Address: ____________________________
               City: ____________________________
               State: ___
               Zip Code: _____
               Telephone Number: ___-____

Input all items using Field Exit before entering.
Press Enter with no input data to print report.
```

You should press the Field Exit key after you enter the name, street address, city, state, and zip code. After you enter the telephone number, press the Enter key.

After you have entered all of your data, press the Enter key again. Then, you will get a formatted printout of all the data stored in the array. Your printout should look something like this (this copy has been reduced):

C U S T O M E R   I N F O R M A T I O N   L I S T

NAME	STREET ADDRESS	CITY	STATE	ZIP CODE	PHONE NUMBER
General Systems	4111 Northside Pkwy	Atlanta	GA	30327	238-2000
Office Products	400 Parsons Pond Dr.	Franklin Lakes	NJ	07417	848-1900
General Business	1133 Westchester Ave	White Plains	NY	10604	696-1900

Programming examples

Example 1 (continued)

```
00010 ! *******          INPUT AND PRINT ARRAY          *******
00020 OPTION BASE 1
00030 DIM NAME$(50)*20,ADDR$(50)*20,CITY$(50)*20,STATE$(50)*3,ZIP$(50)*5
00040 DIM PHONE$(50)*8
00050 DIM L$(8)*9,M$(6)*13,D$(8)*50
00060 DATA "6,5,c 50","7,5,c 50","8,5,c 50","9,5,c 50","10,5,c 50","11,5,c 50"
00070 DATA "14,5,c 50","15,5,c 50"
00080 READ MAT L$
00090 DATA "6,36,c 20,u,n","7,28,c 20,u,n","8,18,c 20,u,n","9,19,c 3,u,n"
00100 DATA "10,22,c 5,u,n","11,30,c 8,u,n"
00110 READ MAT M$
00120 LET D$(1)="Enter NAME (last name first):"
00130 LET D$(2)="      Street Address:"
00140 LET D$(3)="      City:"
00150 LET D$(4)="      State:"
00160 LET D$(5)="      Zip Code:"
00170 LET D$(6)="      Telephone Number:    -"
00180 LET D$(7)="Input all items using Field Exit before entering."
00190 LET D$(8)="Press Enter with no input data to print report."
00200 LET S=17
00210 LET C=0
```

Lines	Description
00010	The first line is a remark. You can change this statement to any comment that will help you identify the program.
00020	OPTION BASE 1 specifies that the lowest array subscript allowed is 1. If you leave this line out, the beginning array subscript is 0.
00030–00050	These lines dimension the arrays for the data items and full screen processing. Notice that these are all *character* arrays.
00060–00080	Line 80 reads the data in lines 60 and 70 into array L$. These describe the fields used in the full screen processing to display the prompts.
00090–00110	Array M$ contains the data to describe the input fields.
00120–00190	Array D$ contains the prompts for full screen processing. Notice that we are using the LET statement to assign values to this array.
00200	S is used to set the left margin for the printed report. You may want to change this value.
00210	The numeric variable C will be used to count the number of customers. Here we initialize it to zero.
	Note: This line is not necessary, since System/23 will automatically initialize variables to zero for you.
	Notice that we have used two different methods to assign values: the LET statement and READ/DATA. These statements are covered in Books I and II.

Example 1 (continued)

```
00220 REM ------------------------------------------ CREATE ARRAY
00230 PRINT NEWPAGE
00240 PRINT FIELDS MAT L$:MAT D$
00250 LET C=C+1
00260 INPUT FIELDS MAT M$:NAME$(C),ADDR$(C),CITY$(C),STATE$(C),ZIP$(C),PHONE$(C)
00270 IF NAME$(C)=RPT$(" ",20) THEN GOTO 300
00280 GOTO 240
```

Lines	Description
00220	This remark tells you that this is the part of the program where you put information into the array.
00230	This line clears the screen. You will want to start with a clear screen before full screen processing.
00240	This line displays the prompts.
00250	Here we add 1 (one) to the number of customers. This number is used as the subscript in the data arrays.
00260	This line inputs data from the screen into the data array elements. Remember that you press the Field Exit key after each input field to get to the next input field. You press the Enter key after all of the data has been entered.
00270	This line checks to see if any more data is to be inputted. If you press the Enter key without entering a name, program control goes to line 300. If you enter a name and other data, the data is stored in the arrays, and program control goes to the next statement, line 280.
00280	This line sends program control back to line 240. There, the prompts are displayed again, and you are ready for more input. You studied the IF-THEN and GOTO statements in Book II. READ MAT is covered in Book IV, and you learned about full screen processing in Chapter 1 of Book VII.

Programming examples

Example 1 (continued)

```
00290 REM ------------------------------------------- PRINT ARRAY
00300 GOSUB OVERFLOW
00310 FOR X=1 TO C-1
00320 PRINT #255:TAB(S);NAME$(X);TAB(S+21);ADDR$(X);TAB(S+42);CITY$(X);
00330 PRINT #255:TAB(S+63);STATE$(X);TAB(S+74);ZIP$(X);TAB(S+88);PHONE$(X)
00340 LET LINE1=LINE1+1
00350 IF LINE1>40 THEN GOSUB 390
00360 NEXT X
```

Lines	Description
00290	This remark statement starts the section that prints the report.
00300	This statement sends control to the OVERFLOW subroutine. Notice that we are using the label OVERFLOW instead of the line number 390.
00310	This statement starts a FOR-NEXT loop. The loop prints the information for the array elements with subscripts of 1 to C-1. C-1 is the number of elements in each array, because C is incremented in line 250, before the value of NAME$ is tested in line 270.
00320–00330	These lines print (on the printer) the information in the arrays. If you do not have a printer, you should leave out the #255: and change the 17 to a 1 in line 200. Or, you can run the program by using the RUN DISPLAY command. Then, the results will be displayed instead of being printed.
00340	This statement keeps track of the number of lines printed on each page. Every time a line is printed, LINE1 is incremented by 1.
00350	This statement sends program control to the OVERFLOW subroutine (line 390), if you print more than 40 lines of information.
00360	NEXT X causes the loop to execute again, until all of the information in the arrays is printed.

You learned about FOR-NEXT loops in Book II. Subroutines are covered in Book V.

Programming examples

Example 1 (continued)

```
00370 STOP
00380 REM -------------------------------- PAGE OVERFLOW ROUTINE
00390 OVERFLOW: PRINT #255:NEWPAGE
00400 PRINT #255:TAB(45);"C U S T O M E R   I N F O R M A T I O N   L I S T"
00410 PRINT #255:
00420 PRINT #255:
00430 PRINT #255:TAB(S);"NAME";TAB(S+21);"STREET ADDRESS";TAB(S+42);"CITY";
00440 PRINT #255:TAB(S+63);"STATE";TAB(S+74);"ZIP CODE";TAB(S+88);"PHONE NUMBER"
00450 PRINT #255:
00460 LET LINE1=0
00470 RETURN
00480 END
```

Lines	Description
00370	This line stops the program after the report.
00380	This remark starts the OVERFLOW subroutine. This subroutine is used to start a new page.
00390	This line skips to a new page on the printer.
00400–00450	These lines print the title and headings at the top of the page. Notice how we use tabs to leave spaces between the columns. This makes the report easier to read.
00460	This statement sets the line counter (LINE1) to 0. LINE1 is used to limit the number of lines on a page to 40.
00470	The RETURN statement sends program control back to line 310 or line 360. This RETURN statement marks the end of the OVERFLOW subroutine.
00480	The END statement tells your System/23 that it has reached the end of your program.

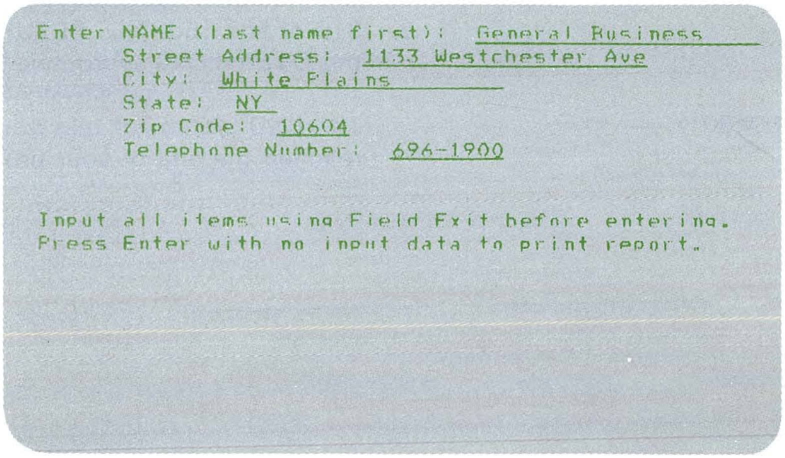
Programming examples

Example 2

Program example 2 is similar to example 1. However, program example 2 creates a file on diskette and stores the data in the file. This program is written to use diskette drive 1. If you are using another diskette drive, you will have to modify your OPEN statement.

Because this program performs almost the same functions as example 1, we will describe only the new lines in example 2. Again, we have numbered the lines in increments of 10. Because example 2 has more lines, the line numbers in this example will not match the line numbers in example 1 exactly.

As in example 1, the following screen is used for data input. It looks like this after you input each data item.



