V. Subroutines and Functions

IBM

Learning System/23 BASIC

V. Subroutines and Functions

IBM

Learning System/23 BASIC

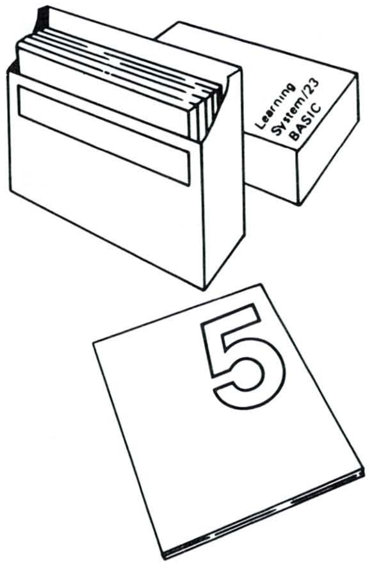
First Edition (January 1981)

V. Subroutines and functions

Contents

V. Subroutines and functions

About this book



This is the fifth in your series of seven books on *Learning System/23 BASIC*. You're already over halfway through this course. But don't stop now. We still have a lot more to show you.

In Book II of this course, you learned how to change the order in which program statements are executed. You learned to use FOR-NEXT loops, IF-THEN statements, and the GOTO statement.

In Book V, you will learn three more ways to change the order of execution. You will learn to use *subroutines*, which are sequences of statements that may be used more than once while a program is running.

You will also learn about *computed* changes in the order of execution. In a computed change, the value of a numeric expression determines the next line number to be executed.

And finally, you will learn about *functions*. A function performs the same action on any number of variables.

Chapter 1. Using subroutines

Introduction

In this chapter, you will learn how to use subroutines. A subroutine is a sequence of statements that may be used more than once while a program is running.

Subroutines are very useful, because they can be used more than once and control can go to a subroutine from many different areas in a program. They can be placed anywhere in a program, which allows you to insert your subroutines near the end of a program. With your subroutines at the end of a program, you won't be so confused when you read a program listing.

Objectives

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

- Write a program that contains a subroutine.
- Call a subroutine by using the GOSUB statement.
- Exit from a subroutine by using the RETURN statement.

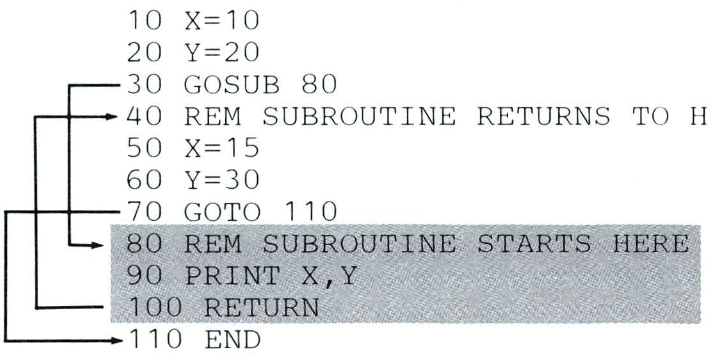
If you are familiar with these tasks, try the exercises located at the end of this chapter. If not, read through the chapter before going on to the exercises.

What is a subroutine?

A *subroutine* is a sequence of statements that may be used more than once during program execution. You need two specific statements to use a subroutine: GOSUB and RETURN.

GOSUB *calls* a subroutine; that is, it sends program control to a specific line number or label. RETURN sends control back to the next program statement following the GOSUB.

Let's look at an example:



```
10  X=10
20  Y=20
30  GOSUB 80
40  REM SUBROUTINE RETURNS TO HERE
50  X=15
60  Y=30
70  GOTO 110
80  REM SUBROUTINE STARTS HERE
90  PRINT X,Y
100 RETURN
110 END
```

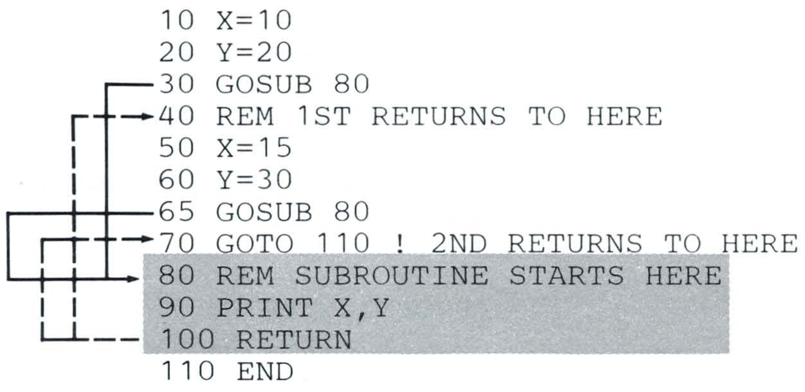
Line 30 calls the subroutine that starts at line 80. When line 100 is executed, control goes back to the line following the GOSUB. In this case, control goes back to line 40.

Notice that we put the subroutine near the end of the program. This makes it easier to follow the flow of control in the program listing.

We said that a subroutine can be used more than once. What would happen if we added line 65?

```
65 GOSUB 80
```

The program would look like this:



```
10 X=10
20 Y=20
30 GOSUB 80
40 REM 1ST RETURNS TO HERE
50 X=15
60 Y=30
65 GOSUB 80
70 GOTO 110 ! 2ND RETURNS TO HERE
80 REM SUBROUTINE STARTS HERE
90 PRINT X,Y
100 RETURN
110 END
```

The same subroutine (lines 80, 90, and 100) is used twice.

When line 30 calls the subroutine, the RETURN in line 100 sends control back to line 40.

When line 65 calls the subroutine, the RETURN in line 100 sends control back to line 70.

What is a subroutine? (continued)

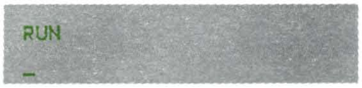
Often, subroutines are used to control paging and print or display headings on reports.

For example, suppose you want a program to print three reports, and you want each report to have the same heading. If you have a printer, enter the following program.

