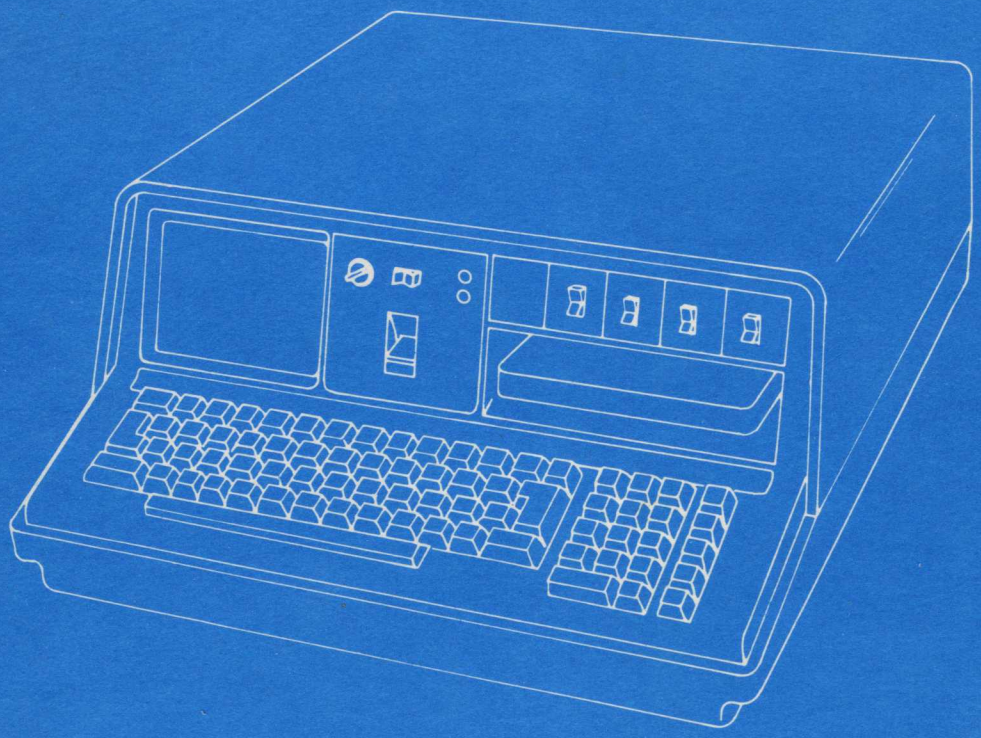
IBM

IBM 5110
BASIC Introduction

5110



IBM 5110 BASIC Introduction

SA21-9306-0

Preface

This manual introduces the IBM 5110 Computer and its BASIC programming capability. It is intended to provide the users of these products with the information necessary to operate the 5110 using the BASIC language.

Related Publications

- *IBM 5110 BASIC Reference Manual*, SA21-9308
- *IBM 5110 BASIC User's Guide*, SA21-9307
- *IBM 5110 BASIC Reference Handbook*, GX21-9309
- *IBM 5110 General Information and Physical Planning Manual*, GA21-9300
- *IBM 5110 Computing System Setup Procedure*, SA21-9318

First Edition (January 1978)

Changes are continually made to the specifications herein; any such changes will be reported in subsequent revisions or technical newsletters.

Requests for copies of IBM publications should be made to your IBM representative or the IBM branch office serving your locality.

A Reader's Comment Form is at the back of this publication. If the form has been removed, address your comments to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901. Comments become the property of IBM.

Chapter 1. Introduction

ABOUT THIS MANUAL

This manual will show you how to operate the 5110 using the BASIC language. If you are already familiar with the BASIC language, you may be able to skip most of the language-only topics and simply learn how to operate the 5110. If you are *not* familiar with the BASIC language, you should read the manual from cover to cover while performing the suggested keying operations or examples on your 5110. Not all of the features and functions of the BASIC language are discussed in this manual. For more information about the 5110 or the BASIC language, see the *IBM 5110 BASIC Reference Manual*, SA21-9308, or the *IBM 5110 BASIC User's Guide*, SA21-9307.

This manual assumes that your 5110 has been installed and checked out. If this is not the case, use the *IBM 5110 Customer Setup Manual*, SA21-9318, to install your system.

ABOUT BASIC

BASIC is an interactive computer language; that is, whatever you enter into the system is processed immediately. BASIC has many built-in functions that allow you to effectively solve your problems. BASIC also allows you to write programs using BASIC language statements and facilities. These programs can be stored on the tape cartridge or diskette for later use.

