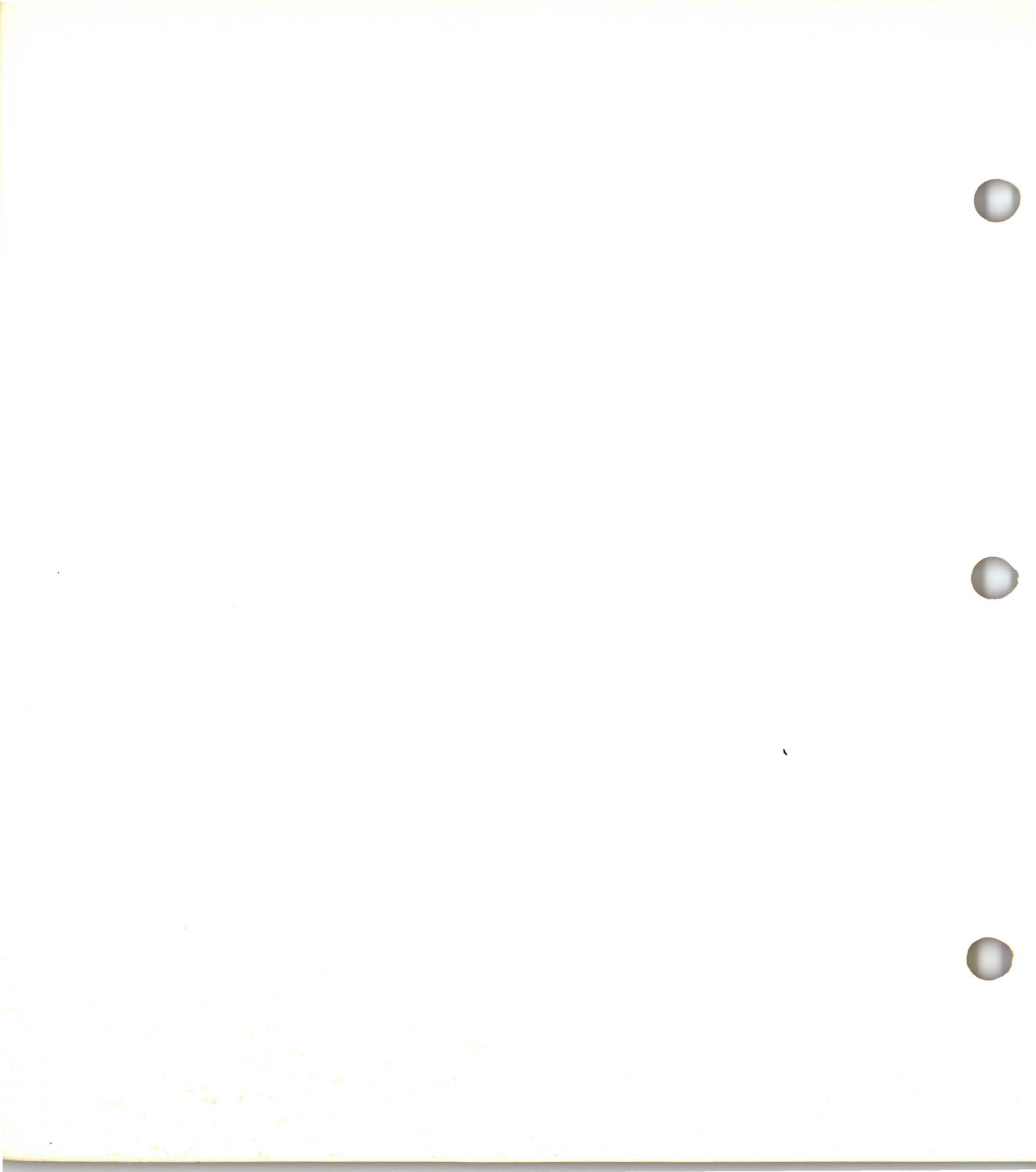


## II. Inputting Data and Using Loops

IBM

Learning System/23 BASIC



## II. Inputting Data and Using Loops



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.



# II. Inputting data and using loops

---

## Contents

<b>About this book</b> .....	v
<b>Chapter 1. Assigning values from the keyboard</b> .....	1-1
Introduction.....	1-1
Assigning a numeric value .....	1-2
Assigning a character value .....	1-4
Assigning more than one value.....	1-6
Chapter summary.....	1-13
Exercises .....	1-14
Answers .....	1-16
<b>Chapter 2. Changing the order of execution</b> .....	2-1
Introduction.....	2-1
Using the GOTO statement.....	2-2
Using labels .....	2-5
Chapter summary.....	2-7
Exercises .....	2-8
Answers .....	2-9
<b>Chapter 3. Branching with IF, THEN, ELSE</b> .....	3-1
Introduction.....	3-1
Using IF - THEN.....	3-2
Adding a statement to IF - THEN .....	3-10
Adding ELSE to IF - THEN.....	3-13
Using STOP .....	3-15
Chapter summary.....	3-16
Exercises .....	3-17
Answers .....	3-20
<b>Chapter 4. Using loops to control a program</b> .....	4-1
Introduction.....	4-1
Using an IF - THEN loop.....	4-2
Using a FOR - NEXT loop.....	4-6
Using nested loops .....	4-10
Chapter summary.....	4-14

# II. Inputting data and using loops

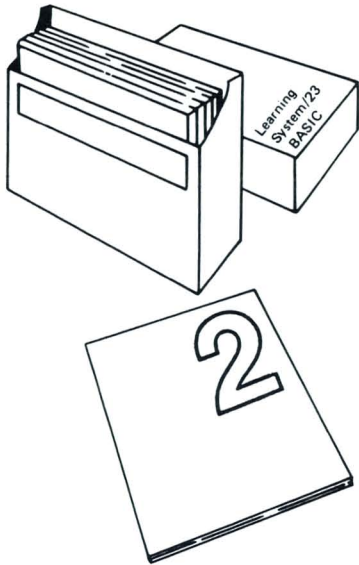
---

## Contents (continued)

Exercises .....	4-15
Answers .....	4-17
<b>Chapter 5. Assigning values with READ and DATA</b> .....	5-1
Introduction .....	5-1
Using READ and DATA statements .....	5-2
Using the RESTORE statement .....	5-9
Chapter summary .....	5-12
Exercises .....	5-13
Answers .....	5-16

---

## About this book



You've already finished your first book of *Learning System/23 BASIC*. It was probably even easier than you expected. And just think, you've already written several BASIC programs. Where do we go from here?

In Book II, we will discuss two major topics: inputting data and controlling the order in which program statements are executed.

In Chapters 1 and 5, we will introduce two different ways to assign values to variables. We will show you how to enter values from your keyboard, and how to make the values an actual part of your program.

In Chapters 2, 3, and 4, we will teach you how to control the order in which program statements are executed. You will also learn how to use labels to identify program statements.

We will be using one of the programs from Book I that you saved on your diskette, so make sure your diskette is still inserted in diskette drive 1 (or diskette drive 3 if you are using a 5322(0)). If you have removed your diskette, insert it again before going on with this book.



# Chapter 1. Assigning values from the keyboard

---

## Introduction

In Book I, you learned how to assign values to variables by using the LET statement. In this chapter, you're going to learn another way to assign values to variables.

You will learn how to use the INPUT statement. The INPUT statement allows you to *input*, or enter values, from the keyboard as you run a program.

## Objectives

---

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

- Enter values for numeric variables from the keyboard by using the INPUT statement.
- Enter values for character variables from the keyboard by using the INPUT 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.

# Assigning values from the keyboard

## Assigning a numeric value

In the accumulated savings program in Book I, you assigned values to N and I with LET:

```
10 LET N=2
20 LET I=.06
30 LET A=100*(1+I)**N
40 PRINT A
50 END
```

Suppose you wanted to run the same program several times, but use different values for N and I. You could reenter the program each time, changing lines 10 and 20 to reflect the new values. But there is an easier way--use the INPUT statement.

We'll change the accumulated savings program in a few moments. But first, let's take a look at how INPUT works.

The INPUT statement allows you to assign a value to a variable from the keyboard. You can then use different values every time you run the program. For example, enter the following:

```
CLEAR
10 INPUT N
20 PRINT N
30 END
```

Now run the program:

```
RUN
```

Notice that a question mark appears on the screen.

Your System/23 is waiting for you to enter a value for N.

```
CLEAR
10 INPUT N
20 PRINT N
30 END
RUN
```

```
?_
```



---

When you enter a number, that value is assigned to the variable N. Go ahead and enter the number 1000:

```
?1000
```

```
RUN
?1000
 1000
-
```

Once you have entered a number, the PRINT statement in line 20 displays that number.

You can rerun this program as many times as you want, and enter a different value for N each time.

Let's try it again. Enter:

```
RUN
```

This time, enter a value of -250 for N:

```
?-250
```

```
RUN
?_
```

```
RUN
?-250
-250
-
```

Notice that the number -250 is not indented one space on your screen, because it is a negative number.

In these examples, you have been entering *numbers* following the question mark. You can enter only a number in response to INPUT N, because N is a numeric variable.

*Note:* If you enter a number with more than 15 digits, the extra digits will be changed to 0's. Run the program again, and enter:

```
RUN
?12347608579148423
```

```
RUN
?12347608579148423
 12347608579148400
-
```

If you enter more than 22 digits, the number will be changed to floating-point format. (See "Arithmetic data" in your *Basic Language Reference*.)

# Assigning values from the keyboard

## Assigning a character value

You can also enter a value for a character variable with INPUT. You use the same type of variable name that you used in the LET statement--a letter of the alphabet followed by zero to seven letters and/or numbers and a \$. For example, A\$ or NAME\$.

If you enter INPUT N\$, your System/23 expects you to enter a value for a character variable. For example, enter the following program. (Don't forget that you can use AUTO to number your statements if you wish. If you use the AUTO command, enter AUTO after you enter CLEAR.)

```
CLEAR
10 REM N$ STANDS FOR NAME
20 PRINT "ENTER YOUR NAME"
30 INPUT N$
40 PRINT "HELLO ";N$
50 END
```

Run this program:

```
RUN
```

Notice that the question mark appears on the screen below ENTER YOUR NAME.