Note: This program has a lot of statements, so take your time and enter the lines carefully.

```
CLEAR
10 OPTION BASE 1
20 DIM ITEM$(5),QUANTITY(5),COST(5)
30 DATA "NUTS","BOLTS","SCREWS"
40 DATA "NAILS","HAMMERS"
50 DATA 2000,1500,850,5000,7
60 DATA .29,.39.,11.,02,10.98
70 READ MAT ITEM$,MAT QUANTITY,MAT COST
80 PRINT #255:HEX$("2B0205000A1042")
90 GOSUB HEADING
100 PRINT #255:MAT ITEM$ ! REPORT 1
110 GOSUB HEADING
120 PRINT #255:MAT QUANTITY ! REPORT 2
130 GOSUB HEADING
140 PRINT #255:MAT COST ! REPORT 3
150 GOTO 200
160 HEADING: ! START A NEW REPORT
170 PRINT #255:NEWPAGE,"XYZ COMPANY"
180 PRINT #255:
190 RETURN
200 END
```

Notice that we are using a label in lines 90, 110, 130, and 160 instead of a line number. The label HEADING identifies the subroutine.



```
RUN
_
```

Now run the program:

```
RUN
```

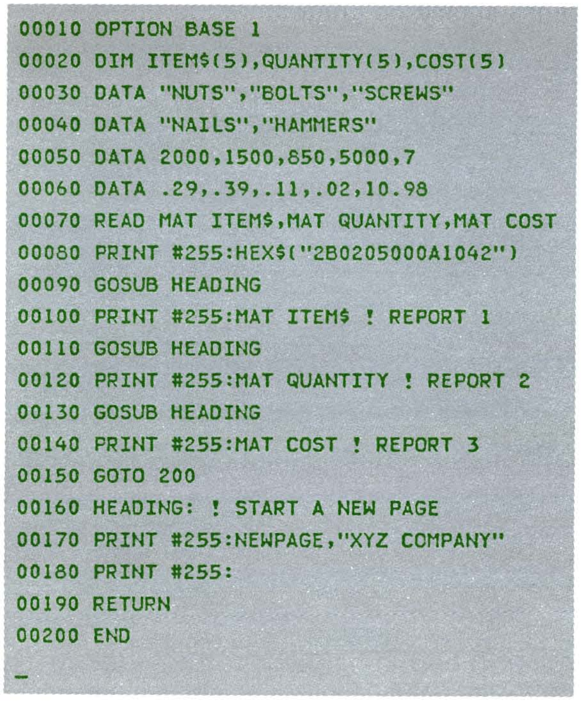
The following pages should be printed:

```
XYZ COMPANY

NUTS
BOLTS
SCREWS
NAILS
HAMMERS


XYZ COMPANY

 2000
 1500
 850
 5000
 7


XYZ COMPANY

 .29
 .39
 .11
 .02
 10.98
```

What is a subroutine? (continued)

List the program:

```
LIST
```



```
00010 OPTION BASE 1
00020 DIM ITEM$(5),QUANTITY(5),COST(5)
00030 DATA "NUTS","BOLTS","SCREWS"
00040 DATA "NAILS","HAMMERS"
00050 DATA 2000,1500,850,5000,7
00060 DATA .29,.39,.11,.02,10.98
00070 READ MAT ITEM$,MAT QUANTITY,MAT COST
00080 PRINT #255:HEX$("2B0205000A1042")
00090 GOSUB HEADING
00100 PRINT #255:MAT ITEM$ ! REPORT 1
00110 GOSUB HEADING
00120 PRINT #255:MAT QUANTITY ! REPORT 2
00130 GOSUB HEADING
00140 PRINT #255:MAT COST ! REPORT 3
00150 GOTO 200
00160 HEADING: ! START A NEW PAGE
00170 PRINT #255:NEWPAGE,"XYZ COMPANY"
00180 PRINT #255:
00190 RETURN
00200 END
_
```

What happened when you ran this program?

First, as you learned in Book IV, you assigned values to entire arrays by using a READ MAT statement (line 70).

After the values were assigned, you skipped to a new page and printed a heading:

```
XYZ COMPANY
```

Then you skipped a line on the page. This was all done in the subroutine (lines 160-190).

The first report was then printed. It showed the names of the items (line 100).

Then you called the heading subroutine again. This time, it returned to line 120.

The second report was printed (line 120).

You called the heading subroutine again. This time, it returned to line 140.

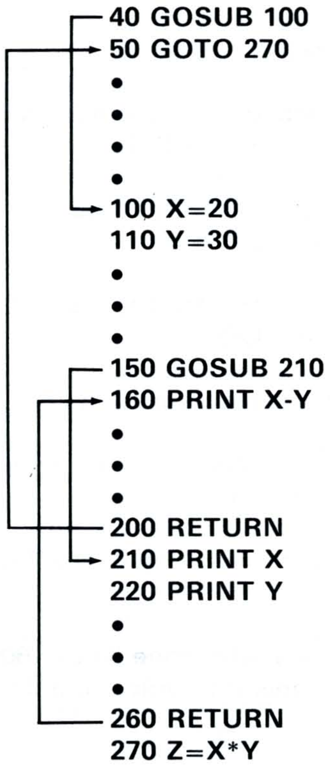
The third report was printed (line 140), and the program ended.

Notice that one subroutine (lines 160-190) was used three times. Each time, it returned to a different line number.

Nested subroutines

One subroutine can call another subroutine. That means that you do not have to execute a RETURN statement in one subroutine before going to another subroutine. These are sometimes called *nested* subroutines.

Look at the following example:



```
40 GOSUB 100
50 GOTO 270
●
●
●
●
100 X=20
110 Y=30
●
●
●
150 GOSUB 210
160 PRINT X-Y
●
●
●
200 RETURN
210 PRINT X
220 PRINT Y
●
●
●
260 RETURN
270 Z=X*Y
```

If we assume there are no other statements that direct program control, the order of execution in this example is:

Line number

40
100-150
210-260
160-200
50
270

Whenever a RETURN statement is executed, program control goes to the line number following the last GOSUB statement that was executed.