BASIC is a good language to experiment with. Nothing you do from the keyboard can damage the system; and the more you experiment, the more you will learn about BASIC and the system.

ABOUT THE 5110

The 5110 Model 1 (Figure 1) is a computer designed to help solve your business problems. The display screen and indicator lights communicate information to you, and the keyboard and switches allow you to control the operations the system will perform. The 5110 Model 2 is identical to the 5110 Model 1 except that it does not contain the built-in tape unit.

Before you begin to use the 5110, you should become familiar with the keys and the control panel (Figure 1). The control panel consists of a series of switches, which will be explained later.

What follows is a brief description of the keys. How you use the keys will be described later. For now, familiarize yourself with the names and locations of keys on the system. Figure 2 shows a BASIC-only keyboard, and Figure 3 shows a combined BASIC/APL keyboard.

If your system is equipped to operate either BASIC or APL programs, you may be unfamiliar with the symbols appearing at the top and on the front of the alphameric keys (Figure 3). For BASIC operations, *even on a BASIC-only machine (Figure 2) where they are not shown on the key top,* these symbols can be displayed or printed, although their APL functions do not apply to BASIC operations. See the *IBM 5110 BASIC Reference Handbook,* GX21-9309, for the APL characters and their locations.

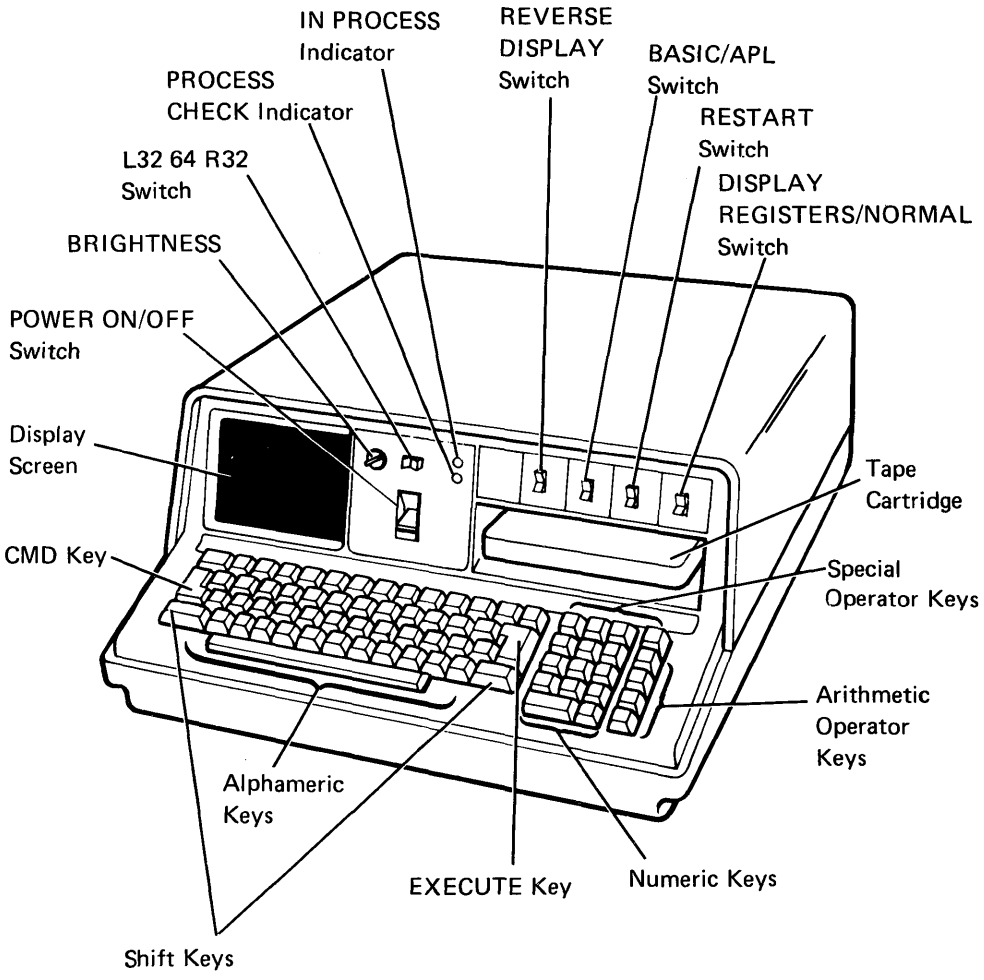


Figure 1. IBM 5110 Model 1 Computer

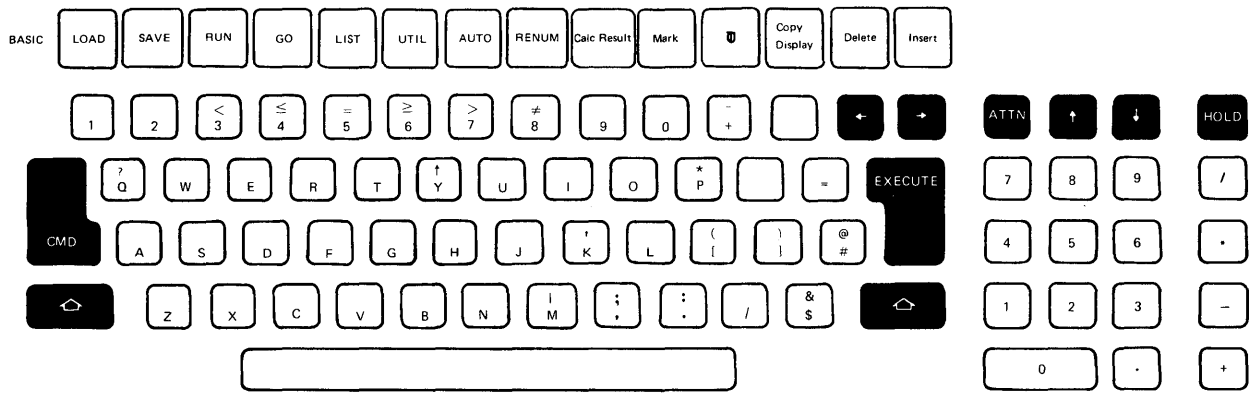


Figure 2. BASIC-Only Keyboard

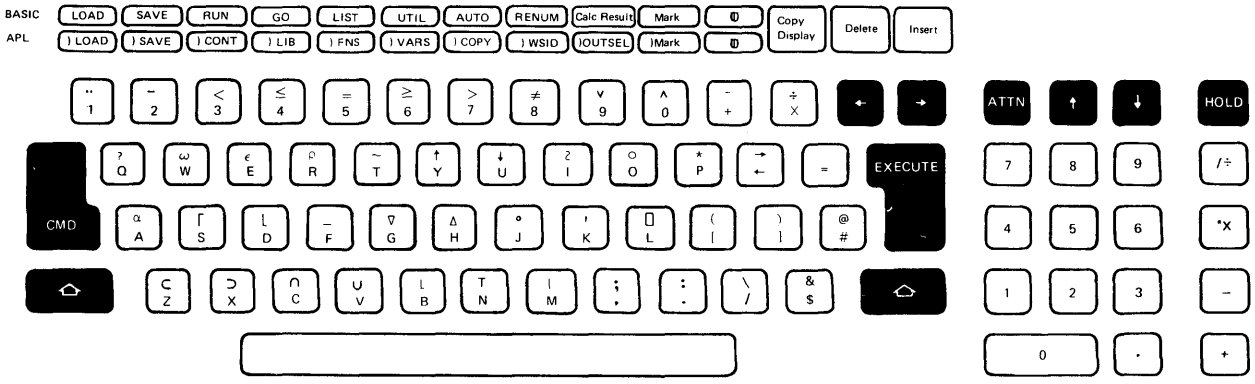


Figure 3. Combined BASIC/APL Keyboard

Alphameric Keys

The alpha keys are similar to those on a standard typewriter, except that there are no lowercase characters. The alpha characters are all uppercase, even though they are in the lowercase position on the keys. Thus, you *do not* use the shift key  for alpha characters.

If you want to enter an upper-shift character, you must hold down the shift key and then press the key to enter the character, just as you would to type an uppercase character on an ordinary typewriter.

You can also enter lowercase alphabetic characters from the keyboard. How you enter lowercase alphabetic characters is discussed later in this chapter.

Numeric Keys

Either the top row of alphameric keys or the special calculator arrangement of numeric keys on the right of the keyboard can be used to enter numbers.

Operating Keys

The dark gray keys with the legend names CMD, EXECUTE, ATTN, and HOLD, and the dark gray keys with the arrows are special operating keys (Figure 4). The dark gray keys with the arrows and the spacebar (used to enter blank characters) automatically repeat the operation they perform when held down.