Enter your name next to the question mark. (We'll use JOHN DOE--you use your name.)

```
?JOHN DOE
```

Notice that your name is displayed on the screen following HELLO. There should be a space between HELLO and your name, because a space is indicated in line 40.

```
40 PRINT "HELLO ";N$
```

```
CLEAR
10 REM N$ STANDS FOR NAME
20 PRINT "INPUT YOUR NAME"
30 ENTER N$
40 PRINT "HELLO ";N$
50 END
RUN
ENTER YOUR NAME

?_

RUN
ENTER YOUR NAME

?JOHN DOE
HELLO JOHN DOE
-
```

---

The value of N\$ is not defined within the program. You assign a value (your name) to N\$ from the keyboard. Then the PRINT statement in line 40 displays two character strings: "HELLO " and your name.

Run the program again:

RUN

This time, enter another name:

JANE DOE

As you can see, the value of N\$ can change each time you run the program.

Now look at line 20:

```
20 PRINT "ENTER YOUR NAME"
```

This statement displays a *prompt message* on the screen. A prompt is a word or words that tell you what to enter in response to an INPUT statement. In this example, the prompt tells you to enter your name when the question mark appears on the screen.

It is helpful to include prompts any time you use INPUT statements in a program. Without a prompt to remind you or whoever uses your program, you may forget what you are supposed to enter when the question mark appears.

A question mark appears on the screen whenever an INPUT statement is executed. The question mark tells you to enter something from the keyboard.

```
RUN
ENTER YOUR NAME
?_
```

```
RUN
ENTER YOUR NAME
?JANE DOE
HELLO JANE DOE
-
```

# Assigning values from the keyboard

## Assigning more than one value

Let's get back to the accumulated savings program.

Enter the following:

```
CLEAR
10 ! VALUES ENTERED FROM KEYBOARD
20 PRINT "ENTER PRINCIPAL"
30 INPUT P
40 PRINT "ENTER INTEREST RATE"
50 INPUT I
60 PRINT "ENTER NUMBER OF YEARS"
70 INPUT N
80 A=P*(1+I)**N
90 PRINT A
100 END
```

```
10 ! VALUES ENTERED FROM KEYBOARD
20 PRINT "ENTER PRINCIPAL"
30 INPUT P
40 PRINT "ENTER INTEREST RATE"
50 INPUT I
60 PRINT "ENTER NUMBER OF YEARS"
70 INPUT N
80 LET A=P*(1+I)**N
90 PRINT A
100 END
```

In this version of the program, the principal amount is a variable also.

Go ahead and run the program:

```
RUN
```

Now, enter the values of 400 for the principal, .065 for the interest rate, and 5 for the number of years:

```
ENTER PRINCIPAL
```

```
?400
```

```
ENTER INTEREST RATE
```

```
? .065
```

```
ENTER NUMBER OF YEARS
```

```
?5
```

```
RUN
ENTER PRINCIPAL

?400
ENTER INTEREST RATE

?.065
ENTER NUMBER OF YEARS

?5
548.0346653664
```



All of the variables in lines 30, 50, and 70 could be placed in one INPUT statement that looks like this:

```
70 INPUT P,I,N
```

Let's change this program so that line 70 looks like the INPUT statement shown above. We will also delete lines 30 and 50.

Enter the following:

```
70 INPUT P,I,N  
DEL 30  
DEL 50
```

```
00010 ! VALUES ENTERED FROM KEYBOARD  
00020 PRINT "ENTER PRINCIPAL"  
00040 PRINT "ENTER INTEREST RATE"  
00060 PRINT "ENTER NUMBER OF YEARS"  
00070 INPUT P,I,N  
00080 LET A=P*(1+I)**N  
00090 PRINT A  
00100 END  
-
```

List the new version of this program:

```
LIST
```

Now run the program:

```
RUN
```

```
RUN  
ENTER PRINCIPAL  
ENTER INTEREST RATE  
ENTER NUMBER OF YEARS
```

When the question mark appears, enter the following values for the principal, the interest rate, and the number of years. You must enter the values in the same order as they appear in the INPUT statement. Enter commas between the values.

```
?_
```

```
ENTER PRINCIPAL  
ENTER INTEREST RATE  
ENTER NUMBER OF YEARS
```

```
RUN  
ENTER PRINCIPAL  
ENTER INTEREST RATE  
ENTER NUMBER OF YEARS
```

```
? 250, .05, 6
```

```
?250, .05, 6  
335.02391015625  
-
```

# Assigning values from the keyboard

## Assigning more than one value (continued)

You can also combine numeric and character variables in one INPUT statement.

Enter the following:

```
CLEAR
10 PRINT "ENTER NAME AND AGE"
20 INPUT NAME$,AGE
30 PRINT NAME$,AGE
40 END
```

Now run this program:

```
RUN
```

```
CLEAR
10 PRINT "ENTER NAME AND AGE"
20 INPUT NAME$,AGE
30 PRINT NAME$,AGE
40 END
RUN
ENTER NAME AND AGE
?_
```

Enter your name and age, in that order. (Again, we will use JOHN DOE in our example.)

```
ENTER NAME AND AGE
```

```
?JOHN DOE,21
```

Notice that the 21 is displayed in the second print zone. Remember that you are displaying two values, and they are separated by a comma in the PRINT statement

```
RUN
ENTER NAME AND AGE

?JOHN DOE,21
JOHN DOE
-
```

Also notice that you must enter a value for the character variable NAME\$ first, because the character variable appears first in the INPUT statement.

Whenever there are two items in the INPUT list, you must enter two items in response to the question mark. If you only enter a value for the first variable, your System/23 will respond with an error code and flashing action code.



Go ahead and run the program again:

RUN

In response to the question mark, enter only your name (for NAMES\$):

```
RUN
ENTER NAME AND AGE

?JOHN DOE

-
RUN      ERROR 24
```

ENTER NAME AND AGE

?JOHN DOE

```
0501      20
```

As you will see in your *Messages* book, error code 0501 means that you have entered too few items for the INPUT statement. Notice also that the number 20 appears to the right of the error code 0501 on the status line. This means that the error occurred when your System/23 tried to execute line 20 - the INPUT statement.

Press the Error Reset key.

Under a line of asterisks, the bottom part of the screen should be blank. The words READY INPUT should appear.

Now, because the error occurred before the program finished running, enter:

GO END

```
*****

GO END _
READY INPUT
```

GO END tells your System/23 to end the program after you get an error code.

```
0501      20
```

# Assigning values from the keyboard

## Assigning more than one value (continued)

Now, you can run the program again. Enter:

RUN

This time, enter the following:

ENTER NAME AND AGE

? JOHN DOE, 21, 45

```
RUN
ENTER NAME AND AGE
?JOHN DOE,21,45
-
RUN      ERROR 24
                                0504      20
```

You also get an error when you enter too many values. Again, press the Error Reset key, and this time enter:

GO 10

```
GO 10
```

GO 10 tells your System/23 to continue running the program, starting at line 10.

This time, enter:

ENTER NAME AND AGE

? JOHN DOE,

```
ENTER NAME AND AGE
?JOHN DOE,
?21
JOHN DOE      21
-
```

You can see that another question mark appears. Your System/23 expected another input item after the comma. So, enter:

? 21

Remember: If you forget to enter something after the comma, your System/23 lets you enter the second item.

---

You learned in Book I that when you assign a value to a character variable by using a LET statement, you must include the value in quotation marks. For example, you would say:

```
LET N$="JOHN DOE"  OR  N$="JOHN DOE"
```

But, you do not have to enclose a value in quotation marks every time you assign the value from the keyboard by using the INPUT statement. So far, you have just been entering your name, like this:

```
?JOHN DOE
```

However, if you want to enter JOHN DOE, JUNIOR, you must enclose the name in quotation marks, like this:

```
? "JOHN DOE, JUNIOR"
```

Otherwise, because of the comma, your System/23 will assume that JUNIOR is a separate value to be assigned to another variable.

Run the program again:

```
RUN
```

Now, enter the following values:

```
ENTER NAME AND AGE
```

```
? "JOHN DOE, JUNIOR", 21
```

Any time you want to include a comma or a semicolon as part of a string, you must enclose the entire string in quotation marks.

```
RUN
ENTER NAME AND AGE

?"JOHN DOE, JUNIOR",21
JOHN DOE, JUNIOR      21
-
```

# Assigning values from the keyboard

## Assigning more than one value (continued)

### Your turn!

In Chapter 6 of Book I, you saved a program called SALESTAX. Load this program back into the work area. Because the LOAD command clears the work area before it loads a program, you do not have to enter CLEAR.

```
LOAD SALESTAX
```

```
LOAD SALESTAX
```

If you did not save a copy of this program, enter it now:

```
CLEAR
10 PRICE=600 ! ADD TAX NOW
20 TOTAL=PRICE+(PRICE*.06)
30 PRINT PRICE,TOTAL
40 END
```

Now, change line 10 so that you enter a value for PRICE from the keyboard. Also, add a statement using line number 5 that displays the prompt ENTER PRICE. Run the program and enter 600 for PRICE.

Answer: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

Here's our solution:

```
5 PRINT "ENTER PRICE"
10 INPUT PRICE
RUN
```

```
5 PRINT "ENTER PRICE"
10 INPUT PRICE
RUN
ENTER PRICE
```

```
?600
```

```
600
```

```
636
```

Now, save this program in a new file called PRICE. Please note that the file name and variable name do not need to be the same in a program.

```
SAVE PRICE//1 or SAVE PRICE//3
```

---

## Chapter summary

You can enter values from the keyboard while running a program by using the INPUT statement. INPUT assigns a value that you enter from the keyboard to a variable in a program. A question mark appears on the screen when an INPUT statement is executed.

You can input numbers, strings, or both by using one INPUT statement. INPUT statements look like this:

- A number or numbers

```
10 INPUT X
20 INPUT X,Y
```

- A string or strings

```
10 INPUT N$
20 INPUT N$,A$
```

- Numbers and strings

```
10 INPUT NAME$,AGE
20 INPUT X,X$
```

To tell you what value is expected as you run a program, you should use a prompt. A prompt is a message that is displayed on the screen. For example, here is a prompt used with an INPUT statement:

```
10 PRINT "ENTER YOUR AGE"
20 INPUT AGE
```



# Assigning values from the keyboard

---

## Exercises

### Question 1

---

Which of the following are valid entries in response to INPUT N?