Remember: A subroutine requires *both* a GOSUB and a RETURN. You can't have a GOSUB without a RETURN. You can't have a RETURN without a GOSUB. But one RETURN statement can send control to different lines in a program.

Note when using subroutines: All variables will keep the value they had before the subroutine was called, unless statements in the subroutine cause them to change.

Using subroutines

Chapter summary

A subroutine is a sequence of program statements that may be used more than once during program execution. It requires two programming statements:

- GOSUB—tells program control to go to a specific line number.
- RETURN—tells program control to go back to the line following the GOSUB.

One subroutine can call another subroutine, but each subroutine requires a RETURN statement.

Exercises

Question 1

Add a statement to the following program, on line 20, that sends control to a subroutine beginning in line 40.

```
10 NAME$="XXX BUILDING"
30 STOP
40 PRINT NAME$
50 RETURN
60 END
```

Answer: ______________________________

Question 2

What is the order of execution in the following program?

```
10 GOSUB 40
20 PRINT "JOHN DOE"
30 GOTO 60
40 PRINT "EMPLOYEE"
50 RETURN
60 PRINT "MARY SMITH"
70 END
```

Answer: ______________________________
____
____
____
____
____
____
____

Exercises (continued)

Question 3

What is the order of execution in the following program?

```
10 GOSUB 30
20 STOP
30 X=1
40 PRINT X
50 GOSUB 80
60 PRINT X
70 RETURN
80 X=2
90 RETURN
100 END
```

Answer: ________________________________

Answers

Question 1

```
20 GOSUB 40
```

Question 2

Line number

10
40
50
20
30
60
70

Question 3

Line number

10
30
40
50
80
90
60
70
20

Chapter 2. Making decisions with ON

Introduction

In this chapter, you will learn another way to control the order of program execution. You will learn how to use the ON statement with GOTO and GOSUB.

When you use the ON statement, your programs will not send control to one specific line. Instead, your programs will *compute*, or calculate, the value of an expression. The line that will get control next will depend upon the calculated value.

Objectives

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

- Change the order of program execution by using the ON GOTO statement.
- Change the order of program execution by using the ON GOSUB and RETURN statements.

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.

Making decisions with ON

Using ON GOTO

In Book II of this course, you learned how to use the GOTO statement to change the order of execution.

```
10 PRINT "JANUARY"
20 GOTO 40
30 PRINT "FEBRUARY"
40 PRINT "MARCH"
50 END
```

Line 20 sends control directly to line 40.

You also learned how to test the value of an expression. The resulting value decided whether you went to another line or not.

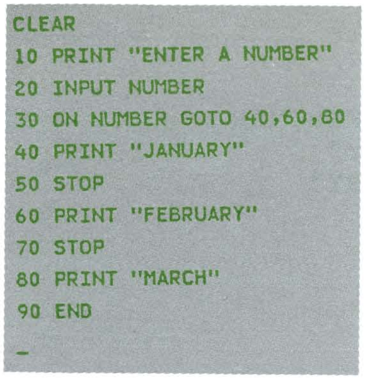
```
10 PRINT "ENTER A NUMBER"
20 INPUT NUMBER
30 IF NUMBER>5 THEN GOTO 70
40 IF NUMBER<5 THEN GOTO 90
50 PRINT "NUMBER = 5"
60 GOTO 100
70 PRINT "NUMBER > 5"
80 GOTO 100
90 PRINT "NUMBER < 5"
100 END
```

Lines 30 and 40 test the value of NUMBER. Each line sends program control to a specific line number, depending on the value of NUMBER.

Now we'll show you how one statement can send control to one of several different lines. Again, we will test the value of NUMBER.

Enter the following program:

```
CLEAR
10 PRINT "ENTER A NUMBER"
20 INPUT NUMBER
30 ON NUMBER GOTO 40,60,80
40 PRINT "JANUARY"
50 STOP
60 PRINT "FEBRUARY"
70 STOP
80 PRINT "MARCH"
90 END
```



```
CLEAR
10 PRINT "ENTER A NUMBER"
20 INPUT NUMBER
30 ON NUMBER GOTO 40,60,80
40 PRINT "JANUARY"
50 STOP
60 PRINT "FEBRUARY"
70 STOP
80 PRINT "MARCH"
90 END
_
```

What happens when you run this program?

First, you will enter a number. In line 30, your System/23 tests the number.

If the number is 1, control goes to line 40. If the number is 2, control goes to line 60. If the number is 3, control goes to line 80.

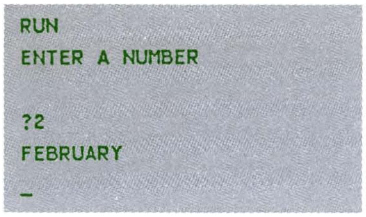
This is a *computed* GOTO. Your System/23 computes the value of an expression (in this case, NUMBER). The value tells your System/23 which line to send control to.

```
30 ON NUMBER GOTO 40,60,80
                  ↑  ↑  ↑
(Value of NUMBER)=1, 2, 3
```

Control transfers to the line whose position in the list equals the value of NUMBER.

Making decisions with ON

Using ON GOTO (continued)

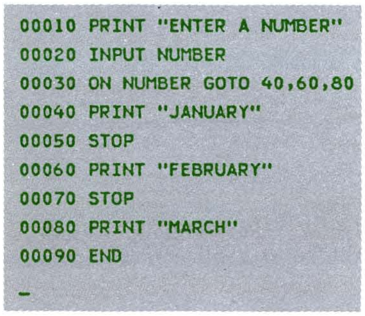


```
RUN
ENTER A NUMBER

?2
FEBRUARY
_
```

Now run the program and enter 2:

```
RUN
ENTER A NUMBER
?2
```

What happened? FEBRUARY was displayed on the screen when you entered a 2. Control went to line 60, because 60 was the *second* line number in the list.