```
00010 ! *******        INPUT, PRINT AND STORE ARRAY        *******
00020 OPTION BASE 1
00030 DIM NAME$(50)*20,ADDR$(50)*20,CITY$(50)*20,STATE$(50)*3,ZIP$(50)*5
00040 DIM PHONE$(50)*8
00050 DIM L$(8)*9,M$(6)*13,D$(8)*50
00060 DATA "6,5,c 50","7,5,c 50","8,5,c 50","9,5,c 50","10,5,c 50","11,5,c 50"
00070 DATA "14,5,c 50","15,5,c 50"
00080 READ MAT L$
00090 DATA "6,36,c 20,u,n","7,28,c 20,u,n","8,18,c 20,u,n","9,19,c 3,u,n"
00100 DATA "10,22,c 5,u,n","11,30,c 8,u,n"
00110 READ MAT M$
00120 LET D$(1)="Enter NAME (last name first):"
00130 LET D$(2)="        Street Address:"
00140 LET D$(3)="        City:"
00150 LET D$(4)="        State:"
00160 LET D$(5)="        Zip Code:"
00170 LET D$(6)="        Telephone Number:      -"
00180 LET D$(7)="Input all items using Field Exit before entering."
00190 LET D$(8)="Press Enter with no input data to print report."
00200 LET S=17
00210 LET C=0
00220 REM ---------------------------------------- CREATE ARRAY
00230 PRINT NEWPAGE
00240 INLOOP: PRINT FIELDS MAT L$:MAT D$
00250 LET C=C+1
00260 INPUT FIELDS MAT M$:NAME$(C),ADDR$(C),CITY$(C),STATE$(C),ZIP$(C),PHONE$(C)
00270 IF NAME$(C)=RPT$(" ",20) THEN GOTO STORE
00280 GOTO INLOOP
```

Programming examples

Example 2 (continued)

```
00290 REM ---------------------------------- STORE DATA ON DISKETTE
00300 STORE: OPEN #1:"NAME= master//1,SIZE= 0,RECL= 100",INTERNAL,OUTPUT
00310 FOR X=1 TO C-1
00320 WRITE #1,USING 330:NAME$(X),ADDR$(X),CITY$(X),STATE$(X),ZIP$(X),PHONE$(X)
00330 FORM 3*C 20,C 3,C 5,C 8
00340 NEXT X
00350 CLOSE #1:
00360 REM ----------------------------------------------- PRINT ARRAY
00370 GOSUB OVERFLOW
00380 FOR X=1 TO C-1
00390 PRINT #255:TAB(S);NAME$(X);TAB(S+21);ADDR$(X);TAB(S+42);CITY$(X);
00400 PRINT #255:TAB(S+63);STATE$(X);TAB(S+74);ZIP$(X);TAB(S+88);PHONE$(X)
00410 LET LINE1=LINE1+1
00420 IF LINE1>40 THEN GOSUB 460
00430 NEXT X
00440 STOP
00450 REM ---------------------------------- PAGE OVERFLOW ROUTINE
00460 OVERFLOW: PRINT #255:NEWPAGE
00470 PRINT #255:TAB(45);"C U S T O M E R   I N F O R M A T I O N   L I S T"
00480 PRINT #255:
00490 PRINT #255:
00500 PRINT #255:TAB(S);"NAME";TAB(S+21);"STREET ADDRESS";TAB(S+42);"CITY";
00510 PRINT #255:TAB(S+63);"STATE";TAB(S+74);"ZIP CODE";TAB(S+88);"PHONE NUMBER"
00520 PRINT #255:
00530 LET LINE1=0
00540 RETURN
00550 END
```

Lines	Description
00290	This remark statement lets you know that this part of the program creates your data file.
00300	This line opens the file called MASTER on diskette drive 1. Remember, if you are using drive 3, you will need to enter MASTER//3. This file is a new, internal file, opened for output only.
00310	This FOR-NEXT loop continues until all of the records are written to the MASTER file.
00320	This line writes the array elements to the file. Notice that the subscripts of the array elements are specified by the variable X, from the FOR statement.
00330	This line tells your System/23 how to write the data to the file. Notice the repetition factor on the C 20 specification.
00340	This NEXT statement completes the loop in lines 310–340.
00350	This statement closes the MASTER file while you are still running the program.

You may want to go back to Book VI at this point to review what you have learned about data files. If not, go on to the next example.

Programming examples

Example 3

Program example 3 adds more features to examples 1 and 2. It is much longer, and you should enter the lines carefully.

Examples 1 and 2 allowed you to do certain tasks, but always in a certain order. Program example 3 allows you to perform additional tasks, and it lets you change the order to suit your needs. You can change the order by using a menu.

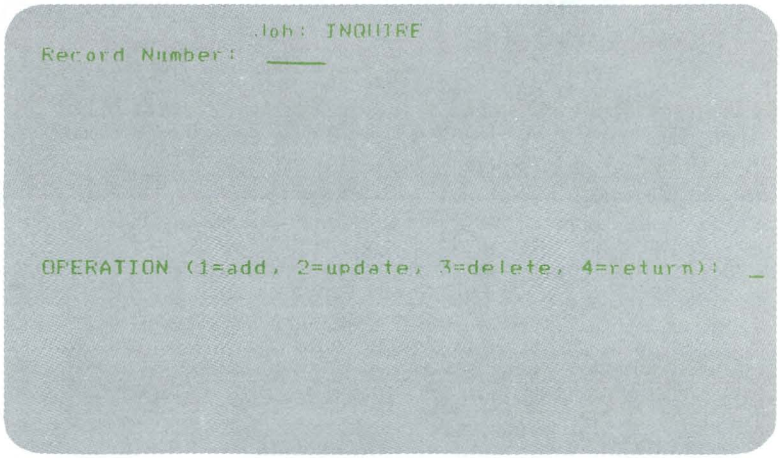
As in example 2, you will be able to create records and store them in a data file. You will also be able to *update*, or change the records you have created.

Example 3 again allows you to create a file called MASTER. So, if you want to run this option, you may first have to free the file you created in example 2.

As with example 2, we will only describe the new lines in this example. The menu will look like this:

```
Job: MAIN MENU

     Options Available:

     1.   Create MASTER File
     2.   Inquire Items
     3.   Print MASTER File
     4.   End Program

     Enter Option No. ?
```

The INQUIRE screen looks like this:



On this screen you enter the number corresponding to the operation you want to perform. For example, if you wanted to add a new record to the file, you would enter a "1" after OPERATION. If you want to update (2) or delete (3) a record, you would also need to enter the relative record number of the record to be changed or deleted.

The screens that allow you to add records or change records are similar to the screen used in examples 1 and 2.

Programming examples

Example 3 (continued)

