

# I. Getting Started with BASIC

IBM

Learning System/23 BASIC

# I. Getting Started with BASIC



**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.

# Preface

---

## The BASIC language

*Learning System/23 BASIC* is a self-study course in which you will learn about a programming language called BASIC. BASIC stands for Beginner's All-purpose Symbolic Instruction Code.

Just what is a *programming language*? It is your means of communicating with the computer. BASIC uses English-language words (such as PRINT and READ) to tell the computer what you want it to do. In this course, we'll tell you how to put these words together to create *programs*. A program is a series of statements that tell the computer what to do.

If you've had no programming training (if you are a *beginner*), then this course is for you!

By the way, even if you're *not* a beginner, this course can help you. System/23 BASIC differs somewhat from other BASICs with which you may be familiar. This course should help point out some of the differences.

We'll move slowly in talking about the BASIC language, but we'll cover most of what you can do with BASIC on your System/23.

# Preface

---

## Course structure

The complete course is made up of seven separate books. You should proceed through each book at your own pace so that you learn at a speed that is comfortable to you.

The first book is the most elementary, and each book that follows deals with increasingly complex material. You should start this course with Book I, and complete each of the rest of the books in order. A complete course index is located at the end of Book VII.

Each book in this course is divided into chapters. At the beginning of each chapter, there is a brief description of what's going to be presented in the chapter. At the end of each chapter, there are exercises that test your knowledge of the material that was presented. When you finish a chapter, complete the exercises located at the end of the chapter. By completing these exercises, you'll be confident that you understand the material in one chapter before moving on to the next.

If you've had some BASIC programming experience, and you are familiar with the material to be presented, you can check your knowledge by completing the exercises. If you successfully complete the exercises, you may choose to skip over a chapter and move on to the next.

For the best results, you should have a System/23 available while you are taking this course, so that you can actually see the results of your programming efforts.

If you don't have a System/23 available, you can still learn the fundamentals of BASIC. Throughout this course, we have provided illustrations of the screen and printed pages to show you what would appear as a result of your actions if you were actually using your System/23.

---

If you do have a System/23, you should remember that when information is highlighted in green, like this:

### INFORMATION

it means you should type that information on your System/23 keyboard.

What you typed  
System/23 response

A screen of what you typed and the response made by the computer will be shown in the left-hand column. By looking at your screen, you can see if you typed the information correctly. We will not show the status line in all of our examples, but remember that you will always have something displayed on the status line, line 24, of your screen.

### BEFORE YOU BEGIN!

You should have completed the training course *Learning to Use System/23* and should know how to operate your System/23 before you start *Learning System/23 BASIC*.

We assume, for example, that you are familiar with the purpose and function of the keys and switches on your System/23. You should also know how to correct typing errors and how to interpret and respond to error codes.

If you are not familiar with these items, you should refer to the *Operator Reference* manual, SA34-0108.

# Preface

---

## Course structure (continued)

One more thing...

From time to time, you will see references to the *BASIC Language Reference* manual. The *BASIC Language Reference* manual was included in your System/23 package at delivery. If you do not have a copy, you can order one from your IBM Representative. The form number is SA34-0109.

The *BASIC Language Reference* manual was designed to provide detailed reference information about the System/23 BASIC language. The manual is written in encyclopedia format. That is, statements, commands, data constants, variables, and concepts are all entered in alphabetic order.

After studying a programming concept in *Learning System/23 BASIC*, you may wish to refer to the *BASIC Language Reference* manual for more detailed information. For example, in Book I of this course, you will learn about the PRINT statement. If you want more information about PRINT, you should look in the *BASIC Language Reference* manual under "PRINT statement."

What else will you need to complete this course?

- A System/23 *Messages* manual, form number  
As you complete the exercises in this course, you may encounter errors. When an error occurs, you should look in your *Messages* manual for the action code and error codes.
- A prepared diskette with System/23 format. You learned how to prepare a diskette in *Learning to Use System/23*. For more information, see "Chapter 2. PREPARE" in *Operator Customer Support Functions*,

---

Volume I. Your diskette shouldn't have any files that may be required for later use.

- A flowcharting template (optional). A template is a plastic or wooden form, which contains patterns of standard flowcharting symbols. You can purchase a template at your computer supply center. Or, when you reach Chapter 2 in Book VII, where we ask you to draw flowcharts, you can simply draw the symbols in by hand.

Please note: While you are taking this course, you will often be told to copy information onto your prepared diskette. You should insert your diskette into drive 1. Or, if you are using a processor without a built-in diskette unit, you should insert your diskette into drive 3 on your 5246 Diskette Unit. (We will refer to this type of processor as a 5322(0) in this course.)

Also note: Most of the examples in this course are presented in uppercase letters. However, you can use either uppercase or lowercase letters in BASIC. Your System/23 will accept either one. Just remember, though, that when your System/23 is trying to tell you something, the message will always be displayed in uppercase letters.

OK? Let's begin.



# I. Getting started with BASIC

---

## Contents

<b>About this book</b> .....	x
<b>Chapter 1. The PRINT statement</b> .....	1-1
Introduction.....	1-1
Displaying numbers.....	1-2
Displaying words.....	1-4
Displaying more than one number or string.....	1-6
Chapter summary.....	1-12
Exercises.....	1-13
Answers.....	1-15
<b>Chapter 2. Writing a program</b> .....	2-1
Introduction.....	2-1
Entering a program .....	2-2
Assigning line numbers.....	2-5
Listing your program.....	2-10
Chapter summary.....	2-12
Exercises.....	2-13
Answers.....	2-14
<b>Chapter 3. Changing a program</b> .....	3-1
Introduction.....	3-1
Changing line numbers .....	3-2
Adding statements.....	3-6
Deleting statements .....	3-9
Replacing statements.....	3-11
Chapter summary.....	3-14
Exercises.....	3-15
Answers.....	3-17
<b>Chapter 4. Performing arithmetic</b> .....	4-1
Introduction.....	4-1
Adding and subtracting.....	4-2
Multiplying and dividing.....	4-3
Raising a number to a power.....	4-4
Arithmetic order.....	4-6

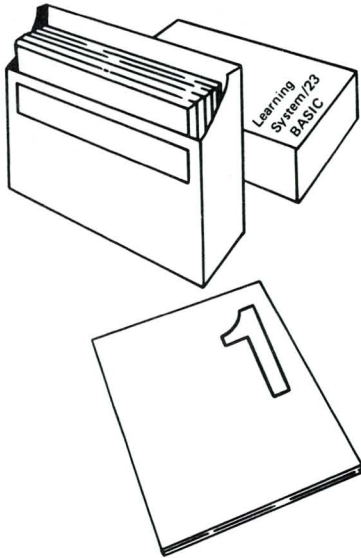
---

Chapter summary.....	4-8
Exercises.....	4-9
Answers.....	4-11
<b>Chapter 5. Using variables and remarks.....</b>	<b>5-1</b>
Introduction.....	5-1
Numeric variables.....	5-2
Character variables.....	5-6
Remark statements.....	5-10
Chapter summary.....	5-13
Exercises.....	5-14
Answers.....	5-16
<b>Chapter 6. Using diskette storage.....</b>	<b>6-1</b>
Introduction.....	6-1
Saving a program.....	6-2
Loading a program.....	6-5
Replacing a program.....	6-7
Chapter summary.....	6-9
Exercises.....	6-11
Answers.....	6-12

# I. Getting started with BASIC

---

## About this book



This is Book I in your series of seven books on *Learning System/23 BASIC*. In this book, you will get your first chance to write a program. Where do we start?

First, we will show you how to display numbers and words on your screen. Then, you will write and run your first program.

We will show you how to modify a program and how to list it on the screen. You will also see how to perform arithmetic on your System/23 and how to use variables and remark statements.

Finally, we will show you how to save a program on a diskette so that you can use the program again without reentering it.

You may think all of this sounds like a lot to learn, but just stay with us, and you won't have any trouble.

# Chapter 1. The PRINT statement

---

## Introduction

In this chapter, you will learn about a BASIC statement called PRINT.

A *statement* is a meaningful expression or instruction in the BASIC language. A statement allows you to enter information into your System/23, specify what is to be done with that information, and determine what the outcome should be.

The PRINT statement tells your System/23 to display information on the screen. We'll show you several examples of the PRINT statement on the following pages.

## **Objectives**

---

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

- Write a PRINT statement that will cause a number to be displayed.
- Write a PRINT statement that will cause words to be displayed.
- Write a PRINT statement that will cause a combination of numbers and words to be displayed.
- Respond to action and error codes and reenter the corrected information.

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.

# The PRINT statement

## Displaying numbers

Before you learn about the PRINT statement, make sure your System/23 is ready to begin. Don't forget to switch on your printer and your diskette unit, if you have one.

The words READY INPUT should be displayed on the *status line*, which is the bottom line of the screen. If you can't remember how to start up your System/23, see "Starting up the system" in your *Operator Reference* manual.

```
-  
READY INPUT
```

All set? Let's get started.

On your keyboard, type the following:

```
PRINT 250
```

Notice that as you press each key, the character on that key appears on the screen. In this course, a *character* means a number (0-9), a letter (A-Z or a-z), a blank, or a special character (such as ! or ,). For a complete list of available characters, turn to "BASIC character set" and "Character set representation" in your *BASIC Language Reference*.

```
PRINT 250  
 250  
-
```

Now, press the Enter key and watch what happens.

The number 250 appears on the line directly below the PRINT statement you just typed. The word PRINT tells your System/23 that you want to display something. The information following the word PRINT (in this case, the number 250) is what you want to display.

Notice that the 250 is indented one space to the right on the screen. That extra space is reserved for a minus sign (-), which is displayed if the number is negative. You can't tell by looking at this example, but your System/23 also leaves a blank space *after* a displayed number.

---

On your System/23, a number is positive unless you indicate otherwise. You do not have to type a plus sign (+) to indicate that a number is positive. Even if you type a number with a plus sign, the + will not be displayed.

For example, type the following:

```
PRINT +250
```

and press the Enter key.

Notice that the number 250 is displayed without the + sign.

Now let's look at negative numbers. If you want to display a negative number, you *must* type a minus sign (-) to indicate that the number is negative.

Type the following:

```
PRINT -250
```

and press the Enter key.

See how the minus sign is displayed in the first position on the line?

*Remember:*

You do not have to type a plus sign (+) for positive numbers.

You must type a minus sign (-) for negative numbers.

A positive number will be displayed *without* a + sign. A negative number will be displayed *with* a - sign.

```
PRINT +250  
250  
—
```

```
PRINT -250  
-250  
—
```

# The PRINT statement

---

## Displaying words

Now let's try displaying some words on the screen. Type the following:

```
PRINT "THIS IS EASY"
```

and press the Enter key.

"THIS IS EASY" is called a *string*. A string is a series of characters, enclosed in quotation marks. A string can be made up of letters, numbers, blanks, special characters, or any combination of these.

Whenever you want to display a string by using a PRINT statement, you *must* enclose the string in quotation marks. But remember, you do not need to use quotation marks if you are displaying only numbers.