- a. 55
- b. JOHN DOE
- c. -15

### Question 2

---

Which of the following are valid entries in response to INPUT A\$?

- a. 123 ELM STREET, CHICAGO
- b. "-55"
- c. 123 ELM STREET

### Question 3

---

Which of the following are valid entries in response to INPUT A\$,N?

- a. JOHN DOE,21,14
- b. JOHN DOE,21
- c. JOHN,JR.,10



---

#### Question 4

---

Change line 20 in the following program so that the value of MILES is assigned from the keyboard.

```
10 REM CONVERT MILES TO KILOMETERS
20 LET MILES=10
30 LET K=.6214*MILES
40 PRINT "THE ANSWER IS";K
50 END
```

Answer: \_\_\_\_\_

#### Question 5

---

Change line 10 in the following program so that the value of D\$ is assigned from the keyboard.

```
10 D$="07-16-79"
20 PRINT "DATE: ";D$
30 END
```

Answer: \_\_\_\_\_

# Assigning values from the keyboard

---

## Answers

### Question 1

---

- a. Valid
- b. Invalid--this is not a numeric value
- c. Valid

### Question 2

---

- a. Invalid--too many entries
- b. Valid
- c. Valid

### Question 3

---

- a. Invalid--too many entries
- b. Valid
- c. Invalid--JOHN,JR. should be enclosed in quotation marks There are too many entries. And JR. will be assigned to N which will fail because JR. is not numeric.

### Question 4

---

20 INPUT MILES

### Question 5

---

10 INPUT D\$

Remember that it is a good idea to use prompts before INPUT statements.

# Chapter 2. Changing the order of execution

---

## Introduction

In this chapter, you will learn how to use the GOTO statement to change the order in which your program statements are executed. You will also learn how to use labels to identify and locate program statements.

## Objectives

---

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

- Change the order in which statements in a program are executed by using the GOTO statement.
- Use labels in your program 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.

# Changing the order of execution

## Using the GOTO statement

In all of the programs you have entered so far, the statements have executed in ascending line number order. In the accumulated savings program:

```
10 REM VALUES ENTERED FROM KEYBOARD
20 PRINT "ENTER PRINCIPAL"
40 PRINT "ENTER INTEREST RATE"
60 PRINT "ENTER NUMBER OF YEARS"
70 INPUT P,I,N
80 LET A=P*(1+I)**N
90 PRINT A
100 END
```

line 10 is executed first, then line 20, and so forth.

With the BASIC language, you can change the order in which statements are executed by using the GOTO statement. Let's see how.

Enter the following program:

```
CLEAR
10 PRINT "SUNDAY"
20 PRINT "MONDAY"
30 GOTO 90
40 PRINT "TUESDAY"
50 PRINT "WEDNESDAY"
60 PRINT "THURSDAY"
70 PRINT "FRIDAY"
80 PRINT "SATURDAY"
90 END
```

Now run the program:

RUN

```
CLEAR
10 PRINT "SUNDAY"
20 PRINT " MONDAY"
30 GOTO 90
40 PRINT "TUESDAY"
50 PRINT "WEDNESDAY"
60 PRINT "THURSDAY"
70 PRINT "FRIDAY"
80 PRINT "SATURDAY"
90 END
RUN
SUNDAY
MONDAY
-
```

---

Only lines 10, 20, 30, and 90 are executed.

Lines 10 and 20 are executed, so SUNDAY and MONDAY are displayed. But when line 30 is executed, control *branches* to line 90. That is, statements 40 through 80 are bypassed, and control goes directly to another statement in the program. In this example, control branches to line 90, the END statement.

### **Your turn!**

---

What will be displayed on the screen when the following program is run?

```
10 PRINT "OCTOBER"  
20 GOTO 40  
30 PRINT "NOVEMBER"  
40 PRINT "DECEMBER"  
50 END
```

Answer: \_\_\_\_\_  
\_\_\_\_\_

You should have said:

```
OCTOBER  
DECEMBER
```

Line 30 is skipped, because the GOTO in line 20 sends control to line 40.



# Changing the order of execution

## Using the GOTO statement (continued)

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

```
CLEAR
10 PRINT "ENTER PRINCIPAL,RATE,YEARS"
20 INPUT P,I,N
30 LET A=P*(1+I)**N
40 PRINT "PRINCIPAL = ";P
50 PRINT "INTEREST RATE = ";I
60 PRINT "NUMBER OF YEARS = ";N
70 PRINT "SAVINGS = ";A
80 END
```

```
CLEAR
10 PRINT "ENTER PRINCIPAL,RATE,YEARS"
20 INPUT P,I,N
30 A=P*(1+I)**N
40 PRINT "PRINCIPAL = ";P
50 PRINT "INTEREST RATE = ";I
60 PRINT "NUMBER OF YEARS = ";N
70 PRINT "SAVINGS = ";A
80 END
```

Now, we are going to add a statement to the program that will cause lines 40, 50, and 60 to be skipped. Enter the following:

```
35 GOTO 70
```

```
35 GOTO 70
```

Now list the new version of your program:

```
00010 PRINT "ENTER PRINCIPAL,RATE,YEARS"
00020 INPUT P,I,N
00030 LET A=P*(1+I)**N
00035 GOTO 70
00040 PRINT "PRINCIPAL = ";P
00050 PRINT "INTEREST RATE = ";I
00060 PRINT "NUMBER OF YEARS = ";N
00070 PRINT "SAVINGS = ";A
00080 END
```

```
LIST
```

To be sure that the GOTO works, run the program:

```
RUN
```

Enter the following values:

```
ENTER PRINCIPAL,RATE,YEARS
```

```
?1000,.12,5
```

As you can see, lines 40, 50, and 60 are not executed.

```
RUN
ENTER PRINCIPAL,RATE,YEARS

?1000,.12,5
SAVINGS = 1762.3416832
```



## Using labels

You have been using line numbers with the GOTO statement to tell your System/23 which statement to execute next.

You can also use *labels* to direct the GOTO statement. A label is another way to identify or locate a statement. For instance, in the program you last entered, you could label statement 70 like this:

```
70 SAVINGS: PRINT "SAVINGS = ";A
```

Line Number      Label      Required      Program statement

A label looks like a numeric variable name. It must start with a letter and can include up to seven additional letters or numbers. The label must be followed by a colon. When you list your program, the colon will be followed by a space, whether you enter one or not.

The GOTO statement in line 35 could then be written to send control to the statement with the label SAVINGS. It would look like this:

```
35 GOTO SAVINGS
```

↑  
Label of the next  
statement to be executed

```
35 GOTO SAVINGS  
70 SAVINGS: PRINT "SAVINGS = ";A  
-
```

Go ahead and change lines 35 and 70. Enter:

```
35 GOTO SAVINGS  
70 SAVINGS: PRINT 'SAVINGS = ";A
```

Now list your program:

LIST

```
00010 PRINT "ENTER PRINCIPAL,RATE,YEARS"  
00020 INPUT P,I,N  
00030 LET A=P*(1+I)**N  
00035 GOTO SAVINGS  
00040 PRINT "PRINCIPAL = " ;P  
00050 PRINT "INTEREST RATE = " ;I  
00060 PRINT "NUMBER OF YEARS = " ;N  
00070 SAVINGS: PRINT "SAVINGS = " ;A  
00080 END  
-
```

# Changing the order of execution

## Using labels (continued)

Like remark statements, labels can help a programmer see what a program is doing. The label indicates what part of a program control is going to. For example, the label SAVINGS tells you that control is going to the statement that displays the amount of savings.

Now run the program and enter the following:

```
RUN
ENTER PRINCIPAL,RATE,YEARS
?1000,.12,5
SAVINGS = 1762.3416832
-
```

```
RUN
ENTER PRINCIPAL,RATE,YEARS
?1000,.12,5
```

As you can see, when line 35 is executed, control goes to line 70, which has the label SAVINGS.

A note about labels: You cannot use the same name for a label that you use for a numeric variable in the same program. A label can only be defined on one line. For example, the following program would cause an error at line 30:

```
10 LET TOTAL=250
20 GOTO TOTAL
30 TOTAL: PRINT "THE AMOUNT IS",TOTAL
40 END
```

Note also that you cannot use reserved system keywords such as PRINT and END as labels. For a complete list of these keywords, see "Reserved words" in your *BASIC Language Reference* manual.

There are many valuable uses for the GOTO statement, as you will learn in the following chapters. The important thing to remember is that the GOTO statement changes the order in which statements are executed. You can go to a label as many as you want.

---

## Chapter summary

You can change the order in which your program statements are executed by using the GOTO statement. GOTO causes program control to branch to, or go to, a particular statement.

You can indicate the statement you want control to branch to by entering either:

- A line number, like this:

```
10 GOTO 40  
20 PRINT  
30 PRINT  
→40 PRINT  
50 END
```

- A label, like this:

```
10 GOTO THIRD  
20 PRINT  
30 PRINT  
→40 THIRD: PRINT  
50 END
```

Labels are used to identify and locate program statements. They have names like numeric variable names. A label and a numeric variable cannot have the same name in a program. A reserved system keyword cannot be used as a label.

# Changing the order of execution

---

## Exercises

### Question 1

---

What will be displayed on the screen when you run the following program?

```
10 PRINT "PRINCIPAL = 100"  
20 GOTO 50  
30 PRINT "INTEREST = .12"  
40 PRINT "NUMBER OF YEARS = 10"  
50 END
```

Answer: \_\_\_\_\_

### Question 2

---

Which line will the control branch to in the following programs?

a. 

```
10 INPUT P  
20 INPUT R  
30 GOTO ADD  
40 PRINT P  
50 PRINT R  
60 ADD: PRINT P+R  
70 END
```

b. 

```
10 PRINT 100  
20 GOTO FINISH  
30 PRINT 100+20  
40 PRINT 100-50  
50 FINISH: END
```

Answer: a. \_\_\_\_\_

b. \_\_\_\_\_

---

## Answers

### Question 1

---

PRINCIPAL = 100

### Question 2

---

- a. from line 20 to line 60
- b. from line 20 to line 50



# Chapter 3. Branching with IF, THEN, ELSE

---

## Introduction

In this chapter, you will learn how to control the order in which your program statements are executed by using the IF statement. The IF statement transfers program control, but only when specified conditions are met.