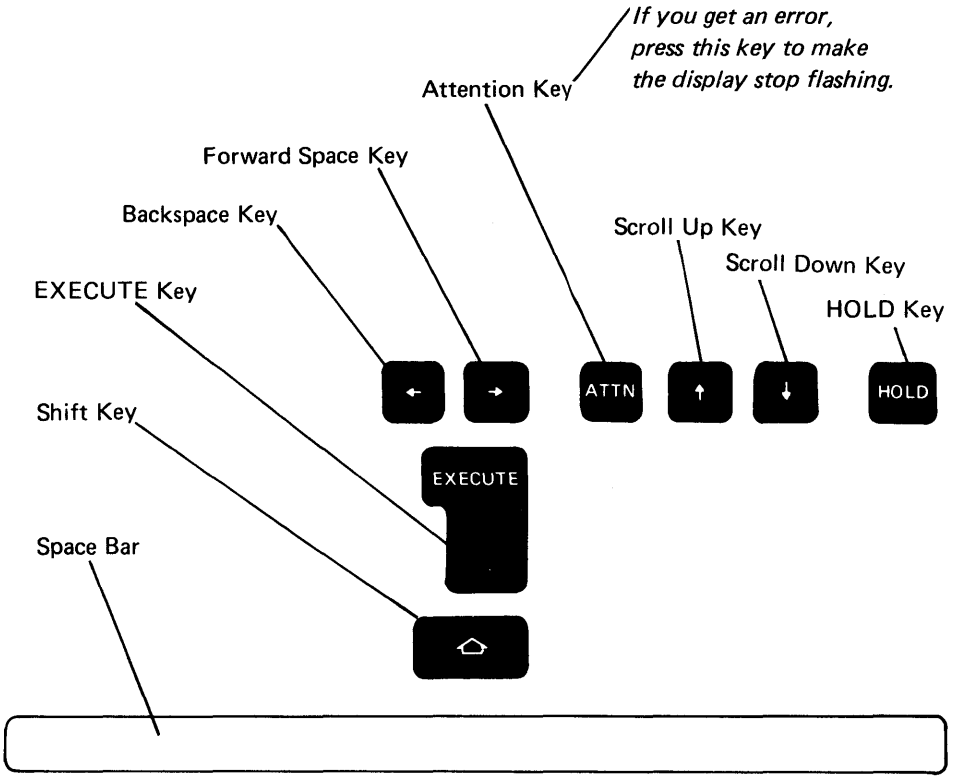


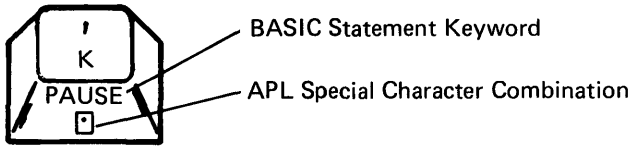
Figure 4. Special Operating Keys

BASIC Command Keywords

The words listed above the top row of alphameric keys (1-0) are BASIC command keywords that you can enter by holding down the CMD key and then pressing the key below the desired command. For example, to enter the LOAD command keyword, hold down the CMD key and press 1. These commands and their use are described later.

BASIC Statement Keywords

Notice the special character combinations of BASIC keywords engraved on the front of the alphabetic keys. If you have a combined APL/BASIC machine, there is also an APL special character combination on the front of the key. You can enter the BASIC keywords by holding down the CMD key and then pressing the appropriate key. You will see how these BASIC keywords are used as you become familiar with the BASIC language.



Note: You can use the APL special character combination only when you are using your system as an APL machine.

Arithmetic Operator Keys

The four keys to the right of the calculator arrangement of numeric keys are the arithmetic operator keys that are used to perform division, multiplication, subtraction, and addition. These operator keys also appear on the alphameric keyboard. In BASIC the symbol / is used for division, and the symbol * is used for multiplication.

GETTING STARTED

Make sure the switches on your control panel are set as follows:

Switch	Setting
L32 64 R32	64
DISPLAY REGISTERS/NORMAL	NORMAL
BASIC/APL (combined machines only)	BASIC