Before we ask you to display some more strings with the PRINT statement, there is one more thing you should know. So far, we've been telling you to press the Enter key every time you type a statement on your keyboard. From now on, when we tell you to "enter" a statement, you should type the statement and then press the Enter key.

For example, when we ask you to enter:

```
PRINT "THIS IS EASY"
```

you should:

1. Type PRINT "THIS IS EASY"
2. Press the Enter key

OK? Back to strings.

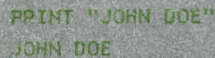


```
PRINT "THIS IS EASY"  
THIS IS EASY  
—
```

---

Enter a PRINT statement to display your name. We'll use JOHN DOE in our example--you use *your* name.

```
PRINT "JOHN DOE"
```



```
PRINT "JOHN DOE"  
JOHN DOE
```

Notice that your name is *not* indented one position on the screen. No space is reserved for a - when a string is displayed. Now, just to prove that you really do need those quotation marks around a string, try displaying your name (first and last name) without the quotation marks. Enter:

```
PRINT JOHN DOE
```



```
PRINT JOHN DOE  
READY ERROR 94
```

1000

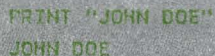
### DON'T PANIC!

The 4-digit error code and 2-digit flashing action code indicate that you made a mistake. Your System/23 doesn't understand what you want it to do.

Look in your *Messages* manual for action code 94 and error code 1000. Action code 94 tells you that you have entered incorrect information. Error code 1000 says you have an invalid statement. In this case, you forgot the quotes.

To stop the flashing, press the Error Reset key. Now, locate the scroll up key (⌆) and press it once. This gives you a blank line on which to enter your statement correctly. So, enter:

```
PRINT "JOHN DOE"
```



```
PRINT "JOHN DOE"  
JOHN DOE
```

See, it works when you include the quotation marks.



# The PRINT statement

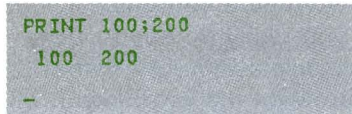
---

## Displaying more than one number or string

You can use one PRINT statement to display several numbers, strings, or numbers and strings. Let's see how.

We'll start with numbers. Enter the following:

```
PRINT 100;200
```

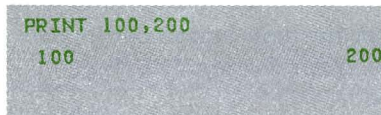


```
PRINT 100;200
100 200
-
```

The numbers 100 and 200 are displayed with two spaces between them. One space is reserved for a minus sign before each number, in case the number is negative. The other space is always used when you are displaying a number. Remember: A number is always followed by a blank.

Now enter the two numbers with a comma between them:

```
PRINT 100,200
```



```
PRINT 100,200
100
200
```

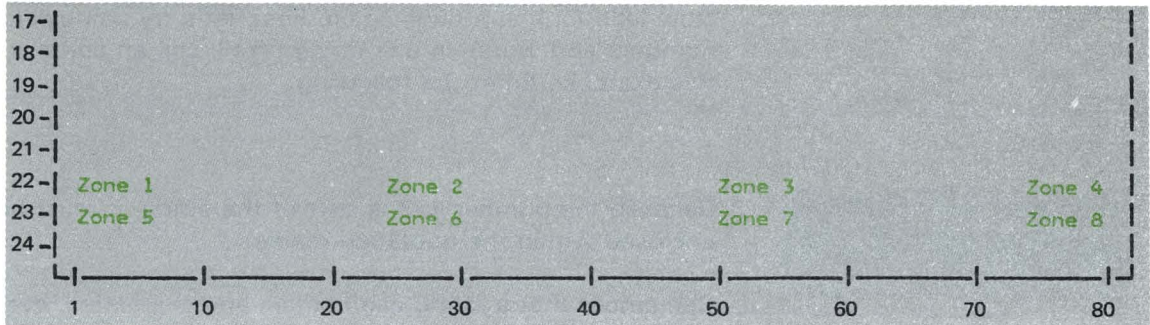
Notice that the 200 is displayed several spaces to the right of the 100. Here's why:

Each line on the screen is 80 characters wide and is divided into areas called *print zones*. The first three print zones on each line are 24 characters wide. The print zone that begins in character position 73 is only 8 characters wide.

When you enter a comma between the numbers or strings in a PRINT statement, the second number or string begins in the next print zone.

Let's look at this more closely.

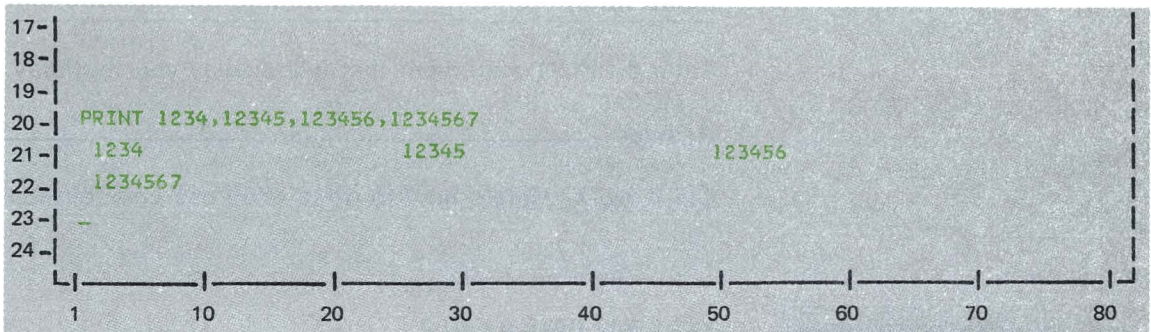
On the following page, you will see an illustration of the screen. We have indicated eight print zones.



What happens if you try to display a string with more than eight characters, or a number with more than six digits, in zone 4? What you are displaying will begin in zone 5.

Let's try it. Enter:

```
PRINT 1234,12345,123456,1234567
```



Remember: Numbers are displayed with a blank before (reserved for a minus sign) and after the number.

# The PRINT statement

---

## Displaying more than one number or string (continued)

Now let's look at strings again. First, let's try combining numbers and words in one string. We'll use an address for this example. Enter the following:

```
PRINT "123 ELM STREET"
```

```
PRINT "123 ELM STREET"  
123 ELM STREET  
-
```

Because the number 123 is part of the string, it must be enclosed within the quotation marks.

Remember that a string can contain any characters, even commas. For example, enter the following:

```
PRINT "CHICAGO, IL"
```

```
PRINT "CHICAGO, IL"  
CHICAGO, IL  
-
```

Notice that the string CHICAGO, IL is displayed in only one print zone. The comma does not divide the information into print zones; the comma is part of the string.

### Your turn!

---

Enter a PRINT statement that will display your birthday.

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

Did it work? Here's how to do it. (You use *your* birthday.)

```
PRINT "JANUARY 4, 1952"
```

```
PRINT "JANUARY 4, 1952"  
JANUARY 4, 1952  
-
```

We could have entered

```
PRINT "1-4-52" or PRINT "1/4/52"
```

The important thing to remember is that the quotation marks are required when you display a string.

---

You can also display more than one string with a single PRINT statement. Look at the following two PRINT statements that display a name and address:

```
PRINT "JOHN DOE";"123 ELM"  
PRINT "JOHN DOE","123 ELM"
```

In these two examples, the name and address are two separate strings. In the first PRINT statement, a semicolon separates the name and address. In the second PRINT statement, a comma separates the name and address.

Enter the first PRINT statement:

```
PRINT "JOHN DOE";"123 ELM"
```

```
PRINT "JOHN DOE";"123 ELM"  
JOHN DOE123 ELM  
-
```

As you can see, the two strings are displayed with no space between them. When you display strings with a PRINT statement, the semicolon indicates that no space should be left between the two strings on the screen.

You *can* insert a blank space within the quotation marks, after the E in DOE, to leave a space between the strings. For example, enter:

```
PRINT "JOHN DOE ";"123 ELM"
```

```
PRINT "JOHN DOE ";"123 ELM"  
JOHN DOE 123 ELM  
-
```

Now, enter the two strings with a comma between them:

```
PRINT "JOHN DOE","123 ELM"
```

```
PRINT "JOHN DOE","123 ELM"  
JOHN DOE          123 ELM  
-
```

Notice that the address is displayed several spaces to the right of the name. Since the address is the second string, and the two strings are separated by a comma, the address is displayed in the second print zone. These print zones are just like the print zones used for numbers.

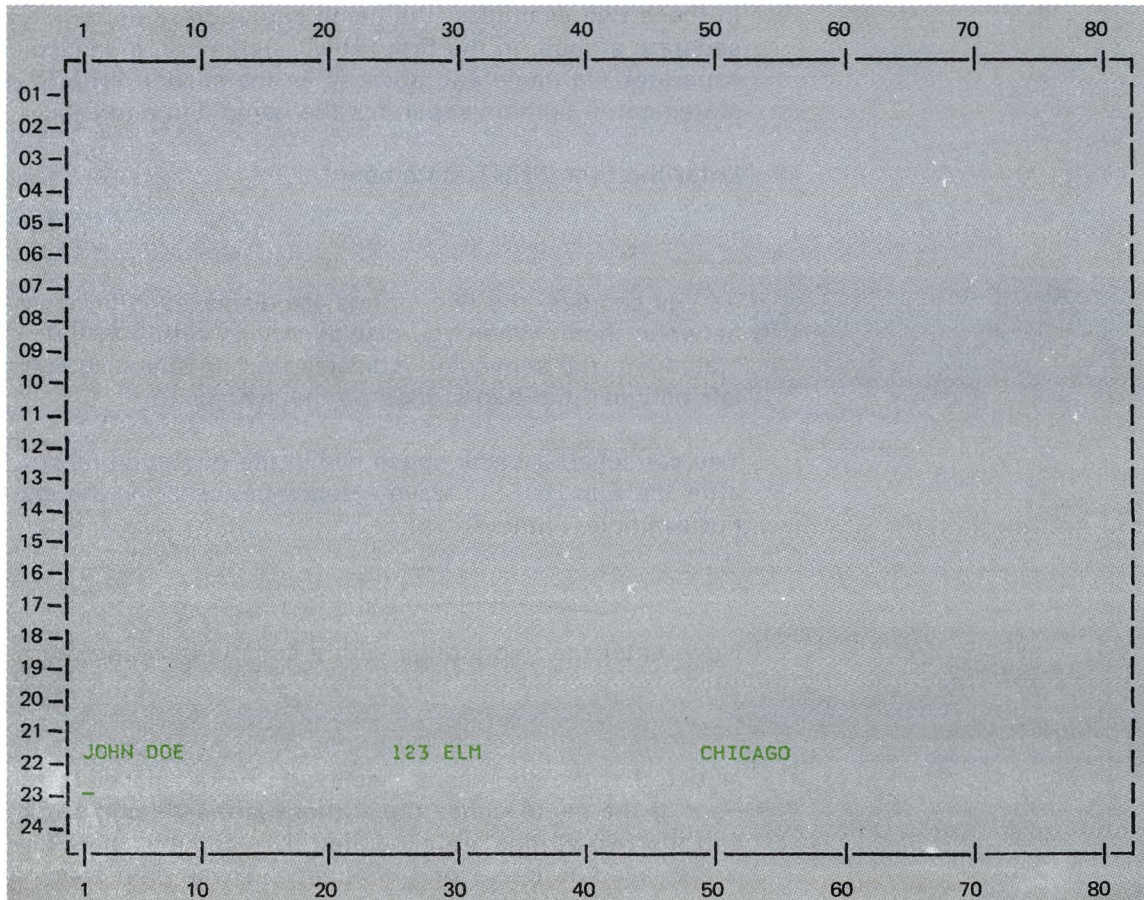
# The PRINT statement

## Displaying more than one number or string (continued)

What if we added a third string to the PRINT statement?

```
PRINT "JOHN DOE", "123 ELM", "CHICAGO"
```

The string "CHICAGO" would begin in position 49 of the screen.