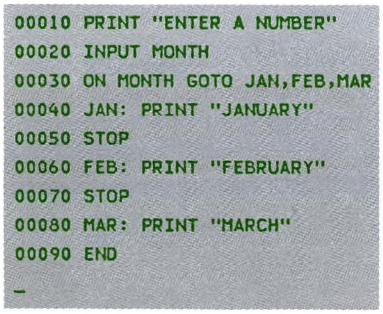
```
00010 PRINT "ENTER A NUMBER"
00020 INPUT NUMBER
00030 ON NUMBER GOTO 40,60,80
00040 PRINT "JANUARY"
00050 STOP
00060 PRINT "FEBRUARY"
00070 STOP
00080 PRINT "MARCH"
00090 END
_
```

List your program:

```
LIST
```

Let's see if we can make this a little easier to follow by using labels. Enter these changes:

```
20 INPUT MONTH
30 ON MONTH GOTO JAN,FEB,MAR
40 JAN: PRINT "JANUARY"
60 FEB: PRINT "FEBRUARY"
80 MAR: PRINT "MARCH"
```



```
00010 PRINT "ENTER A NUMBER"
00020 INPUT MONTH
00030 ON MONTH GOTO JAN,FEB,MAR
00040 JAN: PRINT "JANUARY"
00050 STOP
00060 FEB: PRINT "FEBRUARY"
00070 STOP
00080 MAR: PRINT "MARCH"
00090 END
_
```

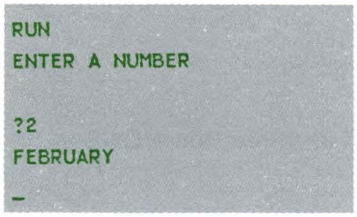
Now list your program:

```
LIST
```

In this version, you input a value for the variable MONTH. If you input a 1, control goes to the statement with the label JAN. If you input a 2, control goes to FEB. And if you input a 3, control goes to MAR.

```
30 ON MONTH GOTO JAN,FEB,MAR
                  ↗   ↗   ↗
(Value of MONTH)=1,  2,  3
```

Run the program again, and enter a 2:

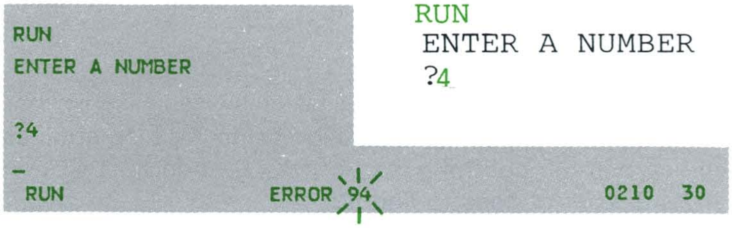


```
RUN
ENTER A NUMBER

?2
FEBRUARY
_
```

```
RUN
 ENTER A NUMBER
 ?2
```

You can see that the results are the same.

What would happen if you entered a 4? Let's try it. Enter:



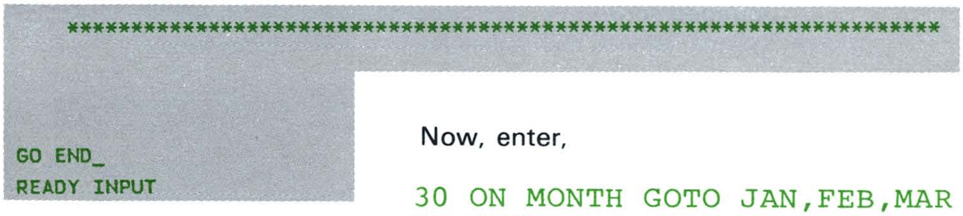
```
RUN
ENTER A NUMBER

?4

_
 RUN                        ERROR 94                   0210   30
```

```
RUN
 ENTER A NUMBER
 ?4
```

The program is interrupted with an error. You entered a 4 for NUMBER, but there are only three line numbers following the GOTO.

How can you correct this? Press the Error Reset, and enter:

```
GO END
```



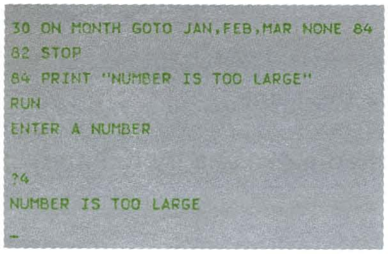
```
     *****************************************************************************



GO END_
READY INPUT
```

Now, enter,

```
30 ON MONTH GOTO JAN,FEB,MAR NONE 84
82 STOP
84 PRINT "NUMBER IS TOO LARGE"
```

Making decisions with ON

Using ON GOTO (continued)



```
30 ON MONTH GOTO JAN,FEB,MAR NONE 84
82 STOP
84 PRINT "NUMBER IS TOO LARGE"
RUN
ENTER A NUMBER

?4
NUMBER IS TOO LARGE
_
```

Now run the program again and enter a 4:

```
RUN
ENTER A NUMBER
?4
```

If you enter any number greater than three, the NONE in line 30 sends control to line 84.

The value you test in an ON-GOTO statement can be any arithmetic expression. For example,

```
10 ON X+1 GOTO 30,50,70
```

In this example, the value of the expression X+1 determines which line number control goes to. If X=0, then X+1=1. So, if X=0, control goes to line 30.

```
10 ON X+1 GOTO 30,50,70
                ↑  ↑  ↑
(Value of X+1)= 1, 2, 3
```

If the expression produces a decimal value that is not a whole number, it is rounded to a whole number (an integer). For example, if the value of the expression equals 2.75, control transfers to the *third* line number in the list. (2.75 is rounded to 3.)

Let's look at another example. This time, we'll show a list of options available in an example inventory program. This inventory program will be expanded in Books VI and VII.

```
10 PRINT "1 = CREATE FILE"
20 PRINT "2 = ADD ITEM"
30 PRINT "3 = UPDATE ITEM"
40 PRINT "ENTER OPTION NO."
50 INPUT CHOICE
60 ON CHOICE GOTO 80,110,140 NONE 70
70 STOP
80 REM OPTION 1
•
•
110 REM OPTION 2
•
•
140 REM OPTION 3
•
•
170 END
```

If you choose OPTION 1, control goes to line 80. If you choose OPTION 2, control goes to line 110. If you choose OPTION 3, control goes to line 140.