You'll also learn how to end your programs with the STOP statement.

## Objectives

---

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

- Use IF-THEN statements to change the order in which program statements are executed.
- Use IF-THEN-ELSE statements to change the order in which program statements are executed.
- Add other BASIC statements such as PRINT, INPUT, and GOTO to previously written IF-THEN statements.
- Use the STOP statement to end a program.

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.



# Branching with IF, THEN, ELSE

---

## Using IF-THEN

In the last chapter, you learned how to use GOTO to change the order in which statements are executed.

When your System/23 executes a GOTO statement, control immediately branches to the line number (or label) listed on the GOTO statement. Your System/23 branches to the specified statement *unconditionally*, no matter what else is happening in the program.

But, suppose you want to branch to another statement only under certain conditions. In the BASIC language, the IF-THEN statement lets you do just that.

IF and THEN work together to test conditions. You cannot use one without the other.

Let's look at an example.

```
30 IF NUMBER=100 THEN GOTO 50
```

In this example, control would transfer from line 30 to line 50 if the value of NUMBER was equal to 100.

Let's look at this IF-THEN statement in a program. Enter the following program:

```
CLEAR
10 PRINT "ENTER ANY NUMBER"
20 INPUT NUMBER
30 IF NUMBER=100 THEN 50
40 PRINT "THE NUMBER IS ";NUMBER
50 END
```

```
CLEAR
10 PRINT "ENTER ANY NUMBER"
20 INPUT NUMBER
30 IF NUMBER=100 THEN 50
40 PRINT "THE NUMBER IS ";NUMBER
50 END
```

---

In this program, the value of NUMBER determines whether or not control branches to line 50.

```
30 IF NUMBER=100 THEN GOTO 50
```

If NUMBER equals 100, then control goes to line 50. If NUMBER is equal to anything except 100, control goes to line 40, the next statement in the program. Remember that NUMBER here is a numeric variable.

Let's run this program and see what happens:

RUN

When the question mark appears, enter 75 for NUMBER:

ENTER ANY NUMBER

? 75

You can see that line 40 was executed, because the number was displayed. Control did not branch from line 30 to line 50.

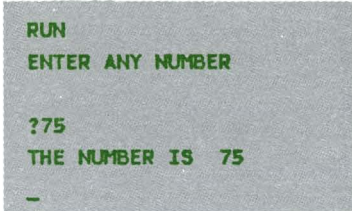
Run the program again, but enter 100 for NUMBER:

RUN

ENTER ANY NUMBER

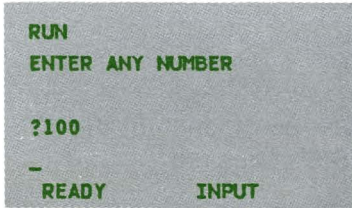
? 100

This time, line 40 wasn't executed. As soon as you entered 100, control transferred to line 50, the END statement.



```
RUN
ENTER ANY NUMBER

?75
THE NUMBER IS 75
-
```



```
RUN
ENTER ANY NUMBER

?100
-
READY      INPUT
```

# Branching with IF, THEN, ELSE

---

## Using IF-THEN (continued)

Look at the following program:

```
10 A=100
20 B=A*50
30 IF B=5000 THEN 50
40 PRINT "B DOES NOT EQUAL 5000"
50 END
```

Can you predict the order in which this program will run?  
Here's how:

Line 10  
Line 20  
Line 30  
Line 50

Line 40 will be skipped because the condition in line 30 (B = 5000) is true.

There are several conditions that can be tested with an IF-THEN statement:

= equal to (must be exactly equal)

>< or <> not equal to

> greater than

< less than

>= or => greater than or equal to

<= or =< less than or equal to

You must use the symbol (such as = or <), and not the words, to test a condition in a BASIC statement.

Here are some examples of the ways these conditions can be tested in an IF-THEN statement:

```
10 IF A > 100 THEN 50
20 IF B <> 9.5 THEN 150
30 IF C$ = "BASIC" THEN 100
```

Notice in the last example that you can test conditions for character variables.

You can also use labels in IF-THEN statements. Let's look at an example.

```
00010 PRINT "ENTER ANY NUMBER"
00020 INPUT NUMBER
00030 IF NUMBER=100 THEN 50
00040 PRINT "THE NUMBER IS ";NUMBER
00050 END
-
```

List the program currently in the work area:

LIST

Now, make these changes:

```
30 IF NUMBER=100 THEN ENOUGH
50 ENOUGH: END
-
```

```
30 IF NUMBER=100 THEN ENOUGH
50 ENOUGH: END
```

List your new program:

LIST

```
00010 PRINT "ENTER ANY NUMBER"
00020 INPUT NUMBER
00030 IF NUMBER=100 THEN ENOUGH
00040 PRINT "THE NUMBER IS ";NUMBER
00050 ENOUGH: END
-
```

And finally run the program and enter 100 for NUMBER:

RUN

ENTER ANY NUMBER

?100

```
RUN
ENTER ANY NUMBER

?100
-
READY      INPUT
```

As you can see, control again goes to line 50. However, in this example, the label ENOUGH is used to identify line 50.

# Branching with IF, THEN, ELSE

## Using IF-THEN (continued)

Enter the following:

```
CLEAR
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
```

```
CLEAR
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
```

Before you enter the RUN command, look at the different ways that this program could run.

If the value you enter for NUMBER is greater than five, control branches to line 70. After the PRINT statement in line 70 is executed, control branches to line 100, the end of the program.

```
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
```

Notice that the IF-THEN statement in line 30 uses a GOTO statement. This statement does the same thing as IF-THEN without the word GOTO. (If number > 5 then 70.)



---

## Using IF-THEN

If the value you enter for NUMBER is *not* greater than five, control of this program goes to the next statement, line 40. A condition for NUMBER is again tested. If NUMBER is less than five, control of the program goes to line 90.

```
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
```

If the value you enter for NUMBER is neither greater than nor less than five, then NUMBER = 5, and control of the program goes to line 50.

```
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
```

# Branching with IF, THEN, ELSE

## Using IF-THEN (continued)

```
RUN  
ENTER A NUMBER
```

```
?10  
NUMBER > 5  
—
```

```
RUN  
ENTER A NUMBER
```

```
?3  
NUMBER < 5  
—
```

```
RUN  
ENTER A NUMBER
```

```
?5  
NUMBER = 5  
—
```

Go ahead and run this program, entering the following value for NUMBER:

```
RUN  
ENTER A NUMBER
```

```
?10
```

Now run it again, entering the following value:

```
RUN  
ENTER A NUMBER
```

```
?3
```

And finally, run it with the following value:

```
RUN  
ENTER A NUMBER
```

```
?5
```

This program could have been written with labels instead of line numbers in the IF-THEN statements. For example, the program could look like this:

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

---

## Your turn!

---

Enter a program that will do the following:

- Input a number and assign it to the variable NUMBER
- Test whether the number is greater than or equal to one
- Display a message stating that the number is greater than or equal to one or that the number is less than one

Answer: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

Here's our solution:

```
CLEAR
10 PRINT "ENTER A NUMBER"
20 INPUT NUMBER
30 IF NUMBER >= 1 THEN GOTO 60
40 PRINT "NUMBER IS < 1"
50 GOTO 70
60 PRINT "NUMBER IS >= 1"
70 END
```

Now run your program, and enter the following:

```
RUN
ENTER A NUMBER
?0
```

```
CLEAR
10 PRINT "ENTER A NUMBER"
20 INPUT NUMBER
30 IF NUMBER >= 1 THEN GOTO 60
40 PRINT "NUMBER IS < 1"
50 GOTO 70
60 PRINT "NUMBER IS >= 1"
70 END
RUN
ENTER A NUMBER

?0
NUMBER IS < 1
-
```

# Branching with IF, THEN, ELSE

---

## Adding a statement to IF-THEN

So far, you have used line numbers and GOTO statements with IF-THEN, like this:

```
IF X > 100 THEN GOTO 50
```

IF-THEN can also be paired with other BASIC statements, like this:

```
IF X < 10 THEN PRINT "TIME TO REORDER"
```

In this example, if the value of X is less than 10, the message "TIME TO REORDER" is displayed. If the value of X is *not* less than 10, no message is displayed, and control goes to the next program statement.

Let's look at another example:

```
IF X$ = "YES" THEN INPUT DATA$
```

In this example, if the value of X\$ equals "YES", your System/23 displays a question mark on the screen and waits for you to assign a value to DATA\$ from the keyboard. If the value of X\$ does not equal "YES", the INPUT statement is *not* executed, and control goes to the next program statement.

Do you remember the program that tests to see if the value of a number is greater than, less than, or equal to 5? The program looks like this:

```
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
```

We're going to rewrite this program using a combination of IF-THEN-PRINT statements.

```
CLEAR  
10 PRINT "ENTER A NUMBER"  
20 INPUT NUMBER  
30 IF NUMBER>5 THEN PRINT "NUMBER > 5"  
40 IF NUMBER<5 THEN PRINT "NUMBER < 5"  
50 IF NUMBER=5 THEN PRINT "NUMBER = 5"  
60 END  
-
```

```
CLEAR  
10 PRINT "ENTER A NUMBER"  
20 INPUT NUMBER  
30 IF NUMBER>5 THEN PRINT "NUMBER > 5"  
40 IF NUMBER<5 THEN PRINT "NUMBER < 5"  
50 IF NUMBER=5 THEN PRINT "NUMBER = 5"  
60 END
```

Although this program has fewer statements, the two programs each accomplish the same thing. In the first program, control goes to separate PRINT statements because of the GOTO statements. In the second program, the statements are executed in ascending order and no statements are skipped.