---

Before we go on to the next chapter, let's look at one more example of numbers and strings. Look at these two values:

123  
"123"

The first value, 123, is a *number*. It can be added to or subtracted from another number. Let's display the number 123 on the screen. Enter:

```
PRINT 123
 123
-
```

PRINT 123

As you can see, the *number* 123 is indented one position on the screen. Remember that this position is reserved for a minus sign, in case the number is negative.

The second value, "123", is a *string*. It can't be added to another number. Your System/23 doesn't interpret "123" as a number. Instead, it is treated like any other string, such as "JOHN" or "JANUARY 4".

Let's display the string "123" on the screen. Enter:

```
PRINT "123"
 123
-
```

PRINT "123"

Now, enter:

```
PRINT "123";123
123 123
-
```

PRINT "123";123

As you can see, the *string* 123 is not indented one position. Remember that no space is reserved for a minus sign when you display a string. But, the *number* 123 is indented.

Also notice that the quotation marks do not appear when you display a string.

# The PRINT statement

---

## Chapter summary

PRINT is a BASIC statement that allows you to display information on the screen. You can display: numbers; strings, which are series of characters; or numbers and strings. The following statements will display:

- A number or numbers

```
PRINT 250
```

```
PRINT 250,500
```

```
PRINT 1;5;-7
```

- A string or strings

```
PRINT "JOHN"
```

```
PRINT "JOHN","MARY"
```

```
PRINT "JOHN "; "MARY "; "JANE"
```

- Numbers and strings

```
PRINT "JOHN",25
```

```
PRINT "BOOKS ";7
```

When you display more than one number or string, you must separate them in the PRINT statement with a comma or a semicolon. Commas are used to display information in print zones.

Now that you're familiar with the PRINT statement, try the following exercises before starting the next chapter.

---

## Exercises

### Question 1

---

What will be displayed if you enter the following statements?

- a. PRINT 5
- b. PRINT +123
- c. PRINT -1234

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

### Question 2

---

What will be displayed if you enter the following statements?

- a. PRINT "BASIC PROGRAMMING"
- b. PRINT BASIC PROGRAMMING

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

### Question 3

---

What should you do to stop the flashing when an error code and action code appear on your screen?

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



# The PRINT statement

---

## Exercises (continued)

### Question 4

---

Write a statement that will display the following on the screen:

THIS IS QUESTION #4

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

### Question 5

---

Write a statement that will display the number -3 on the screen.

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

### Question 6

---

Write a statement that will display the following on the screen:

a. LEARNING System/23 BASIC

b. LEARNING BASIC

c. -1 10

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

---

## Answers

### Question 1

---

- a. 5
- b. 123
- c. -1234

### Question 2

---

- a. BASIC PROGRAMMING
- b. An error message. BASIC PROGRAMMING must be inside quotation marks.

### Question 3

---

Press the Error Reset key. Remember to determine the cause of the error before you press Error Reset. You can look in the *Messages* manual for the error code.

### Question 4

---

```
PRINT "THIS IS QUESTION #4"
```

### Question 5

---

```
PRINT -3
```

### Question 6

---

- a. PRINT "LEARNING SYSTEM/30 BASIC"
- b. PRINT "LEARNING", "BASIC"
- c. PRINT -1; 10



# Chapter 2. Writing a program

---

## Introduction

In this chapter, you will write your first program. You will use two different BASIC statements: PRINT and END. As you learned in Chapter 1, the PRINT statement causes information to be displayed on the screen. The END statement signals the "end" of a program.

You will also learn how to use the following system commands: RUN, CLEAR, AUTO, and LIST. A *system command* controls the operation of a part of System/23: your 5322 processor, your diskette drives, or your printer.

## Objectives

---

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

- Write a program by using the PRINT and END statements.
- Run a program by using the RUN command.
- Assign line numbers to the statements in a program.
- Automatically assign line numbers to the statements in a program by using the AUTO command.
- Remove the statements in the work area by using the CLEAR command.
- Display the program statements that are in the work area by using the LIST command.

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.

# Writing a program

## Entering a program

In the last chapter, you learned how to use the PRINT statement to display numbers or strings immediately on the screen.

In this chapter, we will show you how to use the PRINT statement as part of a program. Remember from the Preface that a *program* is a series of statements that tell your System/23 what you want it to do.

When a PRINT statement is included as part of a program, information is not immediately displayed. Instead, your System/23 "remembers" the statement and uses it with the other statements in your program.

Before you enter the following BASIC program, which will display the string THIS IS EASY on the screen, enter:

```
CLEAR
--
READY INPUT
```

CLEAR

Now enter the following program exactly as shown. When we tell you to "enter a program", we mean enter all of the statements, one line after the other. (Don't forget to press the Enter key after each line.)

```
10 PRINT "THIS IS EASY"
20 END
--
```

```
10 PRINT "THIS IS EASY"
20 END
```

Notice that the PRINT and END statements are preceded by the numbers 10 and 20. These numbers are called *line numbers*. A line number must precede any statement entered as part of a program. Line numbers can range from 1 to 99999. Also note that you must leave a blank space between the line number and the statement.

---


We will explain more about those line numbers and CLEAR on the following pages. But first, let's look at the rest of the program.

Line 20 contains an END statement. You do not have to include an END statement, but we recommend that you do include an END statement in all of your programs. The END statement can only be entered on the *last* line (highest line number) of a BASIC program.

Now that you have entered your program, how do you run it? That is, how do you tell your System/23 to *execute*, or carry out the instructions in, all of the statements?

Enter the RUN command:

RUN




```
RUN
THIS IS EASY
-
```

Notice that the string THIS IS EASY is displayed, but not the word PRINT. Also, the END statement is not displayed. END only signals the end of the program.

You can run the same program again, because it is still in the *work area*. The work area is the part of System/23 internal storage where calculations are performed and statements are executed. When you enter a program, it is temporarily saved in the work area.

Enter RUN again:

RUN



```
RUN
THIS IS EASY
-
```

The program caused the same string to be displayed.

# Writing a program

---

## Entering a program (continued)

To get rid of a program in the work area, you enter a CLEAR command. You should always enter CLEAR before you enter a new program, as you did earlier in the chapter. Enter it now, but remember that once you enter CLEAR, any program statements in the work area are lost:

```
CLEAR
```

```
CLEAR
```

---

### Your turn!

---

Enter a program that will display your name on the screen. (We will use JOHN DOE in our example--you use *your* name.)

Hint: Use a PRINT statement and an END statement. Also, use line numbers 10 and 20.

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

Here's how we did it:

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

Now run your program:

```
RUN
```

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

If your program didn't run correctly, enter CLEAR and then reenter the statements as shown above, but use *your* name in the PRINT statement. Reminder: Don't forget to use quotation marks around your name.

---

## Assigning line numbers

There are two different ways to assign line numbers to the statements in a program. The first method allows you to select and enter the line numbers of your choice. This is the method we used when we entered our first two programs.

Let's try entering a new program.

First, enter:

```
CLEAR
10 PRINT "THIS"
20 PRINT "IS"
30 PRINT "EASY"
40 END
-
```

CLEAR

Now, enter the following:

```
10 PRINT "THIS"
20 PRINT "IS"
30 PRINT "EASY"
40 END
```

What will happen when you run this program?

Go ahead and run the program. That is, enter:

```
RUN
THIS
IS
EASY
-
```

RUN

Three lines are displayed, in the order that you entered the statements in the program.

Notice that we have assigned line numbers in multiples of ten. You do not have to use these numbers. The following program would do the same thing:

```
1 PRINT "THIS"
2 PRINT "IS"
3 PRINT "EASY"
4 END
```



# Writing a program

## Assigning line numbers (continued)

But, when you skip some numbers between statements, you can easily insert new statements into a program later. (We'll do that in the next chapter.)

Your System/23 executes your program statements in *ascending order*. That is, the lowest line number (10) is executed first, then line 20, then line 30, and so forth.

Your System/23 will execute program statements in ascending order, regardless of the order in which you enter them. To prove this, enter the following:

```
CLEAR
10 PRINT "ONE"
50 PRINT "FIVE"
60 END
40 PRINT "FOUR"
20 PRINT "TWO"
30 PRINT "THREE"
```

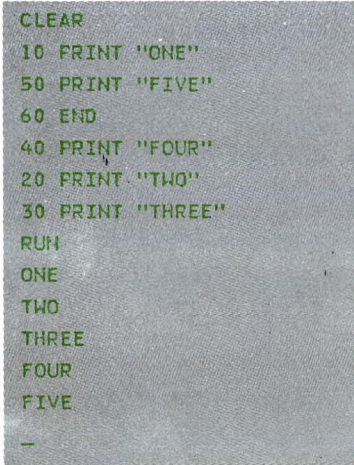
Now, enter:

```
RUN
```

Your System/23 runs the program in ascending line number order.

We do not recommend that you enter your program statements in the order shown in this example. It's a good practice to enter line numbers in the same order that you want them to be executed.

However, if you should accidentally skip a line while you are entering a program, you won't have to worry. You can simply enter the statement as you continue to enter the rest of your statements.



```
CLEAR
10 PRINT "ONE"
50 PRINT "FIVE"
60 END
40 PRINT "FOUR"
20 PRINT "TWO"
30 PRINT "THREE"
RUN
ONE
TWO
THREE
FOUR
FIVE
-
```

---

So far, you have entered program statements and assigned line numbers by typing the line number first and then the program statement.

10	PRINT	"THIS IS EASY"
↑	└──────────────────────────┘	
Line	Program	
number	statement	

Now we will show you how to tell your System/23 to automatically assign line numbers for you. You may remember from *Learning to Use System/23* that you can request automatic line numbers by using the AUTO command.

When you enter AUTO, your System/23 begins assigning line numbers with 00010. After you enter each statement, a new line number appears on the screen. The line numbers are in multiples of ten.

```
00010 PRINT "THIS"  
00020 PRINT "IS"  
00030 PRINT "EASY"  
.  
.  
.
```

Notice that the line numbers have leading zeros. For example, 00010, 00020, 00030 are used for lines 10, 20, and 30. Because all line numbers can have a maximum of five digits (99999), your System/23 provides the necessary number of leading zeros to make each line number the same length.