Your turn!

What happens if you enter 5?

____

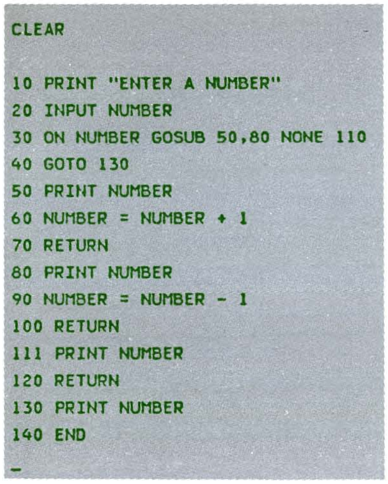
Answer: The program stops (line 70).

Making decisions with ON

Using ON GOSUB

You can also use the ON statement to send program control to one of several different subroutines.

Enter the following program:



```
CLEAR

10 PRINT "ENTER A NUMBER"
20 INPUT NUMBER
30 ON NUMBER GOSUB 50,80 NONE 110
40 GOTO 130
50 PRINT NUMBER
60 NUMBER = NUMBER + 1
70 RETURN
80 PRINT NUMBER
90 NUMBER = NUMBER - 1
100 RETURN
111 PRINT NUMBER
120 RETURN
130 PRINT NUMBER
140 END
_
```

```
CLEAR
10 PRINT "ENTER A NUMBER"
20 INPUT NUMBER
30 ON NUMBER GOSUB 50,80 NONE 110
40 GOTO 130
50 PRINT NUMBER
60 NUMBER = NUMBER + 1
70 RETURN
80 PRINT NUMBER
90 NUMBER = NUMBER - 1
100 RETURN
110 PRINT NUMBER
120 RETURN
130 PRINT NUMBER
140 END
```

Notice that line 30 sends control to three different subroutines.

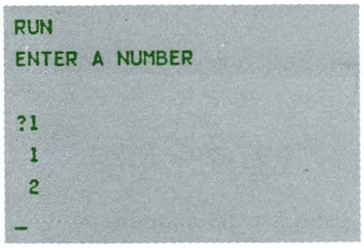
If you enter 1 for NUMBER, you use the first subroutine (lines 50-70). If you enter 2, you use the second subroutine (lines 80-100).

If you enter any other number, you use the third subroutine (lines 110-120).

Remember that values are rounded to integers (whole numbers) when being tested in ON statements.

As with any subroutine, you need the statements GOSUB and RETURN. However, in this program, the RETURNs in lines 70, 100, and 120 *all* send control to line 40.

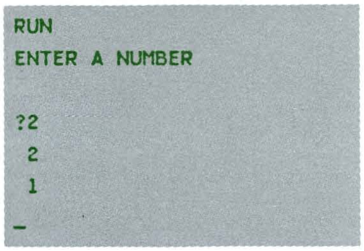
Let's run the program three times to see the different results. The first time, enter 1 for NUMBER:



```
RUN
ENTER A NUMBER

?1
 1
 2
_
```

```
RUN
 ENTER A NUMBER
 ?1
```

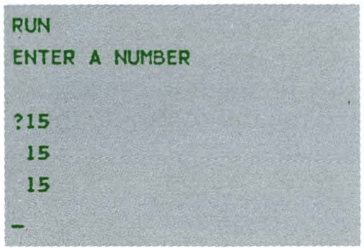
Now run the program again and enter 2 for NUMBER:



```
RUN
ENTER A NUMBER

?2
 2
 1
_
```

```
RUN
 ENTER A NUMBER
 ?2
```

And finally, run the program and enter 15 for NUMBER:



```
RUN
ENTER A NUMBER

?15
 15
 15
_
```

```
RUN
 ENTER A NUMBER
 ?15
```

Just like a computed GOTO, a *computed* GOSUB tests the value of an expression (in this case, NUMBER). Control goes to a subroutine whose first line number is in the list following the word GOSUB. Control goes to the line whose position in the list equals the value of NUMBER.

```
30 ON NUMBER GOSUB 50,80 NONE 110
                   ↑  ↑
   (Value of NUMBER)=1, 2
```

If NUMBER does not equal 1 or 2, control goes to line 110.

Don't forget to end all subroutines with RETURN.

Making decisions with ON

Chapter summary

These statements allow you to change the order of program execution:

- GOTO—sends program control to a specific line number.

```
10 GOTO 40
```

- ON GOTO—sends program control to one of a list of line numbers.

```
10 ON X+1 GOTO 30,50,100,180 NONE 20
```

- GOSUB/RETURN—tells the program to execute a specific subroutine and returns control to the next line following the GOSUB.

```
10 GOSUB 40
●
●
40
●
●
70 RETURN
```

- ON GOSUB/RETURN—tells the program to execute one of a list of subroutines and returns control to the next line following the ON GOSUB.

```
10 ON X+1 GOSUB 30,50 NONE 20
20
30
40 RETURN
50
60
70 RETURN
```

Exercises

Question 1

What is the order of execution in the following programs?

```
a. 10 X=2
   20 ON X GOTO 50,30,40
   30 PRINT X
   40 PRINT X-1
   50 END
```

Answer: ______________________________

____

____

____

____

```
b. 10 X=1
   20 ON X GOTO 50,30,40
   30 PRINT X
   40 PRINT X-1
   50 END
```

Answer: ______________________________

____

____

Exercises (continued)

Question 2

What is the order of execution in the following program?

```
10 NUMBER=5
20 ON NUMBER GOTO 80,40 NONE 30
30 NUMBER=1
40 ON NUMBER GOSUB 60,70
50 GOTO 80
60 PRINT NUMBER
70 RETURN
80 END
```

Answer:

Answers

Question 1

a. Line number

    10
    20
    30
    40
    50

b. Line number

    10
    20
    50

Question 2

Line number

10
20
30
40
60
70
50
80

Chapter 3. Using functions

Introduction

Sometimes in a program, you may want to perform the same action on several different variables. In this chapter, we will show you how to use *functions* to perform the same action on a number of values.