```
00010 ! *******     INPUT,PRINT,STORE, AND UPDATE FILE     *******
00020 OPTION BASE 1
00030 DIM NAME$(50)*20,ADDR$(50)*20,CITY$(50)*20,STATE$(50)*3,ZIP$(50)*5
00040 DIM PHONE$(50)*8
00050 DIM L$(8)*9,M$(6)*13,D$(8)*50
00060 DIM N$(3)*9,O$(2)*13,A$(6)*48,B$(7)*22,K$(7)*10
00070 DATA "6,5,c 50","7,5,c 50","8,5,c 50","9,5,c 50","10,5,c 50","11,5,c 50"
00080 DATA "14,5,c 52","15,5,c 50"
00090 READ MAT L$
00100 DATA "6,36,c 20,u,n","7,28,c 20,u,n","8,18,c 20,u,n","9,19,c 3,u,n"
00110 DATA "10,22,c 5,u,n","11,30,c 8,u,n"
00120 READ MAT M$
00130 DATA "4,20,c 48","5,5,c 48","14,5,c 48"
00140 READ MAT N$
00150 DATA "5,21,n 4,u,n","14,55,n 1,u,n"
00160 READ MAT O$
00170 DATA "4,20,c 22","6,25,c 22","8,25,c 22","9,25,c 22","10,25,c 22"
00180 DATA "11,25,c 22","13,25,c 22"
00190 READ MAT K$
00200 LET A$(1)="Record Number:"
00210 LET A$(2)="OPERATION (1=add, 2=update, 3=delete, 4=return):"
00220 LET A$(3)="Job: INQUIRE"
00230 LET A$(4)="Job: ADD RECORD"
00240 LET A$(5)="Job: UPDATE RECORD"
00250 LET A$(6)="Job: CREATE MASTER"
00260 LET B$(1)="Job: MAIN MENU"
00270 LET B$(2)="Options Available:"
00280 LET B$(3)="1.  Create MASTER File"
00290 LET B$(4)="2.  Inquire Items"
00300 LET B$(5)="3.  Print MASTER File"
00310 LET B$(6)="4.  End Program"
00320 LET B$(7)="Enter Option No. ?"
```

Lines	Description
00060	This line dimensions more arrays for the menu screen in full screen processing.
00130–00190	Here the data is read into the new arrays for full screen processing. MAT N$ will display the INQUIRE screen, including the job name. MAT O$ is used to input data on the INQUIRE screen, and MAT K$ is used to display the menu.
00200–00250	The first two of these lines are prompts. The next four are displayed on the screen to tell you which job you are performing.
00260–00320	These lines define the B$ array. Array B$ contains the prompts for the menu screen.

Note: You can enter all of lines 200–420 *without* the word LET. But, the word LET will be inserted when you list the program.

Programming examples

Example 3 (continued)

```
00330 LET D$(1)="Enter NAME (last name first):"
00340 LET D$(2)="      Street Address:"
00350 LET D$(3)="      City:"
00360 LET D$(4)="      State:"
00370 LET D$(5)="      Zip Code:"
00380 LET D$(6)="      Telephone Number:    -"
00390 LET D$(7)="Input all items using Field Exit before entering."
00400 LET D$(8)="Press Enter with no data to return to MAIN MENU."
00410 LET S=17
00420 LET C=0
00430 REM ------------------------------------------ DISPLAY MENU
00440 PRINT NEWPAGE
00450 PRINT FIELDS MAT K$:MAT B$
00460 INPUT FIELDS "13,42,n 1":G
00470 ON G GOSUB CREATE,INQUIRE,REPORT,ENDMENU NONE 450
00480 GOTO 440
00490 ENDMENU: PRINT NEWPAGE,"Returned to BASIC"
00500 STOP
```

Lines	Description
00450	This line displays the menu on the screen.
00460	This is the line where you input the number of the job you want. You input the job number in the variable G.
00470	This is a computed ON GOSUB, which you learned about in Book V. The value of G determines which subroutine you use.
00480	The subroutines return to here. Then program control goes back to line 440, where the screen is cleared, and you can enter another job.
00440	This message is displayed before the program stops (line 500). ENDMENU is a label.

Example 3 (continued)

```
00510 REM -------------------------------------------- CREATE ARRAY
00520 CREATE: OPEN #1:"NAME=master//1,SIZE=0,RECL=100",INTERNAL,OUTPUT IOERR 1280
00530 PRINT NEWPAGE
00540 PRINT FIELDS MAT N$:A$(6)
00550 INLOOP: PRINT FIELDS MAT L$:MAT D$
00560 LET C=C+1
00570 INPUT FIELDS MAT M$:NAME$(C),ADDR$(C),CITY$(C),STATE$(C),ZIP$(C),PHONE$(C)
00580 IF NAME$(C)=RPT$(" ",20) THEN GOTO STORE
00590 GOTO INLOOP
00600 REM -------------------------------- STORE DATA ON DISKETTE
00610 STORE: FOR X=1 TO C-1
00620 WRITE #1,USING 630:NAME$(X),ADDR$(X),CITY$(X),STATE$(X),ZIP$(X),PHONE$(X)
00630 FORM 3*C 20,C 3,C 5,C 8
00640 NEXT X
00650 CLOSE #1:
00660 RETURN
00670 REM ------------------------------------------------ INQUIRE
00680 INQUIRE: OPEN #1:"NAME= master//1",INTERNAL,OUTIN,RELATIVE
00690 PRINT NEWPAGE
00700 PRINT FIELDS MAT N$:A$(3),A$(1),A$(2)
00710 INPUT FIELDS MAT O$:Q1,Q2
00720 ON Q2 GOTO ADD,UPDATE,DELETE,ENDINQ NONE 690
00730 ENDINQ: CLOSE #1:
00740 RETURN
```

Lines	Description
00530–00540	These lines display a description of the job being performed. This is the subroutine to create the master file.
00550–00650	These lines are just like examples 1 and 2.
00660	This is where we return from the CREATE subroutine.
00680	This line opens the file MASTER on drive 1. The file is opened with OUTIN, which allows both OUTPUT and INPUT.
	This is the start of the INQUIRE subroutine used to check on a record.
00700	The prompts for operation and record number are displayed.
00710	You input the action you want to perform and the record number, if required.
00720	This is a computed GOTO. The action you want to perform directs program control to specific line numbers.
	Notice the spelling DILETE. Because *DELETE* is a reserved system keyword, we can not use *DELETE* as a label.
00730	You close the file before returning to the MAIN MENU from the subroutine.
00740	This line sends control back to the menu, at line 480.

Programming examples

Example 3 (continued)

```
00750 REM ------------------------------------------- ADD RECORD
00760 ADD: CLOSE #1:
00770 OPEN #1:"NAME= master",INTERNAL,OUTPUT
00780 PRINT NEWPAGE
00790 PRINT FIELDS MAT N$:A$(4)
00800 PRINT FIELDS MAT L$:MAT D$
00810 INPUT FIELDS MAT M$:NAME$(1),ADDR$(1),CITY$(1),STATE$(1),ZIP$(1),PHONE$(1)
00820 IF NAME$(1)=RPT$(" ",20) THEN ENDING
00830 WRITE #1,USING 630:NAME$(1),ADDR$(1),CITY$(1),STATE$(1),ZIP$(1),PHONE$(1)
00840 CLOSE #1:
00850 GOTO INQUIRE
00860 REM ----------------------------------------- UPDATE RECORD
00870 UPDATE: ! READ FILE AND DISPLAY EXISTING DATA
00880 READ #1,USING 630,REC=Q1:NAME$(1),ADDR$(1),CITY$(1),STATE$(1),ZIP$(1),PHONE$(1) NOREC ADD
00890 PRINT FIELDS MAT N$:A$(5)
00900 PRINT FIELDS MAT L$:MAT D$
00910 PRINT FIELDS MAT M$:NAME$(1),ADDR$(1),CITY$(1),STATE$(1),ZIP$(1),PHONE$(1)
00920 INPUT FIELDS MAT M$:NAME$(1),ADDR$(1),CITY$(1),STATE$(1),ZIP$(1),PHONE$(1)
00930 IF NAME$(1)=RPT$(" ",20) THEN ENDING
00940 REWRITE #1,USING 630:NAME$(1),ADDR$(1),CITY$(1),STATE$(1),ZIP$(1),PHONE$(1)
00950 GOTO 690
00960 REM ----------------------------------------- DELETE RECORD
00970 DELETE: DELETE #1,REC=Q1: NOREC 690
00980 GOTO 690
```

Lines	Description
00760–00770	These lines close the file, and then in line 770, the file is opened for the ADD routine with OUTPUT. We need to do this in order to open the file for SEQUENTIAL access, instead of RELATIVE. The reason for this is that, at this point, we don't know the record number of the last record in the file.
00780–00790	These lines display a description of the job being performed. This is the routine to add a record.
00820	If you don't enter any data, we return you th the MAIN MENU instead of writing a blank record to the file. ENDINQ is the label of line 730.
00830	The data is written to the file by using the FORM statement in line 630.
00840–00850	After we've added the record, we close the file and go back to the INQUIRE routine.
00880–00910	The record is read from the file, and the current information is displayed on the screen.
00920	This line accepts any changes you make to the current information.
00930	If you erase the data on the screen, we return you to the MAIN MENU rather than erasing the data in the file.
00940	The record is rewritten with your changes.
00950	This line sends control back to the INQUIRE routine at line 690.
00970–00980	This is the DELETE routine. The specified record is deleted from the file, and control goes back to the INQUIRE routine.

Programming examples

Example 3 (continued)