OK? Let's try it.

# Writing a program

## Assigning line numbers (continued)

To start, enter the CLEAR command:

```
CLEAR
```

Next, enter:

```
AUTO
```

You will see the number 00010 on the screen, followed by a blank space and the cursor.

On line 00010, enter:

```
00010 PRINT "SUNDAY"
```

Notice that as soon as you press the Enter key after typing PRINT "SUNDAY", your System/23 responds by displaying 00020 on the next line.

On line 00020, enter:

```
00020 PRINT "MONDAY"
```

Now, for the next five lines, enter PRINT statements to display the remaining five days of the week. On line 00080, enter an END statement.

```
00030 PRINT "TUESDAY"  
00040 PRINT "WEDNESDAY"  
00050 PRINT "THURSDAY"  
00060 PRINT "FRIDAY"  
00070 PRINT "SATURDAY"  
00080 END
```

```
CLEAR  
AUTO  
00010 _
```

```
CLEAR  
AUTO  
00010 PRINT "SUNDAY"  
00020 _
```

```
00010 PRINT "SUNDAY"  
00020 PRINT "MONDAY"  
00030 PRINT "TUESDAY"  
00040 PRINT "WEDNESDAY"  
00050 PRINT "THURSDAY"  
00060 PRINT "FRIDAY"  
00070 PRINT "SATURDAY"  
00080 END  
00090  
—
```

After you have entered the END statement, the screen will look like this:

```
00010 PRINT "SUNDAY"  
00020 PRINT "MONDAY"  
00030 PRINT "TUESDAY"  
00040 PRINT "WEDNESDAY"  
00050 PRINT "THURSDAY"  
00060 PRINT "FRIDAY"  
00070 PRINT "SATURDAY"  
00080 END  
00090  
—
```

What do you do now? You don't want to enter anything on line 00090. How do you stop the AUTO operation?

Press the Scroll up key ( ⤴ ) once. You now have a blank line on which to enter your next command.

So, to make sure the program works, enter:

**RUN**

```
RUN  
SUNDAY  
MONDAY  
TUESDAY  
WEDNESDAY  
THURSDAY  
FRIDAY  
SATURDAY  
—
```

Remember: To stop the AUTO operation, you press the scroll up key ( ⤴ ) once. This gives you a blank line on which to enter your next command.

Variations of the AUTO command allow you to automatically assign line numbers other than 10, 20, 30, etc. Refer to "AUTO command" in your *BASIC Language Reference* manual for more information.

# Writing a program

---

## Listing your program

As you gain more experience and start writing programs with more statements, it will often be necessary for you to examine the contents of the work area.

You can display the contents of the work area by using the LIST command.

When you enter LIST, all of the program statements in the work area are displayed. The information displayed is called a *program listing* or *listing*.

By looking at each line number and statement in the listing, you can verify that you have entered the program correctly, and that the program is still in the work area.

Let's try listing the contents of the work area.

Enter:

LIST

```
00010 PRINT "SUNDAY"  
00020 PRINT "MONDAY"  
00030 PRINT "TUESDAY"  
00040 PRINT "WEDNESDAY"  
00050 PRINT "THURSDAY"  
00060 PRINT "FRIDAY"  
00070 PRINT "SATURDAY"  
00080 END  
_
```

The program shown at the left should be displayed on the screen. This is the program you entered and assigned line numbers 00010-00080 by using the AUTO command.

You can enter LIST as often as you want without destroying or changing the contents of the work area.

Enter LIST again:

LIST

```
00010 PRINT "SUNDAY"  
00020 PRINT "MONDAY"  
00030 PRINT "TUESDAY"  
00040 PRINT "WEDNESDAY"  
00050 PRINT "THURSDAY"  
00060 PRINT "FRIDAY"  
00070 PRINT "SATURDAY"  
00080 END  
_
```

Notice that each time you enter LIST, the same program statements are displayed.

---

You can also list a section of a program. This is especially helpful when your program has a lot of statements. For example, enter the following:

```
00010 PRINT "SUNDAY"  
00020 PRINT "MONDAY"  
00030 PRINT "TUESDAY"  
—
```

```
00030 PRINT "TUESDAY"  
00040 PRINT "WEDNESDAY"  
00050 PRINT "THURSDAY"  
—
```

LIST 30

Only line 30 and any preceding lines are displayed.

Now, enter:

LIST 30,50

Only the lines from 30 through 50 are displayed.

What happens if your program has more than 22 statements? When you enter LIST, the first 22 statements are displayed. You must press the scroll up key (⤴) to display each additional statement.

One more thing about LIST: Notice that the line numbers are displayed with leading zeros, just as they were with AUTO. For example, 00010, 00020, and 00030 are displayed for lines 10, 20, and 30.

Even though a line number, such as 00010, has leading zeros when it is listed, you do not have to include the zeros when you enter the line number. When you assign line number 10 to a statement, you can enter 10 instead of 00010.

As with the AUTO command, LIST provides the leading zeros to display each line number as a 5-digit number.

For more information on LIST, see "LIST command" in your *BASIC Language Reference* manual.

# Writing a program

---

## Chapter summary

A program is a series of statements. Each statement is preceded by a line number. The END statement signals the end of a program to your System/23. If you include an END statement in a program, it must be the last statement in the program.

You enter your programs into your System/23 work area. The work area is where calculations are performed and statements are executed. Remember that program statements must be preceded by a line number.

To run a program in the work area, you enter the RUN command. When you run a program, the statements are executed in ascending line number order.

You can assign line numbers by entering the line numbers of your choice or by entering the AUTO command. Your System/23 will assign line numbers 00010, 00020, 00030, 00040, etc. when you enter AUTO.

You can display the contents of the work area by entering the LIST command. The information that is displayed is called a listing.

To remove all of your program statements from the work area, you enter the CLEAR command.

## Exercises

### Question 1

---

What commands do you enter to do the following?

- \_\_\_\_\_a. Tell your System/23 to automatically assign line numbers 10, 20, 30, etc. to the statements in a program.
- \_\_\_\_\_b. Remove all program statements from the work area.
- \_\_\_\_\_c. Display all of the program statements (or the first 22 statements) in the work area.
- \_\_\_\_\_d. Tell your System/23 to carry out the instructions in a program.

### Question 2

---

Write a program to display LEARNING SYSTEM/30 BASIC on the screen. Use line numbers 10 and 20.

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

### Question 3

---

Assign line numbers to the following statements so that the program will display the screen shown to the left.



```
THIS  
IS  
A  
BASIC  
PROGRAM
```

```
___ PRINT "THIS"  
___ PRINT "IS"  
___ END  
___ PRINT "BASIC"  
___ PRINT "A"  
___ PRINT "PROGRAM"
```



# Writing a program

---

## Answers

### Question 1

---

- a. AUTO
- b. CLEAR
- c. LIST
- d. RUN

### Question 2

---

```
10 PRINT "LEARNING SYSTEM/23 BASIC"  
20 END
```

The END statement is optional, so you could have a program with only one statement.

### Question 3

---

```
10 PRINT "THIS"  
20 PRINT "IS"  
60 END  
40 PRINT "BASIC"  
30 PRINT "A"  
50 PRINT "PROGRAM"
```

You don't have to use 10, 20, 30, etc. But, the line numbers you use should follow the order shown in this answer.

# Chapter 3. Changing a program

---

## Introduction

In Chapter 2, you learned how to write a program. Often, however, you will find it necessary to change your programs.

In this chapter, you will learn how to change line numbers in a program. You will also learn how to add statements to a program, how to remove statements from a program, and how to replace statements in a program.

## Objectives

---

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

- Change the line numbers in a BASIC program by using the RENUM command.
- Add a statement to a program.
- Remove a statement or a series of statements from a program by using the DEL command.
- Replace one statement with another statement in 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.

# Changing a program

## Changing line numbers

In the last chapter, you learned to assign line numbers to a new program by entering the numbers of your choice or by using the AUTO command. Now we will show you how to change the line numbers in an *existing* program.

You enter the RENUM command to change the line numbers in, or *renumber*, an existing program. Why would you ever want to change line numbers? Let's look at an example and see.

First, you should remove any statements that might still be in the work area from the last chapter. Do you remember how to do this? You enter the CLEAR command. So, enter:

```
CLEAR
15 PRINT "ONE"
16 PRINT "TWO"
17 PRINT "FOUR"
18 PRINT "FIVE"
19 END
-
```

CLEAR

Now enter the following:

```
15 PRINT "ONE"
16 PRINT "TWO"
17 PRINT "FOUR"
18 PRINT "FIVE"
19 END
```

Now list your program, so that you can look at what you entered:

```
00015 PRINT "ONE"
00016 PRINT "TWO"
00017 PRINT "FOUR"
00018 PRINT "FIVE"
00019 END
-
```

LIST

Notice that we forgot to enter a statement PRINT "THREE". As you can see, it would be impossible to add a statement between lines 16 and 17.

But, if we change the line numbers in, or *renumber*, this program, we will be able to insert a line with the statement PRINT "THREE".

```
RENUM  
-
```

```
00010 PRINT "ONE"  
00020 PRINT "TWO"  
00030 PRINT "FOUR"  
00040 PRINT "FIVE"  
00050 END  
-
```

```
RUN  
ONE  
TWO  
FOUR  
FIVE  
-
```

Enter:

```
RENUM
```

Now your line numbers have been changed. List the program, so that you can see what RENUM does:

```
LIST
```

You can see that the statements in this program now have line numbers 00010, 00020, etc. RENUM changes the line numbers in a program to multiples of 10, beginning with line number 00010.

Does that sound familiar? These numbers are just like the line numbers produced when you enter AUTO to assign line numbers to a new program.

As you can see, it is now possible to add a statement between lines 20 and 30. We'll do that next.

But first, a note about RENUM: We have shown you an example of RENUM with new line numbers 10, 20, 30, etc. If you want to use any other new line numbers, refer to "RENUM command" in your *BASIC Language Reference* manual.

Just to be sure that RENUM does not change the way your program runs, enter:

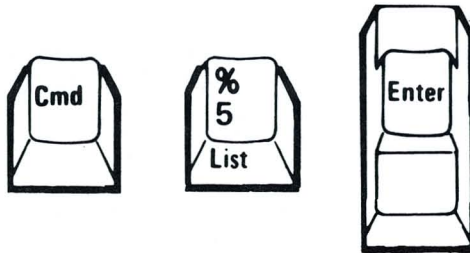
```
RUN
```

# Changing a program

## Changing line numbers (continued)

One more reminder: Many of the commands and BASIC statements that we tell you to enter can be entered by pressing the Cmd key and a special key. At the end of the command or statement, you must press the Enter key.