We will introduce you to several functions that are stored internally in your System/23. We will also show you how to define your own functions in a program.

Objectives

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

- Use the system functions SQR, ROUND, POS, and RPT$.
- Define a function in a single line by using the DEF statement.
- Define a function in a number of lines by using the DEF, LET, and FNEND statements.
- Join two character strings to form one string by using &.
- Refer to a specific portion of a character string.

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.

System functions

A *function* allows your System/23 to perform the same action on a number of different variables.

Several functions are stored internally on your System/23, almost like the internal constant PI. These functions perform several commonly used operations and always have the same meaning. They are called *system functions.*

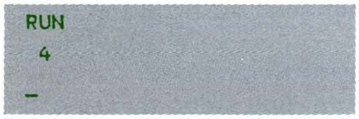
For example, the system function SQR produces the square root of a number. However, your System/23 produces approximate values for SQR, so you should use OPTION RD with SQR.

Let's try it and see. Enter:

```
CLEAR
10 OPTION RD 00
20 PRINT SQR(16)
30 END
```

Now run the program:

```
RUN
```



```
RUN
 4
_
```

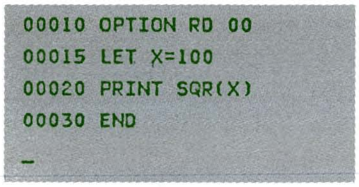
The number 4, which is the square root of 16, is displayed. SQR(16) = 4.

You can also use a variable with SQR. Enter the following:

```
15 X=100
20 PRINT SQR(X)
```

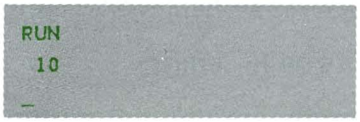
Now list the new version of your program:

```
LIST
```



```
00010 OPTION RD 00
00015 LET X=100
00020 PRINT SQR(X)
00030 END
_
```

Now run your program:



```
RUN
 10

_
```

```
RUN
```

As you can see, the number 10 is displayed. X=100, so SQR(X)=SQR(100), and SQR(100)=10.

Another system function is ROUND. The ROUND function allows you to round an individual number to a specified number of digits. For example,

$$\texttt{ROUND(}\underbrace{\texttt{5.99001}}_{\text{Original number}}\texttt{,}\underset{\substack{\uparrow\\\text{Two digits to the right of the decimal point}}}{\texttt{2}}\texttt{)=5.99}$$

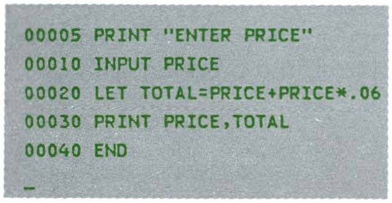
Let's see how this can be used in a program. We'll use the PRICE program you saved in Book II.

Load your program back into the work area:

```
LOAD PRICE
```

List the program:

```
LIST
```



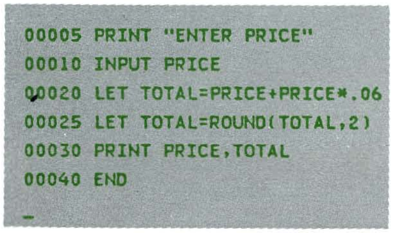
```
00005 PRINT "ENTER PRICE"
00010 INPUT PRICE
00020 LET TOTAL=PRICE+PRICE*.06
00030 PRINT PRICE,TOTAL
00040 END
_
```

We want to make sure that the total price is always rounded to dollars and cents. So, enter:

```
25 TOTAL=ROUND(TOTAL,2)
```

List the new version of your program:

```
LIST
```



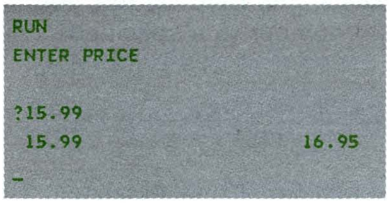
```
00005 PRINT "ENTER PRICE"
00010 INPUT PRICE
00020 LET TOTAL=PRICE+PRICE*.06
00025 LET TOTAL=ROUND(TOTAL,2)
00030 PRINT PRICE,TOTAL
00040 END
_
```

System functions (continued)

Notice in line 25 that we are changing the value of TOTAL. The new value will be rounded to two decimal places.

Now, run the program and enter 15.99 for PRICE:

```
RUN
ENTER PRICE

?15.99
```

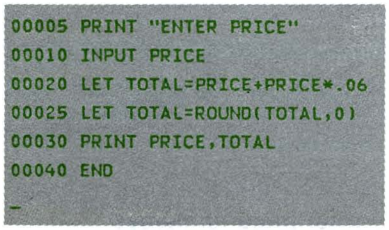


```
RUN
ENTER PRICE

?15.99
 15.99                        16.95
_
```

You can see that 16.95 is displayed, instead of 16.9494. Remember that if you had entered OPTION RD 2, the number 15.99 would also be rounded when it was displayed. But, 15.99 already has only two digits to the right of the decimal point.

Let's look at one more example. Change the 2 in line 25 to a 0. It will look lik this:

```
25 TOTAL=ROUND(TOTAL,0)
```

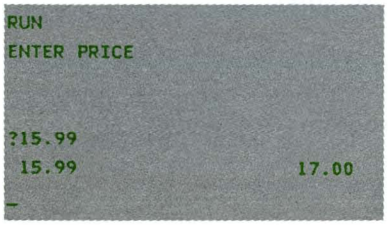


```
00005 PRINT "ENTER PRICE"
00010 INPUT PRICE
00020 LET TOTAL=PRICE+PRICE*.06
00025 LET TOTAL=ROUND(TOTAL,0)
00030 PRINT PRICE,TOTAL
00040 END
_
```

List your program:

```
LIST
```

In this example, we want the total to be rounded to an even number of dollars. Go ahead and run the program with 15.99 for PRICE:



```
RUN
ENTER PRICE


?15.99
 15.99                        17.00
_
```

```
RUN
ENTER PRICE

?15.99
```

We're going to show you two more system functions. The first is POS. It works like this:

POS(A$,B$,X)

Position of B$ in A$, beginning at position X in A$

For example, if

```
A$="ABCABCDE"
and
B$="BC"
```

Then

```
POS(A$,B$,1)=2
and
POS(A$,B$,4)=5
```

The string A$ is searched from left to right for the string B$. In this case, B$ equals "BC".