```
00990 REM ------------------------------------------- PRINT ARRAY
01000 REPORT: OPEN #1:"NAME= master",INTERNAL,INPUT
01010 LET C=1
01020 READ #1,USING 630:NAME$(C),ADDR$(C),CITY$(C),STATE$(C),ZIP$(C),PHONE$(C) EOF CONTPRNT
01030 LET C=C+1
01040 GOTO 1020
01050 CONTPRNT: CLOSE #1:
01060 PRINT NEWPAGE,"Set forms to top of page"
01070 PRINT "Press Enter to continue"
01080 INPUT FIELDS "23,2,c 1":Q$
01090 GOSUB OVERFLOW
01100 FOR X=1 TO C-1
01110 PRINT #255:TAB(S);NAME$(X);TAB(S+21);ADDR$(X);TAB(S+42);CITY$(X);
01120 PRINT #255:TAB(S+63);STATE$(X);TAB(S+74);ZIP$(X);TAB(S+88);PHONE$(X)
01130 LET LINE1=LINE1+1
01140 IF LINE1>40 THEN GOSUB 1180
01150 NEXT X
01160 RETURN
01170 REM --------------------------------- PAGE OVERFLOW ROUTINE
01180 OVERFLOW: PRINT #255:NEWPAGE
01190 PRINT #255:TAB(45);"C U S T O M E R   I N F O R M A T I O N   L I S T"
01200 PRINT #255:
01210 PRINT #255:
01220 PRINT #255:TAB(S);"NAME";TAB(S+21);"STREET ADDRESS";TAB(S+42);"CITY";
01230 PRINT #255:TAB(S+63);"STATE";TAB(S+74);"ZIP CODE";TAB(S+88);"PHONE NUMBER"
01240 PRINT #255:
01250 LET LINE1=0
01260 RETURN
01270 REM ---------------------------------------------- I/O ERROR
01280 PRINT NEWPAGE,"I/O Error has occurred.  Program terminated!"
01290 PRINT "If creating MASTER file, enter 'FREE MASTER' and rerun program."
01300 END
```

Lines	Description
00100	For the report, we open the file for input. This means that we are only going to *read* records from the file.
00102–01040	This loop reads all of the records from the file into the data arrays and counts the number of records read. When the last record is read, control goes to line 1050 (EOF CONTPRNT).
01050	The file is closed after the last record is read.
01060–01080	These lines display a message to set top of forms. The dummy variable Q$ accepts the entry when you press the Enter key.
01090–01150	You saw these lines in Examples 1 and 2. They are used to print the reports and control paging.
01160	This line ends the PRINT subroutine and sends control back to the MAIN MENU.
01280–01290	These lines display a message if an error occurs when you create the file in line 520.

You have seen three different programs that print the same report. In the next chapter, we will show you another example program, which is even more complex.

Chapter 4. Example inventory program

Introduction

In this chapter, we will present an example inventory program. You have studied, in Books I-VII, each of the statements and programming methods used in this program. Therefore, you should find this program useful for review and future reference.

You are not required to enter this program on your System/23. However, we think you will find it helpful. Entering the program and working with it will give you additional practice, something every beginning programmer needs.

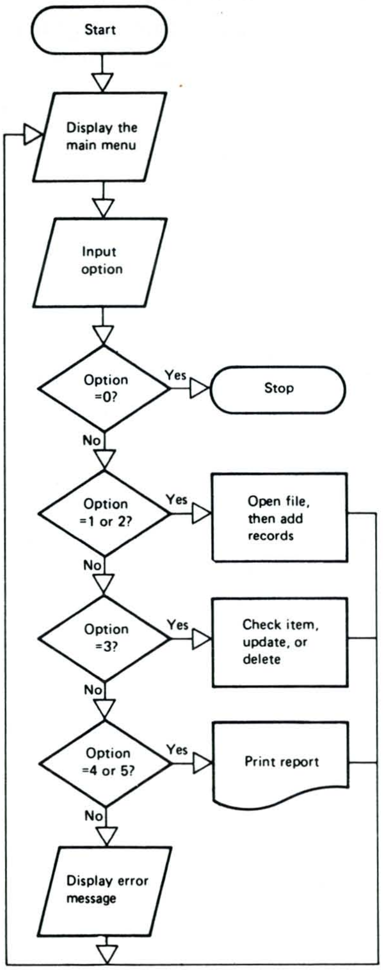
You can change parts of this program for your use. For example, you may want to enter the name of *your* company in line 790. You may also want to change line 880 to print more lines on each page.

We do not expect you, at this point, to be able to write a program as complex as this example. We are including this program to show you what you can do with your System/23 after more study and more practice, practice, practice.

A flowchart is included for this program. Also, you will find pages of notes describing each section of the program.

Example inventory program

Flowchart



Entire program

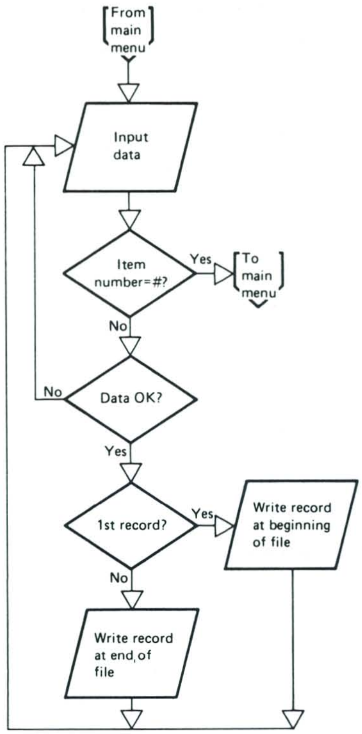
This flowchart is an overall view of the entire program. The program offers five options, plus sign off:

0. Sign off
1. Create master inventory file
2. Add items to master file
3. Inquire items (also update and delete)
4. Status report--all items
5. Status report--zero quantity items

This program uses full screen processing. Messages will be displayed on the screen if you enter a wrong answer. The data stored for each item in the file includes item number, description, unit cost, and quantity on hand.

We will show flowcharts on the next three pages that detail the program options. As you can see from this general flowchart, you can return to the *Main Menu* after each option. The first symbol in this flow chart represents a display of the program options.

Create a file or add records



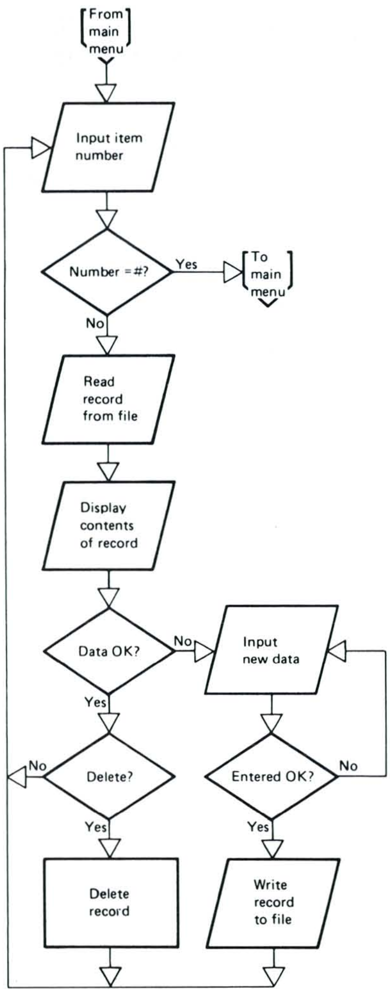
This flowchart is designed for menu options 1 and 2. It shows what happens when you create the inventory file or add records to the file.

The first symbol in this flow chart represents the Main Menu. When you enter # for an item number, the program branches back to the Main Menu.

You enter an item number, description, unit cost, and quantity on hand. If you enter any data incorrectly, you are given a chance to correct it.

Each item is written to a data file. If you are creating the file with option 1, the OPEN statement will include the size of the file.

Flowchart (continued)



Inquire about, update, or delete records

This flowchart is designed for menu option 3. It shows what happens when you check on, update, or delete a record in your file.

The first symbol represents the Main Menu. When you enter # for an item number, the program branches back to the Main Menu.

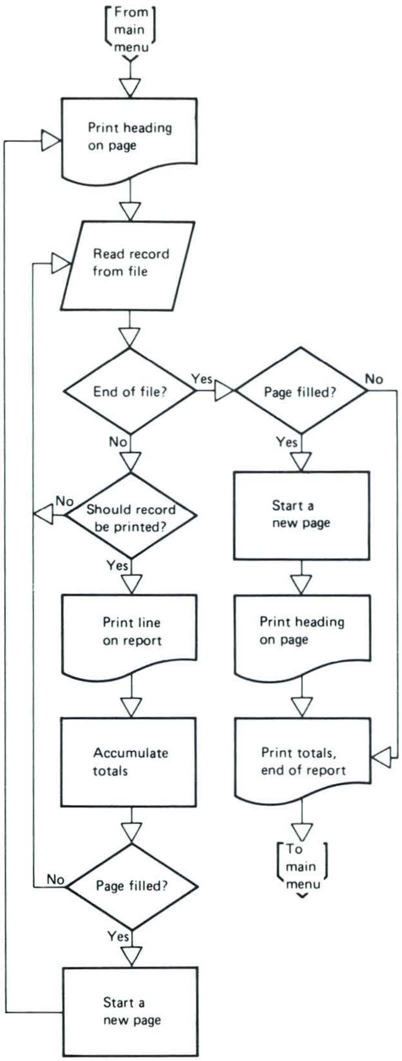
When you enter an item number, the file is read sequentially until the record is found. We could have used a key-indexed file, but we didn't want to interrupt the program to run the INDEX Customer Support Function.

Records are read from the file, and the data is displayed on your screen. If something needs to be changed, you enter the correction, and the record is rewritten to the file.

If you want to delete the record, it is deleted from the file.

After checking on, updating, or deleting a record, you go on to check on, update, or delete another record.

Print reports



This flowchart is designed for menu options 4 and 5. It shows what happens when you print the inventory status reports.

The first symbol represents the Main Menu. At the end of the report, the program branches back to the Main Menu.

The program uses subroutines to print the headings and to advance the paper to a new page.

Records are read from the file in sequential order. When you reach the end of the file, you are ready to print the totals.

The data for all items will be printed when you enter option 4. The data for only zero-quantity items will be printed when you enter option 5.

Example inventory program

Program