For example, you can enter LIST by pressing:



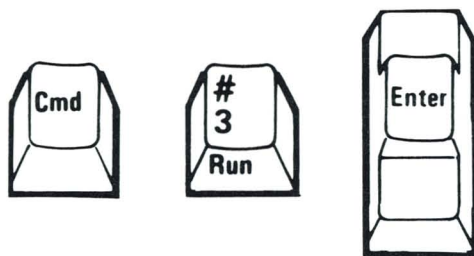
Try it. Enter LIST by holding the Cmd key down and pressing the key with List printed on the front. Let go of those keys and press the Enter key.

```
00010 PRINT "ONE"  
00020 PRINT "TWO"  
00030 PRINT "FOUR"  
00040 PRINT "FIVE"  
00050 END
```

LIST

---

Let's try another. You can run your program by holding the Cmd key down and pressing the key with Run printed on the front.



Go ahead and enter RUN by pressing these keys:

A terminal screen with a dark background and green text. The text reads: RUN, ONE, TWO, FOUR, FIVE, and a cursor line below.

RUN

For a complete list of special keys for commands and BASIC statements, refer to your *Keyboard Aids* and Chapter 3, "The Keyboard" in your *Operator Reference* manual.

# Changing a program

---

## Adding statements

Now we'll show you how to add a new statement to your program. As long as a program is still in the work area, you can make changes to that program.

To make sure your program is still in the work area, enter:

LIST

We will be adding a statement to your program between lines 20 and 30, so enter:

25 PRINT "THREE"

Now, list your program again to make sure that the statements are in the correct order:

LIST

Now run the program:

RUN

You can add one statement, or several statements, to a program by simply entering the statements at any time while your program is still in the work area. The line number of the new statement will determine its place in the program.

*Note:* We did not have to use line number 25 for our new statement. Any number from 21 through 29 would have been okay.

```
00010 PRINT "ONE"  
00020 PRINT "TWO"  
00030 PRINT "FOUR"  
00040 PRINT "FIVE"  
00050 END  
—
```

```
00010 PRINT "ONE"  
00020 PRINT "TWO"  
00025 PRINT "THREE"  
00030 PRINT "FOUR"  
00040 PRINT "FIVE"  
00050 END  
—
```

```
RUN  
ONE  
TWO  
THREE  
FOUR  
FIVE  
—
```

---

You may have noticed that when you enter LIST, your listing does not always look *exact/y* like what you entered. For example, if you enter two spaces between a line number and a program statement, like this:

```
80  PRINT 5
```

your listing will be corrected to show only one space, like this:

```
00080 PRINT 5
```

Also, program statements entered in lowercase, like this:

```
90 print 300
```

Will always be listed in uppercase, like this:

```
00090 PRINT 300
```

There is one exception. Any lowercase letters inside quotation marks, like this:

```
90 print "john doe"
```

will remain in lowercase, like this:

```
00090 PRINT "john doe"
```



# Changing a program

## Adding statements (continued)

### Your turn!

You're going to enter a program and then add a new statement to it.

Ready? First, enter the following:

```
CLEAR
10 PRINT "JANUARY", "FEBRUARY"
20 PRINT "MARCH", "APRIL"
30 PRINT "MAY", "JUNE"
40 PRINT "SEPTEMBER", "OCTOBER"
50 PRINT "NOVEMBER", "DECEMBER"
60 END
```

```
CLEAR
10 PRINT "JANUARY", "FEBRUARY"
20 PRINT "MARCH", "APRIL"
30 PRINT "MAY", "JUNE"
40 PRINT "SEPTEMBER", "OCTOBER"
50 PRINT "NOVEMBER", "DECEMBER"
60 END
-
```

Now, enter a statement so that the program will display JULY and AUGUST on the line under MAY and JUNE.

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

Your answer should look like this:

```
35 PRINT "JULY", "AUGUST"
```

Notice that our example uses line number 35. You could have used *any* line number from 31 through 39.

To make sure it works, run the program:

```
RUN
```

```
RUN
JANUARY          FEBRUARY
MARCH            APRIL
MAY              JUNE
JULY             AUGUST
SEPTEMBER        OCTOBER
NOVEMBER         DECEMBER
-
```

## Deleting statements

You have seen how to add statements to programs. You can also *delete*, or remove, statements from programs. You delete statements with the DEL command.

Let's see how this works.

To begin, enter the CLEAR command:

```
CLEAR
```

Now enter the following program exactly as shown:

```
10 PRINT "TEN"  
20 PRINT "TWENTY"  
30 PRINT "THIRTY"  
40 PRINT "FORTY"  
50 PRINT "FIFTY"  
60 END
```

Let's remove line 30 from the program. Enter the following:

```
DEL 30
```

Now list the program to see that line 30 has been deleted:

```
LIST
```

Run the program:

```
RUN
```

You can see that only four lines are displayed. Line 30 has been deleted.

```
CLEAR  
10 PRINT "TEN"  
20 PRINT "TWENTY"  
30 PRINT "THIRTY"  
40 PRINT "FORTY"  
50 PRINT "FIFTY"  
60 END  
-
```

```
00010 PRINT "TEN"  
00020 PRINT "TWENTY"  
00040 PRINT "FORTY"  
00050 PRINT "FIFTY"  
00060 END  
-
```

```
RUN  
TEN  
TWENTY  
FORTY  
FIFTY  
-
```

# Changing a program

## Deleting statements (continued)

You can delete more than one line at a time, too. When you delete more than one line at a time, the lines must be consecutive. For example, to delete lines 20 through 50, enter the following:

```
DEL 20,50
```

```
DEL 20,50
```

Now list the program, and you will see that only lines 10 and 60 are still in the work area:

```
LIST
```

```
00010 PRINT "TEN"  
00060 END
```

Notice that you must use a comma to separate the beginning and ending numbers of the lines to be deleted.

### Your turn!

What command would you enter if you wanted to delete lines 40 and 50 from the following program?

```
10 PRINT "THIS"  
20 PRINT "IS"  
30 PRINT "A"  
40 PRINT "7-LINE"  
50 PRINT "BASIC"  
60 PRINT "PROGRAM"  
70 END
```

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

You should have said:

```
DEL 40,50
```

## Replacing statements

You can also replace one statement with another statement in a program.

Before you try this, clear the work area:

`CLEAR`

Now enter the following:

```
10 PRINT "JOHN DOE"  
20 PRINT "123 ELM STREET"  
30 PRINT "CHICAGO, IL"  
40 END
```

This program will display a name and address. But, suppose you wanted to change the address without reentering the entire program. Here's how to do it:

Enter the following:

```
20 PRINT "456 MAIN STREET"
```

Now list the program to make sure line 20 was replaced:

`LIST`

And finally, run the program:

`RUN`

To replace one statement with another, you enter the new statement using the line number of the statement to be replaced.

```
CLEAR  
10 PRINT "JOHN DOE"  
20 PRINT "123 ELM STREET"  
30 PRINT "CHICAGO, IL"  
40 END  
-
```

```
20 PRINT "456 MAIN STREET"  
-
```

```
00010 PRINT "JOHN DOE"  
00020 PRINT "456 MAIN STREET"  
00030 PRINT "CHICAGO, IL"  
00040 END  
-
```

```
RUN  
JOHN DOE  
456 MAIN STREET  
CHICAGO, IL  
-
```

# Changing a program

## Replacing statements (continued)

Here's a helpful hint for you. When you change a line in a program, you do not need to reenter the entire line.

Instead, you can scroll down to the line you want to change, and move the cursor to the part that's changing.

Let's see how this works. List the program in the work area:

LIST

```
00010 PRINT "JOHN DOE"  
00020 PRINT "456 MAIN STREET"  
00030 PRINT "CHICAGO, IL"  
00040 END  
-
```

Now, let's change the city and state in line 30.

Press the Scroll down key (⏴) two times to get back to line 30.

```
00030 PRINT "CHICAGO, IL"  
-
```

Press the Cursor right key (→) until the cursor is under the C in CHICAGO. Now, enter:

```
00030 PRINT "NEW YORK, NY"  
-
```

NEW YORK, NY'

Now list the program:

```
00010 PRINT "JOHN DOE"  
00020 PRINT "456 MAIN STREET"  
00030 PRINT "NEW YORK, NY"  
00040 END  
-
```

LIST

As you can see, the line was replaced. Now, run the program:

```
RUN  
JOHN DOE  
456 MAIN STREET  
NEW YORK, NY  
-
```

RUN

---

## Your turn again!

---

Enter the following:

```
CLEAR
10 PRINT "LINE 30"
20 PRINT "LINE 20"
40 PRINT "LINE 40"
45 PRINT "LINE 45"
50 END
```

```
CLEAR
10 PRINT "LINE 30"
20 PRINT "LINE 20"
40 PRINT "LINE 40"
45 PRINT "LINE 45"
50 END
-
```

Now, enter statements and commands to do the following:

- Replace line 10 with a PRINT statement that will display LINE 10
- Add a statement to the program, using line number 30, that will display LINE 30
- Delete the statement that will display LINE 45

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

Here's our solution:

```
10 PRINT "LINE 10"
30 PRINT "LINE 30"
DEL 45
```

Now run your program:

```
RUN
```

```
10 PRINT "LINE 10"
30 PRINT "LINE 30"
DEL 45
RUN
LINE 10
LINE 20
LINE 30
LINE 40
-
```

# Changing a program

---

## Chapter summary

You can change the line numbers in a BASIC program by entering the RENUM command. RENUM changes the line numbers to 00010, 00020, 00030, etc.

You can add statements to a program by simply entering the statements while your program is in the work area.

You can delete statements from a program by using the DEL command. You enter DEL followed by a line number, or DEL followed by the first and last line numbers of consecutive statements.

You can replace one statement with another by entering the new statement using the same line number.

Several commands and BASIC statements can be entered by using the Cmd key and a special key. You should refer to your *Keyboard Aids* and "All about the keyboard" in your *Operator Reference* manual for a complete list of special keys.

---

## Exercises

### Question 1

---

What would you enter to change the line numbers in the following program to 10, 20, 30, and 40?

```
2 PRINT "HOW"  
4 PRINT "ARE"  
6 PRINT "YOU?"  
8 END
```

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

### Question 2

---

What would you enter to remove line 10 from the following program?

```
10 PRINT "LEARNING"  
20 PRINT "SYSTEM/23"  
30 PRINT "BASIC"  
40 END
```

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

### Question 3

---

What would you enter to add line 20 to the following program? (Line 20 contains the statement PRINT "BASIC".)

```
10 PRINT "SYSTEM/23"  
25 END
```

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



# Changing a program

---

## Exercises (continued)

### Question 4

---

What would you enter to replace line 30 with the statement PRINT "PROGRAM" in the following program?

```
10 PRINT "SYSTEM/23"  
20 PRINT "BASIC"  
30 PRINT "BOOK"  
40 END
```

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

### Question 5

---

What would you enter to delete lines 40, 50, and 60 from the following program?

```
10 PRINT "PENNY"  
20 PRINT "NICKEL"  
30 PRINT "DIME"  
40 PRINT "QUARTER"  
50 PRINT "HALF-DOLLAR"  
60 PRINT "DOLLAR"  
70 END
```

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

---

## Answers

### Question 1

---

RENUM

### Question 2

---

DEL 10

### Question 3

---

20 PRINT "BASIC"

### Question 4

---

30 PRINT "PROGRAM"

### Question 5

---

DEL 40,60

or

DEL 40

DEL 50

DEL 60

ANSWER

# Chapter 4. Performing arithmetic

---

## Introduction

In this chapter, you will learn how to perform arithmetic on your System/23. We will discuss the following operations: addition, subtraction, multiplication, and division.