If X = POS(A$,B$,1), the search begins in position 1. The first place BC is found is in position 2. So, X = 2.

If X = POS(A$,B$,4), the search begins in position 4. The first place BC is found is in position 5. So, X = 5.

If BC is not found in A$, the result is 0.

We'll show you an example of POS on the next page.

System functions (continued)

The last system function we're going to show you is RPT$. It works like this:

```
RPT$(A$,X)
```

Repeat the string A$, X times.

For example, if

```
A$="SA"
```

Then

```
RPT$(A$,3)="SASASA"
```

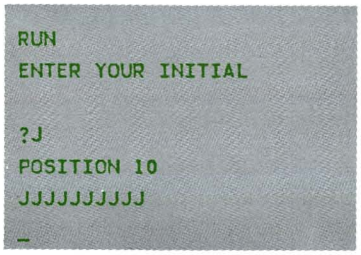
Let's use POS and RPT$ in a program. Enter the following:

```
CLEAR
10 DIM A$*26
20 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
30 PRINT "ENTER YOUR INITIAL"
40 INPUT I$
50 X=POS (A$,I$,1)
60 PRINT "POSITION";X
70 PRINT RPT$(I$,X)
80 END
```

Now, run the program and enter your initial. (We'll use J in our example, you use *your* first initial.)

```
RUN
ENTER YOUR INITIAL
?J
```

Your initial is repeated the number of times indicated after POSITION.



```
RUN
ENTER YOUR INITIAL

?J
POSITION 10
JJJJJJJJJJ

_
```

Single-line functions

You can also define your own functions in a program. The statement for a *single-line function* looks like this:

$$\text{DEF } \underset{\text{Function name}}{\text{FNS}}(\underset{\text{Dummy variable}}{\text{X}}) = \underbrace{2 * \text{X} + 10}_{\text{Value returned}}$$

Where DEF means define; FN means function, and S is the function name. You must always enter FN *and* the function name (for example, FNS) when you use a function in a program.

The function can have any numeric variable name. The X is a *dummy* variable. It stands for any numeric variable in the program. It also can be any numeric variable name. The value of the dummy variable is not affected if used elsewhere in the program.

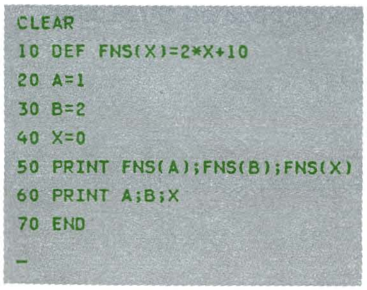
In this example, the function S will always produce a value equal to two times the original value plus ten. For example,

if Z=3, FNS(Z) =2 * 3 + 10 = 16

if B=5, FNS(B) =2 * 5 + 10 = 20

Let's look at another example. Enter the following program:

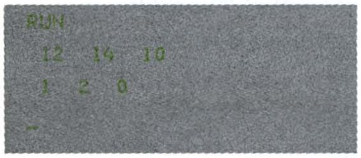
```
CLEAR
10 DEF FNS(X) = 2 * X + 10
20 A=1
30 B=2
40 X=0
50 PRINT FNS(A);FNS(B);FNS(X)
60 PRINT A;B;X
70 END
```



```
CLEAR
10 DEF FNS(X)=2*X+10
20 A=1
30 B=2
40 X=0
50 PRINT FNS(A);FNS(B);FNS(X)
60 PRINT A;B;X
70 END
-
```

Using functions

Single-line functions (continued)



```
RUN
 12  14  10
 1  2  0

_
```

Now run the program:

```
RUN
```

Notice that FNS(1), FNS(2), and FNS(0) are not individually defined. The S function of any value is defined in line 10.

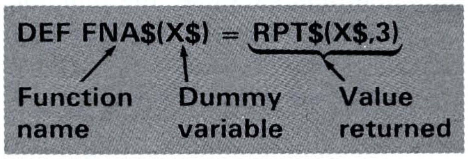
```
10 DEF FNS(X) = 2 * X + 10
```

So, FNS(1) equals 12, FNS(2) equals 14, and FNS(0) equals 10.

Notice also that the values of A, B, and X do not change when you use the functions.

You can also use single-line functions to assign values to character variables.

The statement for a *single-line function* looks like this:



```
DEF FNA$(X$) = RPT$(X$,3)
```

Where DEF means define; FN means function, and A$ is the function name. The function can have any character variable name. The X$ is a *dummy* variable. It stands for any character variable in the program.

In this example, the function A$ is defined. This function produces the original character string repeated three times.

If N$ = "J", then FNA$(N$) = "JJJ".

Let's look at two more examples. These functions will show you two more things you can do with strings.

```
A$="MIAMI, FL"
B$=" 33133"
C$=A$&B$
D$=C$(11:15)
```

The value C$(11:15) means characters 11 through 15 of the string C$. You can also assign a value to certain positions in a string. For example, to make the first three characters of a string E$ be blank, you would enter E$(1:3)="   ".