```
00010 ! * EXAMPLE INVENTORY PROGRAM *
00020 ! *      FOR BOOK VII OF      *
00030 ! * LEARNING SYSTEM/23 BASIC  *
00040 OPTION BASE 1
00050 ON ERROR GOTO ERREND
00060 ! -----------------------------------------------------
00070 ! INITIALIZATION
00080 ! -----------------------------------------------------
00090 ! ------------------------------------------------ DIMS
00100 DIM DSPFSP$(5)*25,INPFSP$(4)*15
00110 DIM MSGFSP$(5)*10
00120 DIM QTYFSP$(4)*15,SCRFSP$(8)*14,TTLFSP$(4)*10
00130 DIM FLDNM$(5)*12,MENU$(8)*38,MSG$(19)*58
00140 DIM STATRPT$(4)*95
00150 DIM ANS$*1,AP$*39,DESCR$*20,ITEM$*5,ITEM1$*5
00160 DIM OPTNAME$*35
```

Lines	Description
00010-00030	The first three lines are used for remarks. You can change these statements to any comments that will help you identify the program.
00040	OPTION BASE 1 specifies that the lowest array subscript allowed is 1.
00050	If an error occurs anywhere in the program, control goes to ERREND (line 2800), and the program ends.
00060-00090	These lines are remark statements. They are used to separate sections of the program. They also tell you what the different sections do in the program.
00100-00120	These lines dimension the arrays that will be used for full screen processing.
00130-00160	These lines dimension the data arrays and tell how long the character variables can be.

Example inventory program

Program (continued)

```
00170 ! -------------------------------------------- FSP DATA
00180 DATA "10,38,c 20,n","11,38,n 7.2,n"
00190 DATA "12,38,n  6,n","13,38,pic($$$,$$$,$$#.##)"
00200 DATA " 8,38,c  5,n"
00210 READ MAT DSPFSP$
00220 !
00230 DATA "10,38,c 20,u ,n","11,38,n  7,u ,n"
00240 DATA "12,38,n  6,u ,n"," 8,38,c  5,u ,n"
00250 READ MAT INPFSP$
00260 !
00270 DATA "17,12,c 53","18,12,c 51","19,24,c 27"
00280 DATA "20,24,c 39","21,24,c 44"
00290 READ MAT MSGFSP$
00300 !
00310 DATA "12,24,c 12","12,38,n  6,u ,n","12,54,c  8"
00320 DATA "12,63,n  6"
00330 READ MAT QTYFSP$
00340 !
00350 DATA "10,24,c 34","11,24,c 38","12,24,c 45"
00360 DATA "13,24,c 38"," 8,24,c 19"," 9,24,c 35"
00370 DATA " 6,24,c 18"," 3,24,c 35,h,n"
00380 READ MAT SCRFSP$
00390 !
00400 DATA " 1,10,c 39"," 1,58,c  5"," 1,64,c  8"
00410 DATA " 3,18,c  4"
00420 READ MAT TTLFSP$
```

Lines	Description
00180-00210	The DSPFSP$ array contains the fields that display the data in a record during INQUIRY.
00230-00250	The INPFSP$ array contains the fields that input data during CREATE/ADD and INQUIRY.
00270-00290	The MSGFSP$ array contains the fields that display the instructions and error messages at the bottom of the screen.
00310-00330	The QTYFSP$ array contains the fields that rewrite the quantity line during UPDATE.
00350-00380	The SCRFSP$ array displays the prompts for the MAIN MENU, INQUIRY, and ADD/CREATE.
00400-00420	The TTLFSP$ array displays the headings at the top of the screen.

Example inventory program

Program (continued)