We will also discuss *exponents*. An exponent is a number that indicates the power to which another number is raised.

### Objectives

---

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

- Perform arithmetic operations and display the results by using the PRINT statement.
- Determine the order in which operations will be performed on your System/23.

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.

# Performing arithmetic

## Adding and subtracting

You can add and subtract numbers on your System/23 almost like you do on a calculator. You enter the arithmetic operation in a PRINT statement, and your System/23 calculates and displays the results.

For example, enter the following:

```
PRINT 2+3
```

```
PRINT 2+3
5
-
```

Notice that the answer is displayed on the next line of the screen. You can also enter more than one operation in a PRINT statement. Let's try another one. Enter the following:

```
PRINT 5-1+4
```

```
PRINT 5-1+4
8
-
```

Again, the answer appears on the next line of the screen.

### Your turn!

What would you enter to add 12 plus 10?

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

You should have said:

```
PRINT 12+10
```

```
PRINT 12+10
22
-
```

You can also enter an arithmetic operation as part of a statement in a program. Enter:

```
CLEAR
10 PRINT 2+3-4
20 END
RUN
```

```
CLEAR
10 PRINT 2+3-4
20 END
RUN
1
-
```

## Multiplying and dividing

You can also multiply and divide numbers on your System/23. You may be used to seeing the symbols X and ÷, but on your System/23, you use \* and /.

Enter the following:

```
PRINT 12*10
```

```
PRINT 12*10
120
-
```

The \* is the multiplication symbol on your System/23.

Now, enter:

```
PRINT 10/5
```

```
PRINT 10/5
2
-
```

The / is the division symbol on your System/23.

If you use a negative number in an arithmetic operation, you must place the minus sign and the number in parentheses. If you do not enclose the minus sign and number in parentheses, your System/23 displays an error code and flashing action code.

For example, to multiply 12 by -2, enter:

```
PRINT 12*(-2)
```

```
PRINT 12*(-2)
-24
-
```

The only time you do not have to enclose the minus sign and number in parentheses is when the negative number is the first number in the operation. For example, you could enter:

```
PRINT -50/5
```

without getting an error. Go ahead and try it:

```
PRINT -50/5
```

```
PRINT -50/5
-10
-
```

# Performing arithmetic

---

## Raising a number to a power

Another arithmetic operation that you can perform on your System/23 is called *exponentiation*. Exponentiation means raising a number to a power.

For example, suppose you want to multiply the number 3 times itself. You may have heard this referred to as 3 squared ( $3^2$ ), or 3 raised to the power of 2.

On your System/23, you use the symbol `**` to indicate exponentiation. For example, you could say `3X3`, or

`3**2`

In this example, the 2 is the *exponent*. An exponent is the power to which a number is raised.

### Your turn!

---

How do you represent four raised to the third power on your System/23? (You may have heard this referred to as 4 cubed, or  $4^3$ .)

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

You should have said:

`4**3`

Go ahead and enter this.



```
PRINT 4**3
64
```

```
PRINT 4**3
```

---

Let's try an example to see how you might use exponents.

Suppose you want to determine the amount of money you would have after a period of years if you put \$100 in a savings account at 5% annual interest. The formula for calculating the amount is:

$$100*(1+.05)**N$$

↑                      ↙                      ↘  
Initial              Interest              Number  
principal          rate (5%)          of years

In this example, the number of years (N) is the exponent.

Let's try it first for a period of four years. Enter the following:

```
PRINT 100*(1+.05)**4
```

```
PRINT 100*(1+.05)**4  
121.550625  
-
```

You will find that you have \$121.55 at the end of four years. (Your System/23 gives you an answer of 121.550625, which you can round off to \$121.55.)

### Your turn!

---

Try entering a statement with the same formula that you used above for a period of three years.

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

You should have entered the following:

```
PRINT 100*(1+.05)**3
```

```
PRINT 100*(1+.05)**3  
115.7625  
-
```

(You can round the results off to \$115.76.)



# Performing arithmetic

---

## Arithmetic order

An *arithmetic expression* is a number or a series of numbers separated by operation symbols. The expression  $(1+.05)$  in the previous example is enclosed in parentheses, because your System/23 performs arithmetic operations in a particular order. Your System/23 does not necessarily perform operations in a left-to-right order.

Here's the order in which your System/23 performs the various arithmetic operations in an expression:

1. Parentheses ( ). When parentheses are *nested*, within another set of parentheses, the operation in the innermost pair is performed first. For example, in  $100*(1+(2*3)*4)$ , the  $(2*3)$  is performed first.
2. Exponents (\*\*).
3. Positive (+) and negative (-) signs before numbers.
4. Multiplication (\*) and division (/) equal priority.
5. Addition (+) and subtraction (-) equal priority.

If you have two or more operations of the same priority in one expression, the leftmost operation is performed first. For example, in the following expression:

$$2+3-4+5$$

the addition of  $2+3$  is performed first.

Using the accumulated savings formula, we'll demonstrate the order in which operations are performed in an expression.

```
100 * (1 + .05) ** 4
```

The addition is performed first, because it is enclosed in parentheses.

```
100 * 1.05 ** 4
```

The exponentiation is performed next.

```
100 * 1.21550625
```

The multiplication is performed last, and the answer of 121.550625 is obtained.

If the addition had not been enclosed in parentheses, it would have been the last operation performed--and that would have changed the answer.

Go ahead and enter the PRINT statement without parentheses:

```
PRINT 100*1+.05**4
```

```
PRINT 100*1+.05**4
100.00000625
-
```

As you can see, there is a big difference in the answer.

You can also display the results of more than one expression by using one PRINT statement. For example, enter the following:

```
PRINT 2+2,4*5
```

```
PRINT 2+2,4*5
4                20
-
```

The results are the same as if you had entered PRINT 4,20. Go ahead and try it. Enter:

```
PRINT 4,20
```

```
PRINT 4,20
4                20
-
```

# Performing arithmetic

---

## Chapter summary

An expression is a number or a series of numbers separated by operation symbols. The symbols you use on your System/23 are:

- Addition +
- Subtraction -
- Multiplication \*
- Division /
- Exponentiation \*\*

You can perform arithmetic operations on your System/23 and display the results by using an expression in a PRINT statement.

The operations in an expression are performed in the following order:

1. Parentheses
2. Exponents
3. Positive and negative numbers
4. Multiplication and division
5. Addition and subtraction

## Exercises

---

### Question 1

---

What would be displayed if you entered the following?

- a. PRINT 4\*3-5
- b. PRINT 10-15
- c. PRINT 1\*2;1\*\*2

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

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

c. \_\_\_\_\_

### Question 2

---

What would be displayed if you ran the following programs?

- a. 10 PRINT 2\*-2  
20 END
- b. 10 PRINT 2\*(-2)  
20 END

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

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

### Question 3

---

Write a statement that will display the sum of five plus six.

# Performing arithmetic

## Exercises (continued)

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

### Question 4

In what order would your System/23 perform the operations in the following expression?

$$10*2+(2+3)-40/2**2$$

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

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

### Question 5

What would be displayed if you entered the following?

PRINT 10\*2+(2+3)-40/2\*\*2

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

---

## Answers

### Question 1

---

- a. 7
- b. -5
- c. 2 - 1

### Question 2

---

- a. An error code and flashing action code. (You must include -2 in parentheses.)
- b. -4

### Question 3

---

PRINT 5+6

### Question 4

---

$10 * 2 + (2 + 3) - 40 / 2 ** 2$   
 $10 * 2 + 5 - 40 / 2 ** 2$   
 $10 * 2 + 5 - 40 / 4$   
 $20 + 5 - 40 / 4$   
 $20 + 5 - 10$   
 $25 - 10$   
 $+ 15$

### Question 5

---

15

216/10/11

Continued

# Chapter 5. Using variables and remarks

---

## Introduction

In this chapter, you're going to learn about *variables*. A variable is a symbol that represents a number or a string whose value may change during the running of a program.

You'll also learn how to include remarks in your programs.

## Objectives

---

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

- Use the LET statement to assign a value to a numeric or character variable.
- Write a statement that assigns a value to a variable *without* using the word LET.
- Add remarks to a program to make the program easier to use and understand.

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 variables and remarks

---

## Numeric variables

In the last chapter, you entered the formula for calculating the amount of money accumulated in a savings account. The initial principal was entered as \$100, and the interest rate was entered as .05.

$$100 * (1 + .05) ** N$$

The number of years was represented by the letter N. You entered 4 in the formula the first time for a four-year period. You then entered 3 in the formula for a three-year period. In these two instances, we could say  $N=4$  and  $N=3$ .

The letter N is called a *numeric variable* because the value of N is a *number* that you can *vary*, or change, each time you run a program with this formula.

A numeric variable name can be from one to eight characters long. The characters can be letters or numbers, but not spaces. The name must begin with a letter.

The following are valid numeric variable names:

**N**  
**A12**  
**HOURS**

The following are *not* valid numeric variable names:

**N N**  
**12A**  
**HOURSWORKED**

In the BASIC language, there is a statement that allows you to assign a value to a variable. The statement is called LET. For example,

---

```
LET A=2
```

assigns a value of 2 to a numeric variable named A. That is, the value of A is 2.

Just as a program is temporarily saved in the work area, the value of a variable is temporarily remembered.

The numeric variable A will have a certain value in a program until you change A to have another value.

Let's see how this works in a program. Enter the following:

```
CLEAR  
10 LET A=2  
20 PRINT A  
30 END
```

Now run the program:

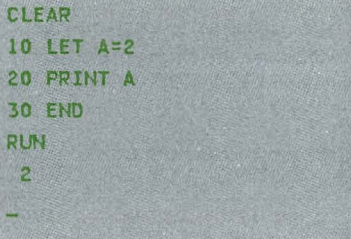
```
RUN
```

Notice line 20 of this program:

```
20 PRINT A
```

In Chapter 1, we told you that PRINT JOHN DOE causes an error if you don't enclose JOHN DOE in quotes. But in line 20, PRINT A does not cause an error. Why not?

In PRINT A, the A is a numeric variable, not a string. The A is only a symbol that represents a number. It does not require quotation marks. You *can't* have blank spaces in a variable name, so JOHN DOE cannot represent a numeric variable.



```
CLEAR  
10 LET A=2  
20 PRINT A  
30 END  
RUN  
2  
-
```

# Using variables and remarks

## Numeric variables (continued)

The following program uses LET to assign values to the variables and to calculate the results in the accumulated savings problem. Notice in lines 20 and 30 that the interest rate (I) is a variable, also. The amount of accumulated savings is then displayed.