The & symbol joins two character strings. The length of the resulting string equals the sum of the lengths of the two joining strings. So,

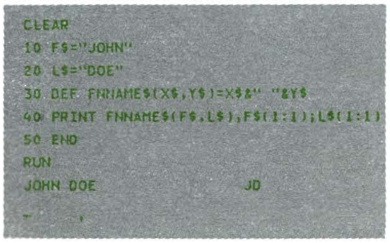
```
C$="MIAMI, FL 33133"
D$="33133"
```

Let's use the & symbol in a function in a program. Enter the following, but use *your* name in lines 10 and 20:

```
CLEAR
10 F$="JOHN"
20 L$="DOE"
30 DEF FNNAME$ (X$,Y$)=X$&" "&Y$
40 PRINT FNNAME$(F$,L$),F$(1:1);L$(1:1)
50 END
```

In line 40, we display your name and your initials. Now run the program:

```
RUN
```



```
CLEAR
10 F$="JOHN"
20 L$="DOE"
30 DEF FNNAME$(X$,Y$)=X$&" "&Y$
40 PRINT FNNAME$(F$,L$),F$(1:1);L$(1:1)
50 END
RUN
JOHN DOE                JD
```

Using functions

Multiple-line functions

A *multiple-line function* also allows your System/23 to perform the same action on a number of different variables. However, it requires more than one statement to define the function. It requires at least the following three statements:

DEF FNM(X)

Function name **Dummy variable**

●

●

LET FNM = $\underbrace{\textbf{X+1}}_{\textbf{Value returned}}$

Function name

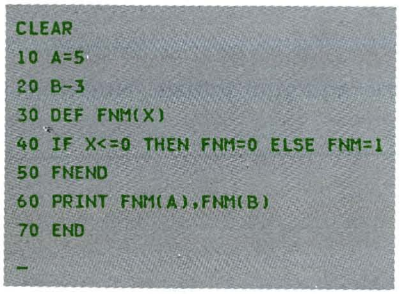
●

●

$\underbrace{\textbf{FNEND}}_{\textbf{Required at end of function}}$

The value of the function is the value last assigned by a LET statement. Remember that the word LET is optional. LET X=1 is the same as X=1.

Let's look at an example. Enter the following:



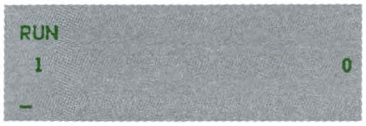
```
CLEAR
10 A=5
20 B-3
30 DEF FNM(X)
40 IF X<=0 THEN FNM=0 ELSE FNM=1
50 FNEND
60 PRINT FNM(A),FNM(B)
70 END
_
```

```
CLEAR
10 A=5
20 B=-3
30 DEF FNM(X)
40 IF X<=0 THEN FNM=0 ELSE FNM=1
50 FNEND
60 PRINT FNM(A),FNM(B)
70 END
```

This function will always produce one of two different values: 0 or 1.

```
A>0, so FNM(A)=1
B<0, so FNM(B)=0
```

Go ahead and run the program:



```
RUN
 1                                    0
_
```

```
RUN
```

Notice that in a multiple-line function, the first line (beginning with DEF) does not assign a value. It is assigned later in the function with LET. A multiple-line function can be used for numeric *or* character variables, but not both.

Things to remember when defining a function:

- A function can be defined anywhere in a program, either before or after it is referenced.
- A function can be defined only once in a program.
- If you use X as a dummy variable to define a function, such as DEF FNA(X), the value of X will not be affected if used elsewhere in the program.
- In a multiple-line function, a value is not assigned in the DEF statement.
- You cannot input or output data within the statements that define the function if that function is called from an I/O statement.

For example, you cannot enter:

Multiple-line functions (continued)

```
10 PRINT FNA(5)
20 STOP
30 DEF FNA(X)
40 PRINT 5
50 FNEND
```

Chapter summary

A function performs the same action on several different variables. There are two different types of functions: those stored internally on your System/23; and those that you define. You can define two different types of functions:

- Single-line—The function is defined in one statement.

```
DEF FNA(X)=X*4+X-1
or
DEF FNA$(X$)=X$&"ABC"
```

- Multiple-line—The function is defined in several statements.

```
DEF FNA(X)
FNA=X*4+X-1
FNEND
or
DEF FNA$(X$)
FNA$=X$&"ABC"
FNEND
```

Several system functions are stored in your System/23. They include:

- SQR(X)—This returns the square root of X.
- ROUND(X,2)—This returns the number X rounded to 2 decimal places.
- POS(X$,Y$,1)—This returns the first position of a character string Y$ in the string X$, beginning at position 1.
- RPT$(X$,3)—This returns the character string X$, repeated 3 times.

Chapter summary (continued)

Several other system functions are stored on your System/23. Refer to "System functions" in your *BASIC Language Reference* manual for a complete list.

You can add two character strings together with the & symbol. You indicate specific characters in a string, such as characters 1 through 4, with X$(1:4).

Exercises

Question 1

What will be displayed if you run the following programs?

```
a. 10 X=-4
   20 Y=10
   30 DEF FNS(T)=T*10
   40 PRINT FNS(X),FNS(Y)
   50 END

b. 10 DEF FNCIRC(R)
   20 FNCIRC=2*PI*R
   30 FNEND
   40 RADIUS=5
   50 PRINT USING 60:FNCIRC(RADIUS)
   60 FORM N 7.3
   70 END

c. 10 A$="NEW"
   20 B$="S"
   30 DEF FNM$(X$,Y$)=X$&Y$
   40 PRINT FNM$(A$,B$)
   50 END
```

Answer: a. ______________________

b. ______________________

c. ______________________

Exercises (continued)

Question 2

What will be displayed if you run the following programs?

```
a.  10 T$="TEXAS"
    20 X$="E"
    30 PRINT POS(T$,X$,1)
    40 PRINT ROUND(183.0010,3)
    50 END

b.  10 I$="EMPLOYEE1"
    20 OPTION RD 00
    30 PRINT I$(1:3)
    40 PRINT SQR(9)
    50 END
```

Answer: a. ______________________________

____

b. ______________________________

____

Answers

Question 1

```
a. -40                                  100
```

```
b.  31.416
```

```
c. NEWS
```

Question 2

```
a.  2
    183.001
```

```
b.  EMP
    3
```

Cut or Fold Along Line

READER'S COMMENT FORM

SA34-0125-0

V. Subroutines and Functions

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.)

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®

SA34-0125-0
Printed in U.S.A.

SA34-0125-0
Printed in U.S.A.