```
00430 ! ------------------------------------------- ARRAY DATA
00440 DATA "Description:","  Unit Cost:","Qty on Hand:"
00450 DATA "Total Value:","Item Number:"
00460 READ MAT FLDNM$
00470 !
00480 DATA 2. ADD ITEMS TO MASTER FILE
00490 DATA 3. INQUIRE ITEMS (+ UPDATE & DELETE)
00500 DATA 4. STATUS REPORT - ALL ITEMS
00510 DATA 5. STATUS REPORT - ZERO QUANTITY ITEMS
00520 DATA 0. SIGN OFF,"1. * CREATE MASTER INVENTORY FILE *"
00530 DATA "Options available:",MAIN MENU
00540 READ MAT MENU$
00550 !
00560 LET MSG$(1)="Enter option number      and press Enter"
00570 LET MSG$(2)="Enter:       Item Number (or ÷ to return "
00580 LET MSG$(2)=MSG$(2)&"to MAIN MENU)"
00590 LET MSG$(3)=RPT$(" ",12)&"Description of Item"
00600 LET MSG$(4)="Unit Cost"
00610 LET MSG$(5)="Quantity on Hand"
00620 LET MSG$(6)="Qty on Hand - Enter received qty (+) or"
00630 LET MSG$(7)=RPT$(" ",14)&"sold qty (-) or 0 if no change"
00640 LET MSG$(8)="May change:"
00650 LET MSG$(9)="Is everything okay?"
00660 LET MSG$(10)="Is Item okay?"
00670 LET MSG$(11)="Y = Yes"
00680 LET MSG$(12)="N = No"
00690 LET MSG$(13)="D = Delete Item"
00700 LET MSG$(14)="** INVALID ENTRY - REENTER **"
00710 LET MSG$(15)="Field must be all numbers"
00720 LET MSG$(16)="New Qty makes Qty on Hand ( 0 or ) 99999"
00730 LET MSG$(17)="Unit Cost is ( 0 or ) 9999.99"
00740 LET MSG$(18)="** ITEM NOT FOUND **"
00750 LET MSG$(19)="Enter new Item Number (or ÷ to return "
00760 LET MSG$(19)=MSG$(19)&"to MAIN MENU)"
```

Lines	Description
00440-00460	The FLDNM$ array contains the prompts, which are displayed by using the SCRFSP$ fields.
00480-00540	The MENU$ array contains the list of menu options, which are displayed by using the SCRFSP$ fields.
00560-00760	The MSG$ array contains all of the instructions and error messages, which are displayed by using the MSGFSP$ fields. Notice that we are assigning values to the MSG$ array by using LET statements. The LET statements allow easy reference when tracing a program listing. All of the arrays prior to MSG$ were assigned values by using READ/DATA statements. Because we are using BASE 1 in our OPTION statement, the first element in each of these arrays has a subscript of (1). Notice lines 590 and 630. We are using the RPT$ function, which repeats the blank string " " either 12 or 14 times. We use this technique to line up the messages on the screen. Also notice that several lines in this section contain the & sign. Remember that this is how you join two strings together.

Example inventory program

Program (continued)

```
00770 !
00780 LET STATRPT$(1)="+"&RPT$("-",93)&"+"
00790 LET STATRPT$(2)="|"&RPT$(" ",15)&"W O R L D  W I D E  M A N U F A C T U R I N G  C O.,  L T D."&RPT$(" ",15)&"|"
00800 LET STATRPT$(3)=STATRPT$(1)
00810 LET STATRPT$(3)(15:15)="+"
00820 LET STATRPT$(3)(38:38)="+"
00830 LET STATRPT$(3)(52:52)="+"
00840 LET STATRPT$(3)(64:64)="+"
00850 LET STATRPT$(4)="| Item Number | Description          | Qty on Hand | Unit Cost | Total Value of On-Hand Items |"
00860 ! ---------------------------------------- SCALAR DATA
00870 LET AP$="Application:  EXAMPLE INVENTORY PROGRAM"
00880 LET PAGELEN=5
00890 ! ------------------------------------ FORM STATEMENTS
00900 DATEFORM: FORM "| Date: ",C 8,POS 35,C 35,POS 84,"Page: ",N 3,"  |"
00910 FILEFORM: FORM C 5,C 20,N 7.2,N 5,N 11.2
00920 RPTFORM: FORM "|",POS 8,C 5,X 4,C 20,POS 45,PIC(ZZ,ZZ#),X 3,PIC($$,$$#.##),POS 80,PIC($$$,$$$,$$#.##)," |"
00930 TOTFORM: FORM "|",POS 27,"TOTALS",POS 41,PIC(ZZ,ZZZ,ZZ#),POS 76,PIC($$$,$$$,$$$,$$#.##)," |"
00940 ! --------------------------- UNIVERSAL EXIT STATEMENT
00950 ERREX: EXIT  EOF ERREND,CONV ERREND,SOFLOW ERREND
```

Lines	Description
00780-00850	The STATRPT$ array contains some lines that are printed on the status reports. Notice that we are using the & sign again. Also, look at lines 790 and 850. These statements have been printed on more than one line, because the statements are longer than 80 characters.
00870	AP$ is the name of the application. The name is displayed by using the TTLFSP$ fields.
00880	PAGELEN is the number of lines printed on a report page. You may want to change this number.
00900-00930	Lines 900, 920, and 930 are FORM statements that are used to print the reports. The first word on each of these lines is a statement label. Line 910 (FILEFORM) is used to read data from and write data to the ITEM file.
00950	This EXIT statement contains error conditions that should never happen. If any of them does happen, control goes to line 2800.

Program (continued)

```
00960 ! --------------------------------------------------------
00970 ! BEGIN PROCESSING - BRING UP MAIN MENU
00980 ! --------------------------------------------------------
00990 START: PRINT NEWPAGE
01000 PRINT FIELDS MAT TILFSP$:AP$,"Date:",DATE$,"Job:"
01010 PRINT FIELDS MAT SCRFSP$:MAT MENU$
01020 PRINT FIELDS MSGFSP$(1):MSG$(1)
01030 INPUT FIELDS "17,33,n 1,h,n":OPT CONV 1050
01040 IF OPT>=0 AND OPT<6 THEN 1070
01050 ! ERROR
01060 GOTO 1030
01070 IF OPT<2 THEN LET OPTNAME$=MENU$(OPT+5)(4:38) ELSE LET OPTNAME$=MENU$(OPT-1)(4:38)
01080 ON OPT GOTO NEW,ADD,INQ,REPORT,REPORT NONE FIN
01090 ! --------------------------------------------------------
01100 ! ADD OR CREATE - MENU OPTIONS 1 AND 2
01110 ! --------------------------------------------------------
01120 NEW: OPEN #1:"NAME= item.master//1,SIZE= 512,RECL= 63",INTERNAL,OUTPUT
01130 GOTO AD1
01140 ADD: OPEN #1:"NAME= item.master",INTERNAL,OUTPUT
01150 ! ------------------------------ PUT UP INPUT SCREEN
01160 AD1: PRINT FIELDS MAT SCRFSP$:MAT FLDNM$,"","",OPTNAME$
01170 AD1NEXT: PRINT FIELDS MAT INPFSP$:"",0,0,"$"
01180 THERE: PRINT FIELDS MAT MSGFSP$:MSG$(2),MSG$(3),MSG$(4),MSG$(5),"" ! DIRECTIONS
01190 LET INPFSP$(4)(13:13)="c" ! PUT CURSOR IN ITEM$ FIELD
```

Lines	Description
00990-01020	These lines clear the screen and display the Main Menu.
01030	This line enters the option number, beginning in line 17, column 33. Notice that the input field is highlighted.
01040-01070	These lines test OPT to see if it is a valid entry for the option. If the entry is invalid, return to line 1030. Then reenter the option number. If OPT is a valid entry, OPTNAME$ becomes the job title. The job title is found in the MENU$ array. Notice that we use character positions in line 1070 to find the correct title.
01080	Using a computed ON-GOTO, we direct program control to the correct section for each option.
01120-01140	These lines open the file ITEM.MASTER. Two different statements are required, because the OPEN statement differs for old and new files.
01160-01190	These lines display the prompts, as well as a default value for each entry. Then the cursor is placed in the first field, where you enter the item number.

Example inventory program

Program (continued)