Enter the following exactly as shown:

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

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

The first LET statement (line 10) assigns the value of 2 to N (the number of years). The second LET statement (line 20) assigns the value of .06 to I (the interest rate).

The third LET statement (line 30) calculates the amount of accumulated savings and assigns that value to A.

Notice that we have used LET in two different ways. You can use LET to assign either a number *or* the value of an expression to a numeric variable.

The PRINT statement (line 40) displays the value of A.

Now enter the RUN command:

```
RUN
```

```
RUN
112.36
```

What would happen if we added the following statement to our program?

```
15 LET N=4
```

Let's try it and see. Enter:

```
15 LET N=4
```

Remember from Chapter 3 that any time a program is in the work area, you can add more statements to the program by simply entering the statements.

Now list your program:

```
LIST
```

```
00010 LET N=2
00015 LET N=4
00020 LET I=.06
00030 LET A=100*(1+I)**N
00040 PRINT A
00050 END
-
```

As you can see, two different statements are assigning a value to N. Which value does your program assign to N? Actually, both values are assigned to N: 2 in line 10 and 4 in line 15. Your System/23 always uses the last value assigned, so the value of N is 4 when line 30 is executed.

Run your program:

```
RUN
```

```
RUN
126.247696
-
```

As you can see, the value of A, and therefore the result, is different. The formula in line 30 is using 4 years instead of 2.

Now enter the following:

```
CLEAR
10 PRINT X
20 END
RUN
```

```
CLEAR
10 PRINT X
20 END
RUN
0
-
```

In this example, you did not assign a value to the numeric variable X. Therefore, your System/23 assigns the value of 0 to X.

# Using variables and remarks

## Character variables

A *character variable* represents a string that may change during the running of a program. A character variable name must begin with a letter and end with a \$. It can include up to six additional letters and/or numbers, but no spaces.

The following are valid character variable names:

```
NAME$  
A10$  
ADDRESS$
```

The following are *not* valid character variable names:

```
$NAME  
10A$  
ADDRESSES$
```

Let's see how we can use a character variable in a program. (Remember from Chapter 1 that character strings must be enclosed in quotes.) Enter the following:

```
CLEAR  
10 LET DAY$="JUNE 6, 1938"  
20 PRINT DAY$  
30 END
```

Now run the program:

```
RUN
```

You can see that JUNE 6, 1938 is displayed. This is the value that we assigned to the character variable DAY\$.

*Note:* When you assign a string to a character variable, the string can not contain more than 18 characters.

```
CLEAR  
10 LET DAY$="JUNE 6, 1938"  
20 PRINT DAY$  
30 END  
-
```

```
RUN  
JUNE 6, 1938  
-
```

Let's insert a character variable into the accumulated savings program.

Enter the following. Again, remember that character strings must be enclosed in quotes.

```
CLEAR
10 LET N=4
20 LET I=.06
30 LET A=100*(1+I)**N
40 LET A$="THE TOTAL IS"
50 PRINT A$
60 PRINT A
70 END
```

Now run the program:

```
RUN
```

Notice that the value of the character variable (A\$) is displayed, as well as the value of the numeric variable (A).

Let's change this program so that we display the values of both A\$ and A on the same line. Enter the following:

```
50 PRINT A$;A
DEL 60
```

Now list the new version of your program:

```
LIST
```

Go ahead and run the program:

```
RUN
```

```
CLEAR
10 LET N=4
20 LET I=.06
30 LET A=100*(1+I)**N
40 LET A$="THE TOTAL IS"
50 PRINT A$
60 PRINT A
70 END
```

```
RUN
THE TOTAL IS
126.247696
```

```
00010 LET N=4
00020 LET I=.06
00030 LET A=100*(1+I)**N
00040 LET A$="THE TOTAL IS"
00050 PRINT A$;A
00070 END
```

```
RUN
THE TOTAL IS 126.247696
```

# Using variables and remarks

## Character variables (continued)

### Your turn!

The area of a circle can be calculated using the formula  $\pi R^2$ , where R equals the radius of the circle. On your System/23, you can enter  $\pi$  (which is approximately equal to 3.14159) as PI. PI is an *internal constant*. It's value never changes, and you cannot use PI as a variable name. Your formula looks like this:

```
AREA=PI*R**2
```

Now, suppose you want to know the area of a circle whose radius is 5 inches. Enter a program that will calculate the area and display the results. Use the numeric variable R for the radius and AREA for the area. (Reminder: Don't forget to clear the work area first.)

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

Here's our solution:

```
CLEAR  
10 LET R=5  
20 LET AREA=PI*R**2  
30 PRINT "THE AREA IS";AREA  
40 END
```

Now run your program:

```
RUN
```

If your program didn't run, reenter it as shown above.

```
CLEAR  
10 LET R=5  
20 LET AREA=PI*R**2  
30 PRINT "THE AREA IS";AREA  
40 END  
RUN  
THE AREA IS 78.5398163397448  
—
```

---

Now that you know how to use the LET statement to assign a value to a variable, we'll show you a shortcut. You don't have to use the word LET in your statement.

X=4

is the same as

LET X=4

To save time, you can omit the word LET from your program statements.

List the program currently in the work area:

LIST

```
00010 LET R=5
00020 LET AREA=PI*R**2
00030 PRINT "THE AREA IS";AREA
00040 END
```

Your program may be slightly different from the one we've shown, depending on how you assigned values and displayed the results.

Now, enter your LET statements again, but take out the word LET.

In our example, this is how to shorten the LET statements:

```
10 R=5
20 AREA=PI*R**2
```

Now run the program:

RUN

```
10 R=5
20 AREA=PI*R**2
RUN
THE AREA IS 78.5398163397463
```

As you can see, the results are the same with or without the word LET. However, the word LET will be inserted in your program listings. Let's see.



# Using variables and remarks

---

## Remark statements

List the program currently in the work area:

LIST

```
00010 LET R=5
00020 LET AREA=PI*R**2
00030 PRINT "THE AREA IS";AREA
00040 END
```

How can you make this program easier to understand? One way is to include descriptions of what the program is doing. These descriptions are called remark statements.

A remark statement provides information for you, the programmer, without performing any operation. Remarks help you, or anyone who might have to use your program, to understand what your program is doing.

You can use two different forms of remark statements in your programs.

The first form is a line number followed by REM followed by any remark you want to enter. These statements can be inserted anywhere in a BASIC program.

Examples of these remark statements are:

```
5 REM THIS PROGRAM CALCULATES AREA
25 REM AT THIS POINT, PRINT THE RESULTS
50 REM SAVINGS PROGRAM AS OF 01/04/81
```

Notice that you must leave a blank space between the word REM and the remark that follows it.

When you enter a remark statement with REM, no other BASIC statement can be entered on the same line.

The other way to enter a remark is to use an exclamation point (!). You can include an ! remark on a line by itself, or on the same line with another statement, like this:

10	<u>PRINT NAME\$</u>	<u>! DISPLAY NAME</u>
↑		
Line	BASIC	Remark
number	statement	statement

If the remark is included on the same line with another statement, the remark must *follow* the other statement.

Let's insert two remark statements in an example program. Enter the following, but use *your* name and *your* age:

```
CLEAR
10 REM DISPLAY NAME,AGE
20 NAME$="JOHN DOE"
30 AGE=42 ! BORN 1938
40 PRINT NAME$;AGE
50 END
```

```
CLEAR
10 REM DISPLAY NAME,AGE
20 NAME$="JOHN DOE"
30 AGE=42 ! BORN 1938
40 PRINT NAME$;AGE
50 END
RUN
JOHN DOE 42
-
```

Now run the program to make sure it works:

```
RUN
```

Finally, list the program:

```
LIST
```

```
LIST
00010 REM DISPLAY NAME,AGE
00020 LET NAME$="JOHN DOE"
00030 LET AGE=42 ! BORN 1938
00040 PRINT NAME$;AGE
00050 END
-
```

You can see that the remark statements do not affect the running of your program. They only make it easier for you to use and understand the program listing.

# Using variables and remarks

---

## Remark statements (continued)

As you begin to write programs with more statements, you may find it helpful to include special characters in your remark statements, like this:

```
10 REM *****DISPLAY NAME, AGE*****  
30 AGE=42 ! *****BORN 1938*****
```

The asterisks or other special characters of your choice, make it easier to find the remarks in a program listing.

*Note:* You cannot use the words REM, LET, or PRINT as variable names. These are reserved system keywords. For a complete list, see "Reserved words" in your *BASIC Language Reference* manual.

---

## Chapter summary

A variable is a symbol that represents a number or a string. The value of a variable may change during the running of a program. A numeric variable represents a number. A character variable represents a string.

Numeric variable names are from one to eight characters long, beginning with a letter. Character variable names are from two to nine characters long, beginning with a letter and ending with a dollar sign.

An internal constant is a value on your System/23 that never changes. On your System/23, the internal constant PI has an approximate value of 3.14159. You cannot use PI as a variable name.

You can assign values to variables by using LET statements. A LET statement can be entered in two different forms:

```
10 LET X=10           or       10 X=10
20 LET X$="XYZ"       or       20 X$="XYZ"
```

You can use a LET statement to assign a specific string or number *or* the value of an expression to a variable.

You can include remark statements in a program to make it easier to use and understand. Remark statements can be entered in two different forms:

```
10 PRINT A ! DISPLAY RESULTS
20 REM PROGRAM DISPLAYS NAME
```

# Using variables and remarks

---

## Exercises

### Question 1

---

Which of the following are valid numeric variable names?

- a. A1
- b. ZYX123
- c. 1A
- d. ADDRESSES
- e. A\$
- f. PRINT

### Question 2

---

Which of the following are valid character variable names?

- a. A1\$
- b. A1
- c. ZYX123\$
- d. 1A\$
- e. ABCDEFG

### Question 3

---

Which of the following are valid program statements?

- a. 10 LET X=10 ! ASSIGN VALUE
- b. 20 AREA=PI\*R\*\*2
- c. 30 A\$="THE AREA IS"
- d. 40 X=THE NUMBER OF ITEMS
- e. 50 REM X=THE NUMBER OF ITEMS
- f. 60 X\$=THE NUMBER OF ITEMS

---

#### Question 4

---

Write a program that will calculate and display the area of a square. The formula for the area of a square is side times side. Assign a value of 5 to the sides. Include REM statements in the program.

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

# Using variables and remarks

---

## Answers

### Question 1

---

- a. Valid
- b. Valid
- c. Invalid (a variable name must start with a letter)
- d. Invalid (a variable name cannot exceed eight characters)
- e. Invalid (a numeric variable name cannot end with \$)
- f. Invalid (PRINT is a reserved word)

### Question 2

---

- a. Valid
- b. Invalid (a character variable name must end with \$)
- c. Valid
- d. Invalid (a variable name must start with a letter)
- e. Invalid (a character variable name must end with \$)

### Question 3

---

- a. Valid
- b. Valid
- c. Valid
- d. Invalid (the single character X represents a numeric variable)
- e. Valid
- f. Invalid (a character string must be enclosed in quotes)

---

#### Question 4

---

```
10 REM THIS PROGRAM CALCULATES AREA
20 LET SIDE=5
30 LET AREA=SIDE*SIDE
40 REM DISPLAY THE RESULTS
50 PRINT AREA
60 END
```

Line 30 could be:

```
30 LET AREA=SIDE**2
```

The LETs are optional. Also, the REMs could be entered as exclamation points.