# Branching with IF, THEN, ELSE

---

## Adding a statement to IF-THEN (continued)

### Your turn!

---

How would you change the following program so that lines 30, 40, and 50 include the PRINT statements now in lines 60, 80, and 100? (Delete any unnecessary statements.)

```
10 PRINT "ENTER A NUMBER"
20 INPUT NUMBER
30 IF NUMBER=0 THEN GOTO 60
40 IF NUMBER>0 THEN GOTO 80
50 IF NUMBER<0 THEN GOTO 100
60 PRINT "NUMBER = 0"
70 GOTO 110
80 PRINT "NUMBER > 0"
90 GOTO 110
100 PRINT "NUMBER < 0"
110 END
```

Answer: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

Here's our solution:

```
30 IF NUMBER=0 THEN PRINT "NUMBER = 0"
40 IF NUMBER>0 THEN PRINT "NUMBER > 0"
50 IF NUMBER<0 THEN PRINT "NUMBER < 0"
DEL 60,100
```

Notice that statements 60 through 100 are no longer necessary.

---

## Adding ELSE to IF-THEN

You've seen how you can include other BASIC statements in your IF-THEN statements. For example,

```
10 INPUT X
20 IF X>5 THEN PRINT X
30 IF X<=5 THEN PRINT "X<=5"
40 END
```

You can add another part to the IF-THEN statement in this program to combine statements 20 and 30. It's called ELSE.

**IF X>5 THEN PRINT X ELSE PRINT 'X<=5'**

 If this is true, then do this. Otherwise, do this.

The program could be rewritten to look like this:

```
10 INPUT X
20 IF X>5 THEN PRINT X ELSE PRINT "X<=5"
30 END
```

### **Your turn!**

---

Enter a program that will do the following:

- Input a single letter for the character variable N\$
- If N\$ is equal to "X", display N\$
- If N\$ is not equal to "X", display the number 1

Answer: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

# Branching with IF, THEN, ELSE

## Adding ELSE to IF-THEN (continued)

Here's our solution:

```
CLEAR
10 PRINT "ENTER ANY LETTER"
20 INPUT N$
30 IF N$="X" THEN PRINT N$ ELSE PRINT 1
40 END
```

Another possible solution is:

```
CLEAR
10 PRINT "ENTER ANY LETTER"
20 INPUT N$
30 IF N$="X" GOTO 60
40 PRINT 1
50 GOTO 70
60 PRINT N$
70 END
```

Run your program, and enter the letter S:

```
CLEAR
10 PRINT "ENTER ANY LETTER"
20 INPUT N$
30 IF N$="X" THEN PRINT N$ ELSE PRINT 1
40 END
RUN
ENTER ANY LETTER
```

```
RUN
ENTER ANY LETTER
```

```
?S
```

The number 1 should appear on your screen. If your program didn't run correctly, reenter the commands and statements and try it again.

There are some more ways to test conditions with the IF-THEN-ELSE statement. See "Relational expressions" in your *BASIC Language Reference* manual for information.

## Using STOP

You can choose to end your program by including the STOP statement with IF-THEN. STOP causes the program to come to an immediate end.

In the last exercise, you could end the program if N\$ does not equal 'X' by entering:

```
20 IF N$="X" THEN PRINT N$ ELSE STOP
```

STOP should not be confused with the END statement. END must be the *last* statement in a program. STOP does end a program. But it can be inserted anywhere in your program.

STOP can also be on a line by itself, like this:

```
20 IF N$="X" THEN 40
30 STOP
40 PRINT N$
```

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

```
CLEAR
10 AGE1=18
20 PRINT "ENTER YOUR AGE"
30 INPUT AGE2
40 IF AGE2<AGE1 THEN 70
50 PRINT "18 OR OVER"
60 STOP
70 PRINT "UNDER 18"
80 END
```

Now run the program, and enter *your* age:

```
RUN
ENTER YOUR AGE
```

```
?28
```

```
CLEAR
10 AGE1=18
20 PRINT "ENTER YOUR AGE"
30 INPUT AGE2
40 IF AGE2<AGE1 THEN 70
50 PRINT "18 OR OVER"
60 STOP
70 PRINT "UNDER 18"
80 END
RUN
ENTER YOUR AGE

?28
18 OR OVER
-
```

# Branching with IF, THEN, ELSE

---

## Chapter summary

You can use IF-THEN statements to change the order in which program statements are executed. Your System/23 tests a condition, and control goes to the line number or label following THEN if the condition is true. If the condition is false, control goes to the next program statement.

```
10 IF X=98.6 THEN 50
20 IF X$="A001" THEN 90
```

You can add other BASIC statements to an IF-THEN statement, like PRINT, INPUT, or GOTO. Your System/23 tests a condition and executes the statement following THEN if the condition is true.

```
10 IF X=0 THEN PRINT "REORDER"
20 IF X+Y<10 THEN INPUT Z
```

You can add ELSE to an IF-THEN statement. Your System/23 tests a condition. If the condition is true, the statement following THEN is executed. If the condition is false, the statement following ELSE is executed, or control goes to the line number or label following ELSE.

```
10 IF X=0 THEN 80 ELSE GOTO REPORT
20 IF AGE=10 THEN INPUT N$ ELSE 80
```

The STOP statement causes a program to end. STOP can be entered as a line by itself anywhere in a program. It can also be included as part of an IF-THEN statement.

```
10 IF X=100 THEN STOP
20 STOP
```



---

## Exercises

### Question 1

---

List the order in which the following programs will run if the value of N is entered as 55:

a. 10 INPUT N  
20 IF N = 100 THEN GOTO 50  
30 PRINT "N <> 100"  
40 GOTO FINISH  
50 PRINT "N = 100"  
60 FINISH: END

Answer: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

b. 10 INPUT N  
20 IF N=100 THEN PRINT N ELSE STOP  
30 END

Answer: \_\_\_\_\_  
\_\_\_\_\_

c. 10 INPUT N  
20 IF N < 235 THEN STOP  
30 PRINT 100  
40 END

Answer: \_\_\_\_\_  
\_\_\_\_\_

# Branching with IF, THEN, ELSE

---

## Exercises (continued)

### Question 2

---

Change line 20 in the following program so that if a value of 235 is entered for XYZ, the program stops.

```
10 INPUT XYZ
20 IF XYZ=235 THEN 40
30 GOTO 10
40 PRINT XYZ
50 END
```

Answer: \_\_\_\_\_

### Question 3

---

What would you enter to combine lines 30, 40, 50, and 60 of the following program into one IF-THEN-ELSE statement?

```
10 PRINT "ENTER A NUMBER"
20 INPUT N
30 IF N>=1 THEN GOTO 60
40 PRINT "N<1"
50 GOTO 70
60 PRINT N
70 END
```

Answer: \_\_\_\_\_  
\_\_\_\_\_

---

#### Question 4

---

Write a program that will do the following:

- Display a prompt and input a value for N\$
- Test to see whether N\$ = "JONES"
- Display REPORT and N\$ if N\$ = "JONES"
- End if N\$ does not equal "JONES"

You can write this program using only four BASIC statements.

Answer: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

# Branching with IF, THEN, ELSE

---

## Answers

### Question 1

---

1. The programs will run in the following order:
  - a. line 10  
line 20  
line 30  
line 40  
line 60
  - b. line 10  
line 20
  - c. line 10  
line 20

### Question 2

---

```
20 IF XYZ=235 THEN STOP  
or  
20 IF XYZ=235 THEN 50
```

### Question 3

---

```
30 IF N>=1 THEN PRINT N ELSE PRINT "N<1"  
DEL 40,60
```

### Question 4

---

```
10 PRINT "ENTER THE NAME"  
20 INPUT N$  
30 IF N$="JONES" THEN PRINT "REPORT",N$  
40 END
```

# Chapter 4. Using loops to control a program

---

## Introduction

In this chapter, you will learn to use loops to control the way your programs run. A *loop* is a series of statements that executes over and over until some condition causes control to go to another program statement.

### Objectives

---

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

- Create a loop by using IF-THEN and GOTO statements.
- Create a loop by using FOR and NEXT statements.
- Determine how many times the statements in a loop will be executed.
- Determine how many times the statements contained within *nested* loops will be executed.

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.



# Using loops to control a program

## Using an IF-THEN loop

You have seen how to use the GOTO statement to change the order in which your program statements are executed. You have also seen how to use the IF-THEN statement to change the order of execution depending on certain conditions.

There is another use for these statements. Using GOTO and IF-THEN, you can control the number of times a statement or series of statements is executed.

In the program that follows, you will use the IF-THEN statement to control the number of times your name is displayed.

Enter the following program, but use *your* name in line 20:

```
CLEAR
10 COUNTER=0
20 PRINT "JOHN DOE"
30 COUNTER=COUNTER+1
40 IF COUNTER<5 THEN 20
50 END
```

```
CLEAR
10 COUNTER=0
20 PRINT "JOHN DOE"
30 COUNTER=COUNTER+1
40 IF COUNTER<5 THEN 20
50 END
```

Before you run this program, let's take a look at some of the statements. In line 10, you *initialize*, or assign a value of 0 to the variable COUNTER. (Your System/23 automatically initializes all numeric variables to 0 when you run a program. But it's a good idea to put this statement in anyway.) This lets you set a counter for the number of times you display your name with line 20.

In line 30, you are adding a value of one to COUNTER after each time that you display your name. COUNTER then equals the new value after each addition. The variable COUNTER keeps track of the number of times your name is displayed.

---

In line 40, you are testing to see whether COUNTER is equal to five. If COUNTER is less than five, control branches back to line 20. Otherwise, the program stops.

Without running the program, can you guess how many times your name will be displayed?

Go ahead and run the program:

```
RUN
JOHN DOE
JOHN DOE
JOHN DOE
JOHN DOE
JOHN DOE
-
```

```
PRINT COUNTER
5
-
```

RUN

Your name is displayed five times. The value of COUNTER is increased by one each time your name is displayed. So, COUNTER is equal to five after the fifth time your name is displayed. Just to be sure, enter:

PRINT COUNTER

As you can see, the value of COUNTER is 5 now.