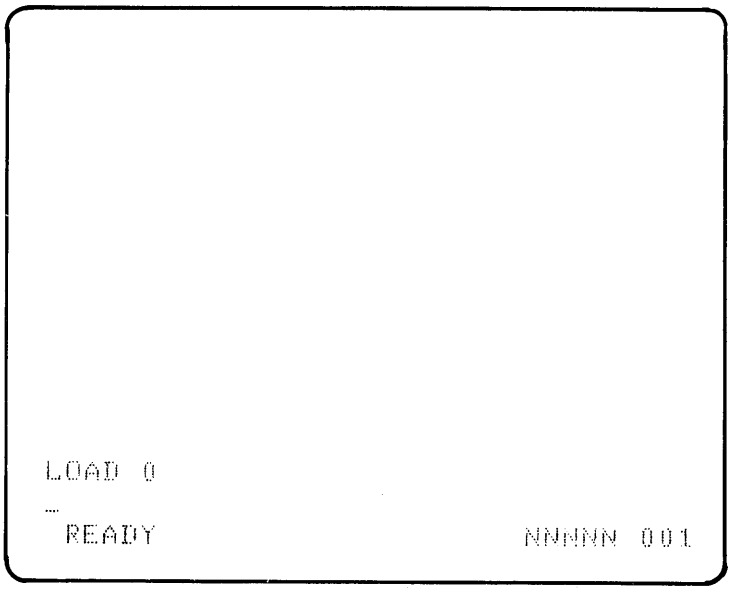
If your system has the BASIC/APL switch, it can execute either BASIC or APL language statements. The language used is selected only during the power up procedure or when the RESTART switch is pressed. Make sure your system is plugged in and turn the power on. If the power is already on, press RESTART and wait a few seconds. During this time, the system performs internal checks to make sure it is operating correctly. Do *not* press any keys while these internal checks are being made. If you inadvertently press a key, you must press RESTART to continue.

If an error is detected during these checks, the PROCESS CHECK indicator may come on. If the PROCESS CHECK indicator comes on, press RESTART. The system will again perform the internal checks. If the light comes on again, call for maintenance service.

The IN PROCESS indicator comes on whenever the display screen is blank, which indicates that the system is doing internal processing.

ENTERING AND DISPLAYING DATA

Next, let's look at the display screen. Your display screen should look like this:



If the READY message does not appear, press RESTART again, and wait a few seconds. If the READY message still does not appear, call your maintenance personnel.

The LOAD 0 (zero) message indicates that the system has a clear work area. The flashing underline (_) between the LOAD 0 and READY messages is called a cursor. It tells you where the next character you enter will be displayed. The READY message indicates that the system is ready to receive your instructions. The number in the lower right corner, indicated by the NNNNN on the display screen drawing, is the number of character positions (bytes) in the work area available for your instructions and data. This number changes during processing. The number is omitted on the remaining display screen drawings in the manual. The number following NNNNN (001) indicates the cursor position on the screen.

The display screen can contain up to 16 lines of data. The bottom line indicates the status of the system and specifies the number of bytes available in the work area (NNNNN) and current cursor position. The line next to the bottom displays the input you are entering from the keyboard. The remaining lines display the preceding 14 lines that have been entered and processed. When the data on the input line is processed, that line is moved up one line, leaving the input line empty so that more data can be entered. Up to 64 characters of data can be entered per line.

Before you start entering data into the system, press the HOLD key and then hold down the ⌂ key and press the . key (located on the right side of the keyboard). This places your system in the same character set used for the examples in this manual. See the *IBM 5110 BASIC Reference Manual* for a description of the character sets available with your 5110. Also, for some 5110 systems, the same character might appear on several keys on the keyboard. When doing the keying operations in this manual, always use the key with the character engraved in white on the top of the key.

Now let's enter some data into the system. Enter the following problem using the numeric keys and arithmetic operator keys:

2 + 3

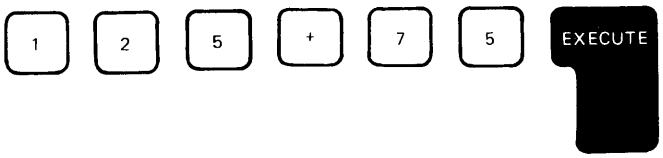
Notice that the characters are displayed as each key is pressed. To process the data you just entered, you must press the EXECUTE key. Press EXECUTE now.

The display screen shows:

```
LOAD 0
2+3
 5

_
 READY
```

Notice that the instruction you entered, 2+3, is on the left margin of the display screen, while the answer, 5, is indented one position from the left margin on the next line. The indentation allows a sign (- or +) to be displayed.

Enter and execute 125+75 by pressing the following keys:



This display screen shows:

```
LOAD 0
2+3
 5

125+75
 200

_
 READY
```

The appearance of your display can be changed by switches on the control panel. The REVERSE DISPLAY switch allows you to change from black characters on a white background to white characters on a black background, or vice versa. Change the switch settings and select the type of display you feel most comfortable with. You may have to adjust the BRIGHTNESS control switch as you change from one background to the other.