# Chapter 6. Using diskette storage

---

## Introduction

So far, everything you have done with your System/23 has been done in the work area. You have executed statements, performed calculations, and entered and run programs.

In this chapter, you will learn how to use *System/23 Diskette Storage*. Storage means a device into which data can be entered, in which it can be held, and from which it can be retrieved at a later time. *Data* refers to facts, concepts, or instructions.

You will learn how to store a copy of a program on a diskette, take a copy of a program that you have stored on a diskette and place it back into the work area, and replace one program on a diskette with another program.

## Objectives

---

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

- Save a copy of a program on a diskette by using the SAVE command.
- Place a copy of a program back into the work area from a diskette by using the LOAD command.
- Make changes to a program that has already been saved, and replace the old version on the diskette with the new version by using the REPLACE command.

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 diskette storage

## Saving a program

In the Preface, we told you that you would need a prepared diskette for this course. If you haven't already done so, insert your prepared diskette into diskette drive 1 at this time. Or, if you are using a processor without an internal diskette unit, insert your diskette into drive 3.

You learned how to prepare a diskette and how to insert a diskette into a diskette drive in *Learning to Use System/23*. For more information, see "Diskettes and diskette drives" in your *Operator Reference* manual.

Now we'll show you how the SAVE command works. We're going to enter a program that will add 4% sales tax to the price of an item. Enter the following:

```
CLEAR
10 PRICE=600 ! ADD TAX NOW
20 TOTAL=PRICE+(PRICE*.04)
30 PRINT PRICE,TOTAL
40 END
```

And run the program:

```
RUN
```

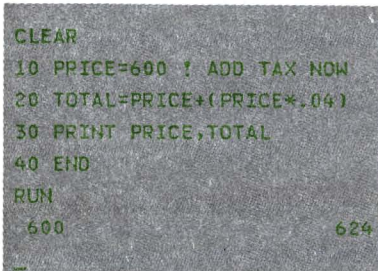
Now, you are going to save a copy of this program in a file called SALESTAX. If your diskette is in drive 1, enter:

```
SAVE SALESTAX//1
```

Or, if your diskette is in drive 3, enter:

```
SAVE SALESTAX//3
```

You have just saved your first program by entering a SAVE command.



```
CLEAR
10 PRICE=600 ! ADD TAX NOW
20 TOTAL=PRICE+(PRICE*.04)
30 PRINT PRICE,TOTAL
40 END
RUN
 600                624
```



```
SAVE SALESTAX//1
```



```
SAVE SALESTAX//3
```

---

SAVE tells your System/23 to store the contents of the work area into a specified file on a diskette. A *file* is a collection of related items that are stored together. A *program file* is a collection of statements that are preceded by line numbers.

SALESTAX is the name we chose for the file in which the program will be stored. A *file name* can be any name from 1 to 17 characters long. If a file name is longer than eight characters, it must be divided into two or more *simple file names*, separated by periods. A simple file name is from 1 to 8 characters long. The first character must be a letter. The rest of the characters can be letters or numbers, but not spaces.

Each file on a diskette must have its own unique name. The following are valid file names:

**SAVINGS**  
**PROGRAM1.SUE**  
**A1234**

The following are *not* valid file names:

**SAVINGSANDCHECKING**  
**1.PROGRAM**  
**TAX PROGRAM**

It is helpful to give a file a name that describes what is in the file. For example, your accumulated savings program could be stored in a file called SAVINGS. In Chapter 5, your program to find the area of a circle could be stored in a file called AREA.OF.CIRCLE.

# Using diskette storage

---

## Saving a program (continued)

When you save any program in a file, you must enter // following the file name. The // separates the file name from the *device address*. The device address tells your System/23 which diskette drive to store the file on. If you are using drive 1, your device address is 1. If you are using drive 3, your device address is 3.

In *Learning to Use System/23*, you learned how to use the DIR command. Let's try it now, to see that your file has been saved.

If you are using drive 1, enter:

```
DIR 1
```

Or, if you are using drive 3, enter:

```
DIR 3
```

```
DIR 1
VOLID 1 604160 0049 0000 512
09 SALESTAX      0008192 0004096 0001
-
```

You can see that SALESTAX has been placed on your diskette.

---

## Loading a program

You have saved a program by entering the SAVE command. You used the DIR command to make sure the program really was saved. Now what do you do?

Now that we know that the program is on your diskette, we can remove the program from the work area.

There are two ways to remove a program from the work area:

- Enter the CLEAR command. You learned in Chapter 2 that the CLEAR command clears the work area of all statements previously entered. This is the recommended way to clear the work area.
- Switch the power off and then switch it back on. This process may take from 25 to 35 seconds and is not recommended when you only want to clear the work area.

Naturally, when you switch the power off at the end of the day and then on again the next day, the work area will be cleared. Also, statements in the work area may be lost during a power interruption. But, if you have saved a copy of your program, nothing previously saved will be lost. Entries made since the last SAVE will be lost.

Now, clear the work area in the recommended way:

CLEAR



```
CLEAR  
-
```

There should be nothing left in the work area now. To be sure that the program is no longer in the work area, enter:

LIST



```
-  
READY ERROR
```

15

2005

# Using diskette storage

---

## Loading a program (continued)

An error code 2005 and flashing action code 15 are displayed. Your *Messages* manual says that this means your program cannot be listed because there is no program in the work area.

Press the Error Reset key.

Now, to use your program again, you must put a copy of the program back into the work area. We'll do that next by using the LOAD command.

LOAD is a command that tells your System/23 to find a specified file on a diskette and copy it into the work area. Enter the following:

**LOAD SALESTAX**



```
LOAD SALESTAX
```

SALESTAX is the name of the specified file in which the program was stored. The file name tells your System/23 which file on the diskette to load back into the work area. You must always enter the same file name when you return a program to the work area that you used when you saved the file on the diskette.

Just to be sure that the program is in the work area, enter:

**LIST**



```
00010 LET PRICE=600 ! ADD TAX NOW
00020 LET TOTAL=PRICE+PRICE*.04
00030 PRINT PRICE,TOTAL
00040 END
```

The program looks exactly like it did before you saved it, except that the keyword LET has been inserted in lines 10 and 20. At this point, you have the program in the work area. You also still have a copy of the program on your diskette in a file called SALESTAX. Any time you want to use this program again, you will enter:

**LOAD SALESTAX**

---

## Replacing a program

Now we will show you how to change your program and save the new version on your diskette. The new version will replace the version of the program that we stored in the file called SALESTAX.

Enter the following:

```
20 TOTAL=PRICE+(PRICE*.06)
```

```
20 TOTAL=PRICE+(PRICE*.06)
```

Now list the new version of your program:

```
LIST
```

```
00010 LET PRICE=600 ! ADD TAX NOW
00020 LET TOTAL=PRICE+PRICE*.06
00030 PRINT PRICE,TOTAL
00040 END
```

And finally, run the new program with the 6% tax rate:

```
RUN
```

```
RUN
600 636
```

You are going to save this new version on your diskette in the same file called SALESTAX. Enter the following:

```
REPLACE
```

REPLACE is a command that tells your System/23 to replace the contents of a file on a diskette with some new data.

Now the new version of your program is stored in the file called SALESTAX. The old version has been replaced.

If you had wanted to save this version of your program in a different file (called TAX06), you would have entered:

```
SAVE TAX06//1
```

or

```
SAVE TAX06//3
```



# Using diskette storage

---

## Replacing a program (continued)

Then, you would have two files stored on your diskette:

- SALESTAX (the original program)
- TAX06 (the revised program)

### Your turn!

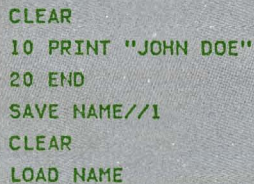
Enter a program to display your name. Save the program on your diskette in a new file called NAME, clear the work area, and load the program back into the work area.

Reminder: Don't forget to clear the work area first.

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

Our solution follows. (Your name and line numbers may be different. And, if you are using drive 3, you should enter a device address of 3.)

```
CLEAR  
10 PRINT "JOHN DOE"  
20 END  
SAVE NAME//1  
CLEAR  
LOAD NAME
```



```
CLEAR  
10 PRINT "JOHN DOE"  
20 END  
SAVE NAME//1  
CLEAR  
LOAD NAME  
-
```

If you had trouble with this example, reenter the commands and statements as shown above, but use *your* name.

---

## Chapter summary

To save a copy of a program on a diskette, you enter the SAVE command. When you enter SAVE, you must enter the name of the file that will store the program on the diskette and the number of the diskette drive into which the diskette is inserted.

A file name is used to distinguish one file from another on a diskette. Each file on a diskette must have its own unique name. A file name consists of 1 to 17 characters, beginning with a letter. If a file name is longer than 8 characters, it must be divided into two or more simple file names (1–8 characters), separated by periods.

To place a program back into the work area from a diskette, you enter the LOAD command. When you enter LOAD, you must enter the same file name that you used when you saved the file on the diskette.

To replace the contents of a file on a diskette with some new data, you enter the REPLACE command. It is a good idea to SAVE a part of a long program, and then enter REPLACE after adding several more lines. Then, if there is a power interruption, you will not have to reenter the entire program.

Remember: When using diskette storage,

**SAVE**—You must tell your System/23 what the file name is and which diskette to save it on.

**LOAD**—You must tell your System/23 which file to load, but you do not need to tell it which diskette. Your System/23 searches the diskettes for the file.

**REPLACE**—Your System/23 remembers the name of the file and which diskette to place it on.

# Using diskette storage

---

## Chapter summary (continued)

For more information on using storage, refer to "SAVE command", "LOAD command", and "REPLACE command" in your *BASIC Language Reference* manual.

---

## Exercises

### Question 1

---

Write a command that will copy a program from the work area into a new file called DATE10 on the diskette in diskette drive 1.

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

### Question 2

---

Write a command that will clear the work area. Then write a command that will load a file named PAYROLL from a diskette into the work area.

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

\_\_\_\_\_

### Question 3

---

You have just loaded a program called WORK1 into the work area and made some changes to it. Write a command that will store the updated program back onto the diskette in place of the original program.

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

### Question 4

---

Write a command that will store the updated version of WORK1 in a file called WORK2 on the diskette in drive 2.

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

# Using diskette storage

---

## Answers

### Question 1

---

SAVE DATE10//1

### Question 2

---

CLEAR  
LOAD PAYROLL

### Question 3

---

REPLACE

### Question 4

---

SAVE WORK2//2

# READER'S COMMENT FORM

SA34-0121-0

## I. Getting Started with BASIC

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









SA34-0121-0  
Printed in U.S.A.