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

```
CLEAR
10 NUMBER = 50
20 PRINT "THE NUMBER IS ";NUMBER
30 GOTO 10
40 END
```

# Using loops to control a program

## Using an IF-THEN loop (continued)

Now run the program:

```
CLEAR
10 LET NUMBER = 50
20 PRINT "THE NUMBER IS ";NUMBER
30 GOTO 10
40 END
RUN
THE NUMBER IS 50
THE NUMBER IS 50
THE NUMBER IS 50
THE NUMBER IS 50
.
```

RUN

This program keeps running over and over, because the END statement can never be reached. This program is in an *endless loop*. Control never branches out of the loop. To stop the program, press Cmd Attn, and then enter:

GO END

You should avoid using endless loops like this in your programs. If you run a program that seems to take too long, or that does the same thing over and over, you should check for an endless loop.

```
*****
```

```
GO END
-
  ATTN INPUT
```

Let's try an IF-THEN loop in the accumulated savings program. You'll set up a loop that will calculate and display the amount of savings three times.

Enter the following:

```
CLEAR
10 VALUE=0
20 PRINT "ENTER PRINCIPAL,RATE,YEARS"
30 INPUT P,I,N
40 A=P*(1+I)**N
50 PRINT "SAVINGS = ";A
60 VALUE=VALUE + 1
70 IF VALUE<3 THEN GOTO 20
80 END
-
```

```
CLEAR
10 VALUE=0
20 PRINT "ENTER PRINCIPAL,RATE,YEARS"
30 INPUT P,I,N
40 A=P*(1+I)**N
50 PRINT "SAVINGS = ";A
60 VALUE=VALUE + 1
70 IF VALUE < 3 THEN GOTO 20
80 END
```

In this program, the variable VALUE is the counter. It is initialized to zero. The loop is run three times, so that you can enter three separate values for the principal, interest rate, and number of years.

```
RUN
ENTER PRINCIPAL,RATE,YEARS

?1000,.08,10
SAVINGS = 2158.92499717273
ENTER PRINCIPAL,RATE,YEARS

?1000,.09,10
SAVINGS = 2367.363674592

ENTER PRINCIPAL,RATE,YEARS
?1000,.10,10

SAVINGS = 2593.7424601
-
```

Run the program and enter the following values for P, I, and N:

```
RUN
ENTER PRINCIPAL,RATE,YEARS

?1000,.08,10
ENTER PRINCIPAL,RATE,YEARS

?1000,.09,10
ENTER PRINCIPAL,RATE,YEARS

?1000,.10,10
```

By using IF-THEN, you can control the number of times a loop is executed.



# Using loops to control a program

---

## Using a FOR-NEXT loop

You can also set up a loop with a pair of statements called FOR and NEXT. Using FOR and NEXT, you can control the number of times a statement or series of statements executes.

We'll write another program to display your name five times, but this time we'll use a FOR-NEXT loop.

Enter the following, but use *your* name in line 20:

```
CLEAR
10 FOR X=1 TO 5
20 PRINT "JOHN DOE"
30 NEXT X
40 END
```

Once again, you keep track of the number of times your name is displayed with a variable (X).

The FOR statement indicates the number of times your name is to be displayed. The NEXT statement indicates that the loop is to be executed again. The loop continues until your name has been displayed five times.

You must have a NEXT statement for every FOR statement in your program. The NEXT X statement increases the value of X and sends control back to the FOR statement.

Now, run the program:

```
RUN
```

As you can see, your name is displayed five times.

```
CLEAR
10 FOR X=1 TO 5
20 PRINT "JOHN DOE"
30 NEXT X
40 END
RUN
JOHN DOE
JOHN DOE
JOHN DOE
JOHN DOE
JOHN DOE
-
```



---

Let's take a closer look at the FOR and NEXT statements.

**FOR X=1 TO 5**

**NEXT X**

The variable X is increased by one each time the NEXT X statement is executed. When X equals 5, control goes out of the loop.

You do not have to assign a beginning value of one to the variable. For example, enter this program:

```
CLEAR
10 FOR X=2 TO 14
20 PRINT X
30 NEXT X
40 END
```

```
CLEAR
10 FOR X=2 TO 14
20 PRINT X
30 NEXT X
40 END
```

```
RUN
```

```
2
3
4
5
6
7
8
9
10
11
12
13
14
-
```

Here, the program will actually display the value of X so you can see how the value increases each time the loop is executed. Run the program:

```
RUN
```

You can also increase the value of the variable in the FOR statement by twos or threes or any other amount by using STEP.

We'll change your program to show you how STEP works. Enter the following:

```
10 FOR X=2 TO 14 STEP 2
```

# Using loops to control a program

## Using a FOR-NEXT loop (continued)

```
00010 FOR X=2 TO 14 STEP 2
00020 PRINT X
00030 NEXT X
00040 END
-
```

```
RUN
2
4
6
8
10
12
14
-
```

List your program to see what the new version looks like:

```
LIST
```

Now run the program:

```
RUN
```

Only the even numbers from 2 through 14 are displayed.

STEP 2 indicates that the value of X should increase by *two* each time the loop is run.

### Your turn!

---

Enter a program that will display all of the odd numbers from 21 through 45.

Answer: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

```
CLEAR
10 FOR X=21 TO 45 STEP 2
20 PRINT X
30 NEXT X
40 END
```

```
RUN
21
23
25
27
29
31
33
35
37
39
41
43
45
-
```

Here's our solution:

```
CLEAR
10 FOR X=21 TO 45 STEP 2
20 PRINT X
30 NEXT X
40 END
```

Now run your program:

```
RUN
```

Let's use a FOR-NEXT loop in our accumulated savings program. We'll change the program so that the principal and number of years remain the same. The interest rate will be assigned in the FOR statement.

Enter:

```
CLEAR
10 REM INTEREST RATE WILL VARY
20 P=1000
30 N=10
40 FOR I=.08 TO .10 STEP .01
50 PRINT P*(1+I)**N
60 NEXT I
70 END
```

The interest rate varies from eight to ten percent, with a STEP of .01. Therefore, the formula is calculated for interest rates of .08, .09, and .10.

Notice that the formula is included in the PRINT statement. You can include any arithmetic formula or expression in a PRINT statement and the results of the calculation will be displayed.

Go ahead and run the program:

```
CLEAR
10 REM INTEREST RATE WILL VARY
20 LET P=1000
30 LET N=10
40 FOR I=.08 TO .10 STEP .01
50 PRINT P*(1+I)**N
60 NEXT I
70 END
RUN
2158.924997273
2367.363674592
2593.7424601
-
```

RUN

*Note:* It is usually a good idea to use a variable in your FOR and NEXT statements that will not change in any other statements in the loop. If the variable in your FOR and NEXT statements is modified within the loop, it may affect loop execution.

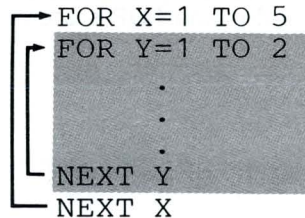
# Using loops to control a program

---

## Using nested loops

You can place one FOR-NEXT loop inside another FOR-NEXT loop, like this:

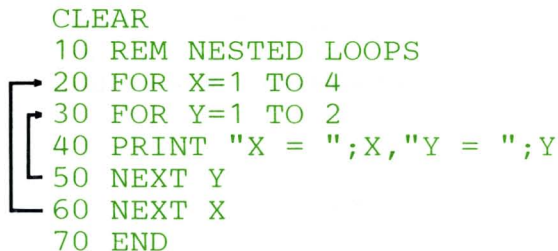
```
FOR X=1 TO 5
  FOR Y=1 TO 2
    .
    .
    .
  NEXT Y
NEXT X
```

A diagram illustrating nested loops. It shows two sets of curly braces on the left side of the code. The outer brace spans the entire code block, from 'FOR X=1 TO 5' to 'NEXT X'. The inner brace spans the inner loop, from 'FOR Y=1 TO 2' to 'NEXT Y'. This indicates that the inner loop is executed multiple times for each iteration of the outer loop.

When you place one loop inside another, you have what is called a *nested loop*. Let's take a closer look.

Enter the following:

```
CLEAR
10 REM NESTED LOOPS
20 FOR X=1 TO 4
30 FOR Y=1 TO 2
40 PRINT "X = ";X,"Y = ";Y
50 NEXT Y
60 NEXT X
70 END
```

A diagram illustrating nested loops. It shows two sets of curly braces on the left side of the code. The outer brace spans from line 20 ('FOR X=1 TO 4') to line 60 ('NEXT X'). The inner brace spans from line 30 ('FOR Y=1 TO 2') to line 50 ('NEXT Y'). This indicates that the inner loop is executed multiple times for each iteration of the outer loop.

The outer loop (beginning at line 20 and ending at line 60) runs four times. The inner loop (beginning at line 30 and ending at line 50) runs twice *every time* the outer loop runs.

The PRINT statement in line 40 displays the values of X and Y when you run the program. You can see the order in which the statements are executed by looking at the different values for X and Y. Go ahead and run the program:



```

CLEAR
10 REM NESTED LOOPS
20 FOR X=1 TO 4
30 FOR Y=1 TO 2
40 PRINT "X = ";X,"Y = ";Y
50 NEXT Y
60 NEXT X
70 END

```

```

RUN
X = 1           Y = 1
X = 1           Y = 2
X = 2           Y = 1
X = 2           Y = 2
X = 3           Y = 1
X = 3           Y = 2
X = 4           Y = 1
X = 4           Y = 2
-

```

RUN

Let's go over that one more time.

```

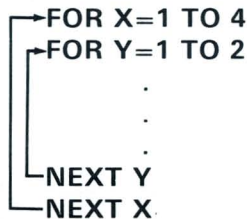
10 REM NESTED LOOPS
20 FOR X=1 TO 4
30 FOR Y=1 TO 2
40 PRINT "X = ";X,"Y = ";Y
50 NEXT Y
60 NEXT X
70 END

```

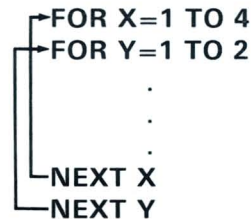
The inner loop is run twice every time the outer loop is run. Therefore, when X is equal to 1, Y is first equal to 1 and then equal to 2. Then, X is equal to 2, and Y is again equal first to 1 and then to 2. This continues until after X equals 4 and Y equals first 1 and then 2.