```
01200 ! ---------------------------------------- INPUT DATA
01210 HERE: INPUT FIELDS MAT INPFSP$:DESCR$,CUST,QTY,ITEM$ CONV INPCONV
01220 IF ITEM$(1:1)="#" THEN ADEND ! END OF INPUT
01230 IF COST>=0 AND COST<10000 THEN TQ
01240 ! ERROR IN COST
01250 PRINT FIELDS MAT MSGFSP$:MSG$(14),MSG$(17),"","",""
01260 LET CURSPARM=2
01270 GOSUB CURSPOS
01280 GOTO HERE
01290 TQ: IF QTY>=0 AND QTY<100000 THEN AD2
01300 ! ERROR IN QUANTITY
01310 PRINT FIELDS MAT MSGFSP$:MSG$(14),MSG$(16),"","",""
01320 LET CURSPARM=3
01330 GOSUB CURSPOS
01340 GOTO HERE
01350 ! ------------------------------------ EVERYTHING OK?
01360 AD2: PRINT FIELDS DSPFSP$(4):QTY*COST
01370 PRINT FIELDS MAT MSGFSP$:MSG$(9),"",MSG$(11),MSG$(12),""
01380 INPUT FIELDS "17,33,c 1,h,n":ANS$
01390 ON POS ("YyNn",ANS$,1) GOTO WRITEIT,WRITEIT,THERE,THERE NONE 1400
01400 ! ERROR
01410 GOTO 1380
01420 ! -------------------------------------- WRITE RECORD
01430 WRITEIT: WRITE #1,USING FILEFORM:ITEM$,DESCR$,COST,QTY,QTY*COST EXIT ERREX
01440 PRINT FIELDS SCRFSP$(4):FLDNM$(4) ! ERASE TOTAL VALUE
01450 GOTO AD1NEXT
01460 ! ---------------------------- END OF ADD/CREATE INPUT
01470 ADEND: CLOSE #1:
01480 GOTO START
```

Lines	Description
01210	This line inputs each field of data. If a conversion error occurs, go to statement INPCONV (line 2620).
01220-01280	If you enter an item number of #, go to statement ADEND (line 1470). If the cost is incorrect, display an error message, position the cursor, and reenter. Otherwise, go on to check the quantity.
01290-01340	If the quantity is entered correctly, go to statement AD2 (line 1360). If not, display an error message, place the cursor in the quantity field, and reenter.
01360-01370	Compute and display the total value for the item.
01380	Ask if everything is okay, and input the answer. This allows you to check your entries.
01390	If everything is okay, go to statement WRITEIT (line 1430), and write the data to the file. If it's not okay, go to THERE (line 1180) so everything can be reentered.
01400-01410	The question was answered incorrectly, so answer the question again.
01430-01440	Write the data to the file. Go back to statement AD1NEXT (line 1170), and enter another time.
01470-01480	You entered an item number of #, which indicates the end of data entry. Close the file and return to the Main Menu (line 990).

Example inventory program

Program (continued)

```
01490 ! -----------------------------------------------------
01500 ! INQUIRE - OPTION 3
01510 ! -----------------------------------------------------
01520 INQ: OPEN #1:"NAME= item.master",INTERNAL,OUTIN
01530 IN1: PRINT FIELDS MAT SCRFSP$:"","","","",FLDNM$(5),"","",OPTNAME$ ! INITIAL PROMPT
01540 PRINT FIELDS MAT MSGFSP$:MSG$(2),"","","","" ! DIRECTIONS
01550 PRINT FIELDS INPFSP$(4):"#" ! DEFAULT
01560 ! -------------------- GET ITEM NUMBER AND FIND RECORD
01570 INQINP: INPUT FIELDS INPFSP$(4):ITEM1$
01580 IF ITEM1$(1:1)="#" THEN INEND ! END OF INQUIRY
01590 ! SEARCH FILE FOR ITEM NUMBER
01600 ITEST: RESTORE #1: ! START SEARCH AT BEGINNING OF FILE
01610 READ #1,USING FILEFORM:ITEM$,DESCR$,COST,QTY,TOTAL EOF NREKR
01620 IF ITEM$)(ITEM1$ THEN 1610
01630 ! ----------------------- DISPLAY DATA FROM FOUND ITEM
01640 IN2: PRINT FIELDS MAT SCRFSP$:MAT FLDNM$ ! PROMPTS
01650 PRINT FIELDS MAT DSPFSP$:DESCR$,COST,QTY,TOTAL,ITEM$
01660 ! ------------------------------------------- ITEM OK?
01670 PRINT FIELDS MAT MSGFSP$:MSG$(10),"",MSG$(11),MSG$(12),MSG$(13) ! DIRECTIONS
01680 INPUT FIELDS "17,27,c 1,h,n":ANS$
01690 ON POS ("YyNnDd",ANS$,1) GOTO IN1,IN1,IN3,IN3,INDEL,INDEL NONE 1700
01700 ! ERROR
01710 GOTO 1680
01720 ! ---------------------------------------- DELETE ITEM
01730 INDEL: DELETE #1:
01740 GOTO IN1
```

Lines	Description
01520	Open the file for INQUIRY. Since the file should already exist, don't state RECL or SIZE.
01530-01540	Display the prompts and instructions for INQUIRY.
01570	Enter an item number.
01580	If the record number is #, the INQUIRY is finished. Go to statement INEND (line 2060).
01600-01620	Read the file sequentially until you find the correct item number. We could find the record quicker with a key-indexed file, but we didn't want to run the INDEX Customer Support Function.
01640-01670	Display the data and ask if everything is okay.
01680	Input the answer.
01690	If everything is okay, go to statement IN1 (line 1530) to ask for another item. If not, go to IN3 (line 1760) to update item. If you want to delete the record, go to INDEL (line 1730).
01700-01710	The question was answered incorrectly, so answer the question again.
01730-01740	Delete the item (record just read), and go back for the next item.

Example inventory program

Program (continued)

```
01750 ! ------------------------------------------ CHANGE ITEM
01760 IN3: PRINT FIELDS MAT QTYFSP$:"   Quantity:",0,"On Hand:",QTY ! REWRITE QUANTITY LINE FOR INPUT
01770 PRINT FIELDS MAT MSGFSP$:MSG$(8),MSG$(3),MSG$(4),MSG$(6),MSG$(7) ! DIRECTIONS
01780 LET CURSPARM=1
01790 GOSUB CURSPOS ! POSITION CURSOR TO 1ST FIELD
01800 HEREIN: INPUT FIELDS MAT INPFSP$:DESCR$,COST,NEWQTY CONV INPCONV
01810 IF COST>=0 AND COST<10000 THEN TESTQIN
01820 ! ERROR IN COST
01830 PRINT FIELDS MAT MSGFSP$:MSG$(14),MSG$(17),"","",""
01840 LET CURSPARM=2
01850 GOSUB CURSPOS
01860 GOTO HEREIN
01870 TESTQIN: IF QTY+NEWQTY>=0 AND QTY+NEWQTY<100000 THEN IN4
01880 ! ERROR IN QUANTITY
01890 PRINT FIELDS MAT MSGFSP$:MSG$(14),MSG$(16),"","",""
01900 LET CURSPARM=3
01910 GOSUB CURSPOS
01920 GOTO HEREIN
01930 ! ------------------------------------- EVERYTHING OK?
01940 IN4: LET QTY=QTY+NEWQTY
01950 PRINT FIELDS SCRFSP$(3):FLDNM$(3) ! REWRITE PROMPT
01960 PRINT FIELDS MAT DSPFSP$:DESCR$,COST,QTY,COST*QTY
01970 PRINT FIELDS MAT MSGFSP$:MSG$(9),"",MSG$(11),MSG$(12),"" ! DIRECTIONS
01980 INPUT FIELDS "17,33,c 1,h,n":ANS$
01990 ON POS ("YyNn",ANS$,1) GOTO INWRITE,INWRITE,IN3,IN3 NONE 2000
02000 ! ERROR
02010 GOTO 1980
02020 ! ------------------------------ WRITE CHANGED RECORD
02030 INWRITE: REWRITE #1,USING FILEFORM:ITEM$,DESCR$,COST,QTY,QTY*COST EXIT ERREX
02040 GOTO IN1
02050 ! ------------------------------------- END OF INQUIRY
02060 INEND: CLOSE #1:
02070 GOTO START
```

Lines	Description
01760-01770	Display prompts and instructions for UPDATE.
01780-01800	Position the cursor and enter new data for the item.
01810	If the cost is okay, go on to test the quantity(line 1870).
01820-01860	If the cost is incorrect, display an error message. Place the cursor in the cost field and reenter. HEREIN (line 1800), is where you go to reenter your data.
01870	Test the new quantity. If it's okay, go to statement IN4 (line 1940).
01890-01920	The new quantity is too large or too small. Display an error message, and enter new data.
01940-01970	Display the new data and total value. Ask if everything is okay.
01980	Answer the question.
01990	If everything is okay, go to statement INWRITE (line 2030). If not, go to IN3 (line 1760) to correct the information.
02000-02010	The question was answered incorrectly, so answer the question again.
02030-02040	Rewrite the record with the new data, and go back for the next item.
02060-02070	You entered a record number of #, which indicates the end of INQUIRY. Close the file and return to the Main Menu.

Example inventory program

Program (continued)