Now watch the display as you set the L32 64 R32 switch to the L32 position. With the switch in this position, the leftmost 32 characters on each line are displayed with a space between characters. With the switch in the L32 position, your display screen shows:

```
L O A D 0
2 + 3
  5

1 2 5 + 7 5
  2 0 0

_
  R E A D Y
```

Now set the switch in the R32 position and notice that the display is blank (except for the storage number and cursor position). In the R32 position, the rightmost 32 characters are displayed with a space between characters.

Return the switch to the 64 position, and notice that all characters are displayed without the space in between. For the exercises in the remainder of this book, keep the switch in the 64 position.

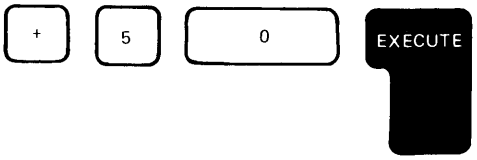
There are two dark gray keys with narrow white arrows above the numeric keyboard. These keys move the display lines (except the status line) up or down. The scroll up key  moves the display lines up one line, and the scroll down key  moves the display lines down one line. Both keys continue to move the display lines if they are held down. Now use the scroll down key  to move the display down two lines.

The display screen shows:

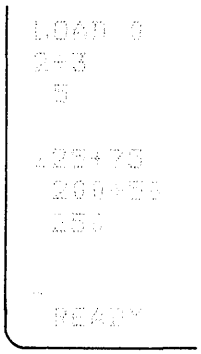
```
LOAD 0
2+3
 5

125+75
_200
 READY
```

The value 200 is now on the input line and can be used as input.

Use the forward space key → and move the cursor to the right of 200. Notice that the cursor (the underline) is replaced by a flashing character as you space the cursor through the numeric characters. The flashing character serves the same function as the cursor; it indicates the position in the line where input from the keyboard will be displayed. Now, press the following keys:



The display screen shows:



You are now familiar with the format of the display screen. From this point on, only the line or lines being discussed will be shown.

Entering Lowercase Alphabetic Characters

Although only the standard BASIC alphabetic characters are shown on the 5110 keyboard, you can enter lowercase alphabetic characters by changing the 5110 to lowercase character mode. One way to change the 5110 to lowercase character mode is to press the HOLD key (the characters HOLD are displayed in the lower left corner of the screen), then hold down the SHIFT key and press the scroll down  key.

The 5110 is now in lowercase character mode. For example, press the  key. The display screen looks like this:

```
k_
```

Now, hold down the shift  key and press the  key. The display screen looks like this:

```
kK_
```

Finally, hold down the command  key and press the  key. The display screen looks like this:

```
kK'_
```

In this example, you are *not* going to execute the data just entered from the keyboard. Instead, press the scroll up  key once to remove the data from the input line. Now, to return the 5110 to the standard BASIC character mode, press the HOLD key, and then hold down the shift  key and press the scroll up  key. The 5110 is now in standard BASIC character mode.

Note: All the entries in this manual are entered in standard BASIC character mode.

Correcting Keying Errors

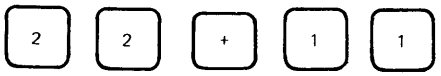
The system has a number of very useful functions that allow you to correct errors made while entering data. On a line-by-line basis, at any time, you can

- Replace a character
- Delete a character
- Insert a character

Replace a Character

To replace a character, move the cursor with the backspace key ← or forward space key → until it is at the incorrect character. The cursor moves one character space in the direction of the arrow each time the appropriate arrow key is pressed. These keys will continue to move the cursor if they are held down. The incorrect character is then replaced simply by keying the correct character over the incorrect character. (In some instances, characters can be combined to form a character not on the keyboard; for example, the period and quotation mark combine to make an exclamation mark. If you want to replace one of these characters (the . or ') with the other, you should backspace to the character, press the spacebar to blank the character, backspace again, then enter the desired character.)

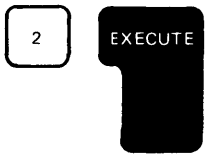
For example, you want to do the problem 22+12, but you press the following keys:



The display screen shows:

```
22+11_
```

To correct this error, the cursor must be moved back one position (under the second 1) so that character can be rekeyed. Now, press the backspace key ← once. To correct the keying error and execute the problem, press the following keys:



Delete a Character

To delete a character, you also use the backspace key ← or the forward space key → to position the cursor. Once the cursor is in the position of the character to be deleted (the character is flashing), hold down the CMD key and press the backspace key ← once.

The character is then deleted, and any characters to the right are shifted one position to the left to close up the space left by the deletion.