Did you notice that NEXT Y (line 50) comes before NEXT X (line 60)? You must place all of the inner loop (from FOR through NEXT) inside the outer loop.

**This is valid:**



**This is invalid:**



*Note:* Do not use the same variable in the FOR/NEXT statements in nested loops. If the outer loop starts with FOR X, the inner loop should *not* start with FOR X. Otherwise, you may change the execution of the outer loop.



# Using loops to control a program

## Using nested loops (continued)

Here's a way to use nested FOR-NEXT loops with the accumulated savings program. You can assign different values for the interest rate in one FOR-NEXT loop (as you did earlier in the chapter).

You can then assign different values for the number of years in another FOR-NEXT loop. The principal value of 1000 will be assigned within the program.

Enter:

```
CLEAR
10 P=1000
20 FOR I=.08 TO .10 STEP .01
30 FOR N=10 TO 12
40 PRINT I,N,P*(1+I)**N
50 NEXT N
60 NEXT I
70 END
```

```
CLEAR
10 LET P=1000
20 FOR I=.08 TO .10 STEP .01
30 FOR N=10 TO 12
40 PRINT I,N,P*(1+I)**N
50 NEXT N
60 NEXT I
70 END
```

In this program, you calculate accumulated savings for interest rates from eight to ten percent. The number of years varies from ten to twelve years.

Do you know how many times the values will be displayed?

Run the program to see if your answer is correct:

RUN

```
.08      10      2158.924997273
.08      11      2331.638997055
.08      12      2518.170116819
.09      10      2367.363674592
.09      11      2580.426405305
.09      12      2812.664781783
.1       10      2593.7424601
.1       11      2853.11670611
.1       12      3138.428376721
```

The PRINT statement is set up so that you can see exactly which values of I and N are being used.

This program is useful for calculating the accumulated savings for various interest rates and periods of time. There's one more thing that will help make the program easier to use.

Enter:

```
5 PRINT "INTEREST", "YEARS", "SAVINGS"
```

List the new version of your program:

```
00005 PRINT "INTEREST","YEARS","SAVINGS"
00010 LET P=1000
00020 FOR I=.08 TO .1 STEP .01
00030 FOR N=10 TO 12
00040 PRINT I,N,P*(1+I)**N
00050 NEXT N
00060 NEXT I
00070 END
```

LIST

You now have a heading (in line 5) to make your results easier to read. Notice that this PRINT statement is outside the loops, because you only want to display the heading once.

In the program listing, you can see that the heading uses three print zones because of the commas. Line 40 also uses the same three print zones.

Now run the program:

```
RUN
INTEREST          YEARS          SAVINGS
.08                10          2158.924997273
.08                11          2331.6389967055
.08                12          2518.170116819
.09                10          2367.363674592
.09                11          2580.426405305
.09                12          2812.664781783
.1                 10          2593.7424601
.1                 11          2853.11670611
.1                 12          3138.428376721
```

RUN

# Using loops to control a program

---

## Chapter summary

A loop is a series of statements that executes over and over until some condition causes control to go to another program statement.

You can write a loop using IF-THEN and GOTO statements, like this:

```
10 INPUT X
20 IF X=5 THEN 40
30 GOTO 10
40 PRINT X
```

You can also write a loop using FOR and NEXT statements. FOR must be the first statement, and NEXT must be the last statement in the loop, like this:

```
10 FOR NUMBER=1 TO 5
20 PRINT NUMBER
30 NEXT NUMBER
```

You can place one loop inside another loop to form nested loops. The entire inside loop (from FOR to NEXT) must be enclosed within the outside loop, like this:

```
10 FOR Y=1 TO 4
20 FOR Z=0 TO 25 STEP 5
30 PRINT Y,Z,Y*Z
40 NEXT Z
50 NEXT Y
```

The FOR statement assigns an initial and ending value to a variable and indicates an optional increase amount with STEP. The NEXT statement increases the value of the variable and sends control back to the FOR statement.

## Exercises

### Question 1

In the following programs, how many times will JOHN DOE be displayed?

a. 10 X=0  
20 PRINT "JOHN DOE"  
30 X=X+1  
40 IF X=10 THEN STOP ELSE GOTO 20  
50 END

b. 10 FOR X=1 TO 5  
20 PRINT "JOHN DOE"  
30 NEXT X  
40 END

c. 10 FOR Y=1 TO 10 STEP 2  
20 PRINT "JOHN DOE"  
30 NEXT Y  
40 END

d. 10 FOR X=1 TO 5  
20 PRINT "JOHN DOE"  
30 END

Answer: a. \_\_\_\_\_  
b. \_\_\_\_\_  
c. \_\_\_\_\_  
d. \_\_\_\_\_



# Using loops to control a program

---

## Exercises (continued)

### Question 2

---

Write a program, using a FOR-NEXT loop, that will display all multiples of three from 15 through 36.

Answer: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

### Question 3

---

What will be displayed after these programs are run?

- a. 

```
10 FOR X=10 TO 20 STEP 5
20 FOR Y=1 TO 2
30 PRINT X,Y
40 NEXT Y
50 NEXT X
60 END
```
- b. 

```
10 FOR X=100 TO 300 STEP 100
20 FOR Y=1 TO 3
30 PRINT "X = ";X,"Y = ";Y
40 NEXT X
50 END
```

Answer: a. \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

b. \_\_\_\_\_



---

## Answers

### Question 1

---

- a. Ten times
- b. Five times
- c. Five times. After the fifth time it is displayed, Y becomes 11, which is greater than 10, so the loop is not run again.
- d. It will not be displayed. An error will occur because there is no matching NEXT statement for the FOR statement.

### Question 2

---

```
10 FOR I=15 TO 36 STEP 3
20 PRINT I
30 NEXT I
40 END
```

### Question 3

---

- a. 

10	1
10	2
15	1
15	2
20	1
20	2
- b. An error message. This program will not run because there is no NEXT Y statement. The NEXT Y must come before the NEXT X statement.

Question 1

a. 1/2

b. 1/4

c. 1/8

d. 1/16

e. 1/32

f. 1/64

g. 1/128

h. 1/256

i. 1/512

j. 1/1024

k. 1/2048

l. 1/4096

m. 1/8192

n. 1/16384

o. 1/32768

p. 1/65536

# Chapter 5. Assigning values with READ and DATA

---

## Introduction

In this chapter, you will learn about two more statements: READ and DATA. The READ and DATA statements work together to assign values to variables.

You will also learn about the RESTORE statement. RESTORE is used with READ and DATA to assign the same values to different variables.

### Objective

---

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

- Assign the values in a DATA statement to the variables in a READ statement.
- Write a program in which values are assigned with READ and DATA.
- Use the RESTORE statement with READ and DATA to change the order in which values are assigned.

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.

# Assigning values with READ and DATA

## Using READ and DATA statements

So far, you have learned two methods for assigning values to variables. In Book I, you learned how to use the LET statement, like this:

```
10 LET X=250 or 10 X=250
```

In Chapter 1 of Book II, you learned how to use the INPUT statement, like this:

```
10 INPUT X
```

In this chapter, we will show you one more way to assign values. The READ and DATA statements work together to assign values that are an actual part of your program.

Let's take a look. Enter the following:

```
CLEAR  
10 DATA 5,10,15  
20 READ A,B,C  
30 PRINT A;B;C  
40 END
```

Can you guess what's going on here? The READ and DATA statements work together to assign values to variables. The values listed in the DATA statement are assigned to the variables listed in the READ statement.

```
DATA 5,10,15  
      ↓ ↓ ↓  
READ A,B,C
```

The values in the DATA statement are assigned in the same order to the variables listed in the READ statement. Once the READ and DATA statements are executed, A is equal to 5, B is equal to 10, and C is equal to 15.

```
CLEAR
10 DATA 5,10,15
20 READ A,B,C
30 PRINT A;B;C
40 END
RUN
5 10 15
-
```

```
00010 READ A,B,C
00020 DATA 5,10,15
00030 PRINT A;B;C
00040 END
-
```

```
RUN
5 10 15
-
```

Go ahead and run the program:

RUN

The DATA statement does not have to come before the READ statement. It can be placed anywhere in a program.

Just to be sure, let's change the program.

Enter:

```
10 READ A,B,C
20 DATA 5,10,15
```

List the new version of your program:

LIST

Now run the program:

RUN

As you can see, the results are the same. In fact, you can place the DATA statements anywhere and in any order in the program.

```
10 DATA 5
20 READ A,B
30 READ C
40 DATA 10,15
50 PRINT A;B;C
60 END
```

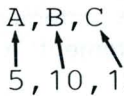
The first DATA value (5) is assigned to the first READ variable (A). The second value is assigned to the second variable, and the third value is assigned to the third variable.

# Assigning values with READ and DATA

## Using READ and DATA statements (continued)

If you list more values in the DATA statements than there are variables in the READ statements, the extra DATA values are not used. For example,

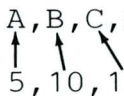
```
READ A,B,C
DATA 5,10,15,20,25
```



In this case, the last two values in the DATA statement would not be used.

But, if you have too few values to assign to the variables in the READ statement, you will get an error. For example,

```
READ A,B,C,D,E
DATA 5,10,15
```



In this case, if no other DATA values exist in the program, an error 0054 will occur. Look in your *messages* manual for error code 0054. You must have a DATA value for every READ variable listed.

Let's try it, and see. Enter the following:

```
CLEAR
10 DATA 1000,2000,3000
20 READ A,B,C,D,E
30 PRINT A,B,C
40 END
```

Now run the program:

```
RUN
```

```
10 DATA 1000,2000,3000
20 READ A,B,C,D,E
30 PRINT A,B,C
40 END
RUN
```

```
RUN ERROR 94
```

```
0054 20
```



To get rid of the error code and flashing action code, press Error Reset, and because the program hasn't finished, enter:

```
*****
GO END _
READY      INPUT
```

GO END

You can assign values to character variables with READ and DATA, too.