```
02080 ! -----------------------------------------------------------
02090 ! REPORTS - OPTIONS 4 AND 5
02100 ! -----------------------------------------------------------
02110 REPORT: PRINT NEWPAGE,"Report in progress - please wait."
02120 LET TOTQTY=0
02130 LET TOTAMT=0
02140 LET PAGENO=0
02150 OPEN #1:"NAME= item.master",INTERNAL,INPUT
02160 GOSUB PAGEHEAD
02170 ! ------------------- READ FILE AND PRINT DETAIL LINES
02180 LOOP: FOR LINENO=1 TO PAGELEN
02190 READ #1,USING FILEFORM:ITEM$,DESCR$,COST,QTY,TOTAL EOF RPTEND
02200 IF OPT=5 AND QTY><0 THEN 2190
02210 PRINT #255,USING RPTFORM:ITEM$,DESCR$,QTY,COST,TOTAL
02220 LET TOTQTY=TOTQTY+QTY
02230 LET TOTAMT=TOTAMT+TOTAL
02240 NEXT LINENO
02250 GOSUB PAGESKIP
02260 GOTO LOOP
02270 ! ------------------------ END OF REPORT - PRINT TOTALS
02280 RPTEND: IF LINENO>PAGELEN-2 THEN GOSUB PAGESKIP
02290 IF LINENO=1 THEN 2330
02300 ! NOT START OF NEW PAGE SO PRINT DIVIDING LINES
02310 PRINT #255:"|",TAB(41);"----------";TAB(77);RPT$("-",17);" |"
02320 LET LINENO=LINENO+1
02330 PRINT #255:"|",TAB(95);"|" ! SKIP A LINE
02340 PRINT #255,USING TOTFORM:TOTQTY,TOTAMT
02350 LET LINENO=LINENO+2
02360 GOSUB PAGEND
02370 CLOSE #1:
02380 GOTO START
```

Lines	Description
02110	Display a message before starting reports. Notice that this statement is not full screen processing.
02120-02140	Initialize the totals to 0.
02150	Open the file for input. In this section of the program you will never *write to* the file, you will just *read from* it.
02160	Go to the subroutine (line 2510) that prints the heading on your reports.
02180-02240	This loop reads the file and prints data for the items we want. If you choose option 5, only zero-quantity items will be printed. The totals are accumulated for the report on both options, but they'll always be zero for option 5.
02250-02260	When the loop ends, advance the paper to a new page, and start the loop over again. Continue the loop until you reach the end of the file. Notice the EOF RPTEND, which sends control to line 2280 when you reach the end of the file.
02280-02360	After you reach the end of the file, print the totals line. If there is not enough room on the page, skip to the next page. Skip a line between the items and the totals.
02370-02380	Close the file and return to the Main Menu line (990).

Example inventory program

Program (continued)

```
02390 !
02400 ! SUBROUTINES USED IN REPORTS
02410 !
02420 PAGEND: FOR I=LINENO TO PAGELEN
02430 PRINT #255:"|",TAB(95);"|" ! SKIP A LINE
02440 NEXT I
02450 PRINT #255:STATRPT$(1)
02460 RETURN
02470 !
02480 PAGESKIP: GOSUB PAGEND
02490 PRINT #255:NEWPAGE
02500 !
02510 PAGEHEAD: LET PAGENO=PAGENO+1
02520 PRINT #255:STATRPT$(1),TAB(1);STATRPT$(2)
02530 PRINT #255:"|",TAB(95);"|" ! SKIP A LINE
02540 PRINT #255,USING DATEFORM:DATE$,OPTNAME$,PAGENO
02550 PRINT #255:STATRPT$(3),TAB(1);STATRPT$(4)
02560 PRINT #255:STATRPT$(3)
02570 LET LINENO=1
02580 RETURN
```

Lines	Description
02420-02460	PAGEND is a subroutine that skips blank lines between the totals line and the bottom line of the report page. This subroutine keeps all the pages the same size.
02480-02580	PAGESKIP is a subroutine that ends the current page, by using the PAGEND subroutine lines (2420-2460). Then the page advances, and the headings are printed.
02510-02580	PAGEHEAD is a subroutine that prints the headings for a report. This subroutine is actually a part of the PAGESKIP subroutine. PAGEHEAD prints the headings without skipping a page first.
02570	You set LINENO=1, in case a page is skipped immediately before printing the totals.
02580	Notice that PAGESKIP and PAGEHEAD use the same RETURN.

Program (continued)

```
02590 ! -------------------------------------------------------
02600 ! ERROR ACTIONS
02610 ! -------------------------------------------------------
02620 INPCONV: LET CURSPARM=CNT+1
02630 PRINT FIELDS MAT MSGFSP$:MSG$(14),MSG$(15),"","",""
02640 GOSUB CURSPOS
02650 RETRY
02660 !
02670 NRERR: PRINT FIELDS MAT MSGFSP$:MSG$(18),MSG$(19),"","",""
02680 GOTO INQINP
02690 !
02700 ! SUBROUTINE USED FOR ERROR ACTIONS
02710 !
02720 CURSPOS: FOR I=1 TO 4
02730 IF I=CURSPARM THEN LET INPFSP$(1)(13:13)="c" ELSE LET INPFSP$(1)(13:13)=" "
02740 NEXT I
02750 RETURN
02760 ! -------------------------------------------------------
02770 ! END OF PROGRAM
02780 ! -------------------------------------------------------
02790 ! --------------------------------------- ABNORMAL END
02800 ERREND: PRINT NEWPAGE,"EOF or I/O error occurred."
02810 PRINT "Program terminated."
02820 STOP
02830 ! ----------------------------------------- NORMAL END
02840 FIN: PRINT NEWPAGE,"Program ended normally."
02850 END
```

Lines	Description
02620-02650	If there is a conversion error on ADD or INQUIRE, display an error message, position the cursor to the field with the error, and return to the statement where the error occurred.
02670-02680	If we can't find the desired record during INQUIRY, we display an error message and return to INQUIRY to reenter.
02720-02750	This subroutine repositions the cursor to the input field where an error occurs.
02790-02820	The program ends here and a message is displayed if you have an IOERR. (Remember the ON ERROR in line 50.)
02830-02850	The program ends here on normal program termination (when you select option 0 from the menu).

Congratulations! You have completed your course in Learning System/23 BASIC.

To learn more about System/23 BASIC, you can refer to your *BASIC Lanuage Reference* manual.

Index

READER'S COMMENT FORM

SA34-0127-0

VII. Full Screen Processing, Summary, and Examples

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or the IBM branch office serving your locality.

Corrections or clarifications needed:

Page Comment

Please indicate your name and address in the space below if you wish a reply.

____

____

____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

Cut or Fold Along Line

Reader's Comment Form

Cut Along Line

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Systems Publications, Dept 27T
P.O. Box 1328
Boca Raton, Florida 33432

Fold and tape | Please Do Not Staple | Fold and tape

IBM ®

SA34-0127-0
Printed in U.S.A.

SA34-0127-0
Printed in U.S.A.