For example, you want to do the problem 13+45, but you press the following keys:

The display screen shows:

Press the backspace key to move the cursor (flashing character) back to the 2. Look at the labels that appear above the backspace and forward space keys: Delete and Insert. To perform the delete function, hold down the CMD key while you press ← once.

The display screen shows:

Now press EXECUTE to execute the problem. Pressing the EXECUTE key processes the entire line regardless of the position of the cursor.

You can also use the ATTN key to delete all the characters from the cursor position to the end of the line. For example, you press the following keys:

8 + 6 – 4 * 7 8

The display screen looks like this:

However, you only want to do the problem 8+6. Now, press the backspace key and move the cursor back to the -, then press ATTN

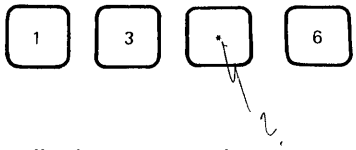
The display screen looks like this:

Press the EXECUTE key to execute the problem.

Insert a Character

To insert a character, position the cursor using the backspace key ← or the forward space key →, then hold down the CMD key and press the forward space key → once. This operation moves the flashing character (and all other characters to the right of it) one position to the right, creating the space you need to insert one character. The cursor is not moved and is now displayed as an underline. To insert the character, simply press the character key. If a character is in the last (64th) position of the line, the insert function is ignored.

For example, you want to do the problem 123*6, but you press the following keys:



The display screen shows:

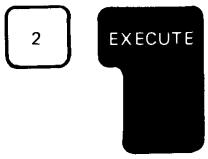
```
13*6_
```

To correct the error, press the backspace key to move the cursor (flashing character) back to the 3. Look at the labels that appear above the backspace and forward space keys: Delete and Insert. To perform the insert function with the cursor position at the 3, hold down the CMD key while you press → once.

The display screen shows:

```
1_3*6
```

To correct the keying error and execute the problem, press the following keys:



There is one more way to correct a keying error. If you make several errors halfway through the line, you can backspace the cursor to the character following the last correct character and then press the ATTN (attention) key. This causes everything from the cursor position to the end of the line to be cleared from the display screen.

Because the data from the input line is not processed until the EXECUTE key is pressed, you can visually verify any input before it is processed. However, if you do press EXECUTE before you notice a mistake, you can simply enter the input again, or you can use the scroll down key  to move the input back to the input line and correct it. When the corrections have been made, press EXECUTE again.

For example, you want to do the problem 135+280, but you entered and executed 134+280. The display screen shows:

```
134+280
 414
```

To correct the original input, press the scroll down key  three times to get the original input back to the input line. The display screen shows:

```
134+280
```

This character is flashing.

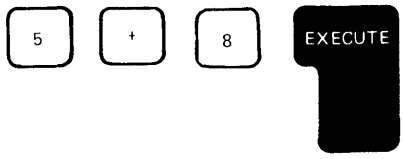
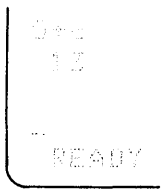
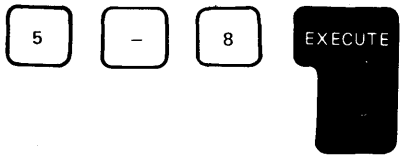
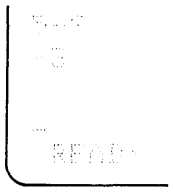
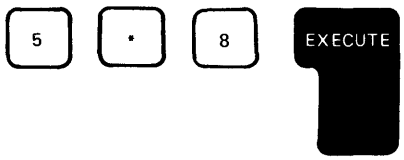
You may now correct the error, and then process the data again by pressing EXECUTE.

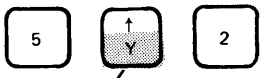
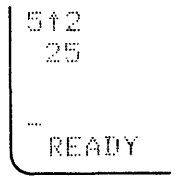
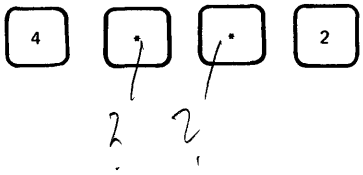
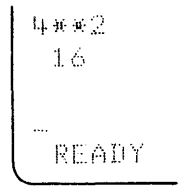
Chapter 2. How Your System Handles Arithmetic

The following examples show some common, simple arithmetic operations you can do on your system.