List the program currently in the work area:

```
00010 DATA 1000,2000,3000
00020 READ A,B,C,D,E
00030 PRINT A,B,C
00040 END
-
```

LIST

Now, add these changes:

```
15 DATA "SMITH","JONES","BROWN"
20 READ A,B,C
25 READ A$,B$,C$
35 PRINT A$,B$,C$
-
```

```
15 DATA "SMITH","JONES","BROWN"
20 READ A,B,C
25 READ A$,B$,C$
35 PRINT A$,B$,C$
```

Notice that we corrected line 20 to read the correct number of values from line 10. Now list the new version:

```
00010 DATA 1000,2000,3000
00015 DATA "SMITH","JONES","BROWN"
00020 READ A,B,C
00025 READ A$,B$,C$
00030 PRINT A,B,C
00035 PRINT A$,B$,C$
00040 END
-
```

LIST

When you run this program, the values listed in the DATA statements will be assigned to the variables listed in the READ statements. Try it. Enter:

```
RUN
1000          2000          3000
SMITH        JONES        BROWN
-
```

RUN

The values in the DATA statements were assigned in the same order as the variables in the READ statements.

# Assigning values with READ and DATA

---

## Using READ and DATA statements (continued)

Look again at lines 10 through 25:

```
10 DATA 1000,2000,3000
14 DATA "SMITH", "JONES", "BROWN"
20 READ A,B,C
25 READ A$,B$,C$
```

The type of values in the DATA statements must be in the same order as the type of variables in the READ statements. That is, you must always assign character values to character variables and numeric values to numeric variables. The number 1000 is assigned to A. The name SMITH is assigned to A\$.

However, when you place character values (strings) in your DATA statements, you do not have to use quotation marks. You can enter your strings just as you would for an INPUT statement. Line 15 could look like this:

```
15 DATA SMITH,JONES,BROWN
```

But remember: If you want to use a comma or a semicolon in a string, the entire string must be enclosed in quotation marks. Also, if you want the string to appear in lowercase, you must enclose the string in quotation marks.

One more note: If you read a number with a character variable, the number will be treated as a string. For example, if line 15 looked like this:

```
15 DATA "SMITH", 27, "BROWN"
```

then the value of B\$ would be the string "27".

## Your turn!

Using READ and DATA statements, enter a program that:

- Assigns 1000 to the principal (P), .12 to the interest rate (I), and 5 to the number of years (N)
- Calculates the amount of accumulated savings
- Displays the results

Hint: In case you've forgotten, the formula is  $P*(1+I)**N$ .

Answer:

---

---

---

---

---

Here's our solution:

```
CLEAR
10 DATA 1000,.12,5
20 READ P,I,N
30 PRINT "SAVINGS=";P*(1+I)**N
40 END
```

To make sure it works, run your program:

```
CLEAR
10 DATA 1000,.12,5
20 READ P,I,N
30 PRINT "SAVINGS = ",P*(1+I)**N
40 END
RUN
SAVINGS = 1762.3416832
-
```

RUN

If your program didn't work, reenter the commands and statements shown above, and try it again. Remember that if your program stops with an error, you will have to press the Error Reset, and enter GO END.

# Assigning values with READ and DATA

---

## Using READ and DATA statements (continued)

Remember: You can assign both character and numeric values with READ and DATA statements. But, the DATA values must be in the same order as the READ variables:

```
DATA 10,20,"XYZ",5,"ZYX"  
READ N1,N2,N1$,N3,N2$
```

OR

```
READ N1,N2  
DATA 10,20  
DATA "XYZ",5,"ZYX"  
READ N1$,N3,N2$
```

You can read a number as a string if the variable in the READ statement is a character variable. The number will then be treated as a string, not as a number.

*Note:* You can not include a remark on a DATA statement.



## Using the RESTORE statement

There is a statement that you can use with READ and DATA. It's called RESTORE. With RESTORE, you can assign the same values to different sets of variables.

Here's how it works:

```
DATA 10,20,30
READ A,B,C
RESTORE
READ D,E,F
```

The values 10, 20, and 30 are assigned to variables A, B, and C. The RESTORE statement tells your System/23 to go back to the beginning of the values listed in the DATA statement, and assign those values to the next set of READ variables.

Let's try this in a program. Enter:

```
CLEAR
10 REM RESTORE EXAMPLE
20 DATA 10,20,30
30 READ NUMBER1,NUMBER2,NUMBER3
40 RESTORE
50 READ NUMBER4,NUMBER5,NUMBER6
60 PRINT NUMBER1;NUMBER2;NUMBER3
70 PRINT NUMBER4;NUMBER5;NUMBER6
80 END
```

Now run the program:

```
RUN
```

The values 10, 20, and 30 are assigned to NUMBER4, NUMBER5, and NUMBER6 as well as NUMBER1, NUMBER2, and NUMBER3.

```
CLEAR
10 REM RESTORE EXAMPLE
20 DATA 10,20,30
30 READ NUMBER1,NUMBER2,NUMBER3
40 RESTORE
50 READ NUMBER4,NUMBER5,NUMBER6
60 PRINT NUMBER1;NUMBER2;NUMBER3
70 PRINT NUMBER4;NUMBER5;NUMBER6
80 END
RUN
 10 20 30
 10 20 30
-
```



# Assigning values with READ and DATA

---

## Using the RESTORE statement (continued)

Look at this program:

```
10 DATA 15,20,25
20 READ A
30 RESTORE
40 READ B
50 RESTORE
60 READ C
70 PRINT A,B,C
80 END
```

Can you guess what values would be displayed for A, B, and C if this program were run?

A, B, C would all be equal to 15. The RESTORE statements in lines 30 and 50 indicate that the value at the beginning of the DATA list should be assigned to the next variable read.

What would happen if we deleted line 50?

The variables A and B would both be equal to 15. However, the variable C would then be equal to 20.

What would happen if we deleted *both* lines 50 and 60?

The variables A and B would both still be equal to 15. However, since the variable C would not be assigned a value, its value would be zero.

Remember from Book I: If you do not assign a value to a numeric variable in a program, its value is zero.

## Your turn!

Enter a program that will do the following:

- Using READ and DATA, assign the names "SMITH" and "JONES" to the variables NAME1\$ and NAME2\$.
- Using RESTORE, reassign those same names to the variables A\$ and B\$.
- Display NAME1\$, NAME2\$, A\$, and B\$ on the screen.

Answer:

---

---

---

---

---

---

Here's our solution:

```
CLEAR
10 DATA "SMITH","JONES"
20 READ NAME1$,NAME2$
30 RESTORE
40 READ A$,B$
50 PRINT NAME1$,NAME2$,A$,B$
60 END
```

```
CLEAR
10 DATA "SMITH","JONES"
20 READ NAME1$,NAME2$
30 RESTORE
40 READ A$,B$
50 PRINT NAME1$,NAME2$,A$,B$
60 END
RUN
SMITH                JONES                SMITH                JONES
```

Run your program:

```
RUN
```

# Assigning values with READ and DATA

---

## Chapter summary

You can assign values to variables in a program by using the READ and DATA statements. Values that are listed in DATA statements are assigned to variables that are listed in READ statements. For example,

```
10 DATA 1,2,3
20 READ X,Y,Z
30 DATA "FRANK","JOHN","JOE"
40 READ X$,Y$,Z$
```

The RESTORE statement causes the next READ statement to assign values beginning with the first value listed in the DATA statements. For example,

```
10 DATA 1,2,3
20 READ X,Y,Z
30 RESTORE
40 READ A,B,C
```

## Exercises

### Question 1

What will be displayed on the screen when the following programs are run?

- a. 10 DATA 100,200,300  
20 READ A,B,C  
30 PRINT A;B;C  
40 END
  
- b. 10 READ A,A\$,B  
20 DATA 10,"INVENTORY",25  
30 PRINT A\$  
40 PRINT A,B  
50 END
  
- c. 10 READ A\$,B\$,C  
20 DATA "INVENTORY"  
30 DATA "PAYROLL"  
40 DATA 500  
50 PRINT A\$,B\$  
60 END
  
- d. 10 DATA "INVENTORY"  
20 DATA "PAYROLL","BILLING"  
30 READ A\$,B\$,C  
40 PRINT A\$,B\$,C  
50 END

Answer: a. \_\_\_\_\_  
b. \_\_\_\_\_  
c. \_\_\_\_\_  
d. \_\_\_\_\_

# Assigning values with READ and DATA

---

## Exercises (continued)

### Question 2

---

Write a program that will do the following:

- Assign the values 100, 200, and 300 to the variables A, B, and C by using READ and DATA statements
- Display the values five times by using a FOR-NEXT loop

Answer: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_



### Question 3

---

Using the RESTORE statement, do the following:

- a. Assign the values 5, 6, and 7 to the variables D, E, and F:

```
10 DATA 5,6,7
20 READ A,B,C
40 READ D,E,F
50 END
```

- b. Assign the name SMITH to the variable PERSON\$:

```
10 DATA "SMITH", "JONES"
20 READ NAME$
40 READ PERSON$
50 END
```

Answer: a. \_\_\_\_\_

b. \_\_\_\_\_

# Assigning values with READ and DATA

---

## Answers

### Question 1

---

- a. 100 200 300
- b. INVENTORY  
10 25
- c. INVENTORY PAYROLL
- d. An error code. The value "BILLINGS" cannot be assigned to the numeric variable C.

### Question 2

---

```
10 DATA 100,200,300
20 READ A,B,C
30 FOR X=1 TO 5
40 PRINT A,B,C
50 NEXT X
60 END
```

### Question 3

---

- a. 30 RESTORE
- b. 30 RESTORE

# READER'S COMMENT FORM

SA34-0122-0

## II. Inputting Data and Using Loops

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
------	---------

Cut or Fold Along Line

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

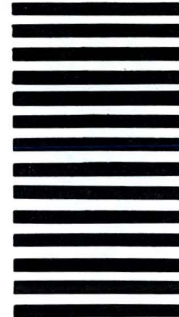


NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

**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









SA34-0122-0  
Printed in U.S.A.