ARITHMETIC OPERATORS

Before you begin these simple operations, you should know that some of the arithmetic signs (called *operators*) you are accustomed to using are different when you use the BASIC language. For example, the multiply sign (x) is not used in BASIC. Instead, the asterisk (*) is used for multiplication. Similarly, the divide sign (÷) is replaced by the slash (/) in BASIC. The sign for exponentiation (raising to a power) in BASIC is ↑ (the upper shift character on the Y key) or **. Now enter these problems:

Arithmetic	You Press	The Display Screen Shows
Add 5 and 8		
Subtract 8 from 5		
Multiply 5 times 8		

Arithmetic	You Press	The Display Screen Shows
Divide 5 by 8		
5 to the power of 2	 (Hold down the shift key and press the Y key to get ↑ symbol.)	
4 to the power of 2		

Problems: Using Addition, Subtraction, Multiplication, and Division

1. Find the total number of cars that a dealer sold during one week if his daily sales were 3, 5, 2, 6, 7, 3 and 4.

2. Find the net number of cars removed from the same dealer's lot if 20 people had trade-ins.

3. Find the dealer's average profit per car if he made a total profit of $2700 for the sales in problem 1.

4. Find the dealer's total earnings if he made $20 on each car sold.

Possible Solutions

Problem 1:

```
3+5+2+6+7+3+4
 30
```

Problem 2:

```
30-20
 10
```

Problem 3:

```
2700/30
 90
```

Problem 4:

```
20*30
 600
```

VARIABLES

You can store data, either direct input that you enter from the keyboard or the result of a calculation. These stored items are called variables. Each variable has a name associated with it. Whenever you use the name of a variable, BASIC supplies the value associated with that name.

Variables That Stand for Numbers

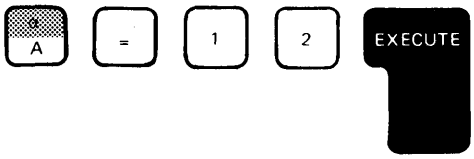
You can *name* a variable in BASIC with a single character of the extended BASIC alphabet (A-Z, @, $, and #). A BASIC variable can also be named with one of the preceding letters or symbols followed by a single digit (0 through 9). Typical variable names are A2, #9, and B1. You can name a maximum of 319 different numeric variables in BASIC.

Assigning Values to Variables

Variables are assigned values by means of the equal (=) sign. The value to the right of the = sign is assigned to the name to the left of the = sign. After you assign a value to a variable, you can use the variable in a calculation.

Examples

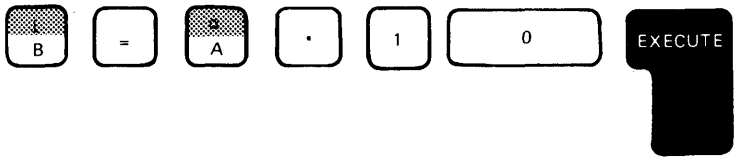
To illustrate the use of variables, enter the following:



After you press the EXECUTE key, you have created a variable named A with a value of 12.

```
A=12
 12
```

and



The result of a calculation can also be assigned to a variable.

```
B=A*10
 120
```

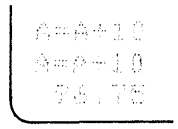
α A = 8 6 . 7 5 EXECUTE

You can change the value of a variable the same way you assigned the original value.

α A = α A + 1 0 EXECUTE

You can also use the variable and change its value in the same instruction.

This is a useful programming tool that will be explained further in a later chapter of this book.



1 B EXECUTE

If you want to know the current value of a variable, you simply enter the name of the variable.

∩ C /÷ α A EXECUTE

If you use the name of a variable to which you have not previously assigned a value, the system automatically assumes its value is zero (0).

Performing Several Functions in the Same Expression

In the preceding examples, only one arithmetic function was used in each example. However, you are not restricted to writing instructions with only one function. Any number of functions can occur in the same instruction. As soon as you use more than one function, however, you must be concerned about the order in which they are used.

The Sequence of Arithmetic Operations

BASIC has a prescribed order of arithmetic execution called arithmetic hierarchy. When two or more operators such as +, -, *, /, or ↑ are used, arithmetic is performed according to this hierarchy. That is, operators with higher priorities are performed first, while operators with the same priority are performed as they are encountered from left to right. The arithmetic hierarchy in BASIC is:

1. Operations enclosed in parentheses (highest priority)
2. Mathematical functions (for example, sine, cosine, or square root)
3. Exponentiation (↑ or **)
4. Positive/negative operations (described later)
5. Multiplication and division (*, /)
6. Addition and subtraction (+, -) (lowest priority)

When an operation is enclosed in parentheses, it is performed first, even though the operator enclosed in the parentheses may have a lower priority than the operators outside the parentheses. As a result, the prescribed order of execution can be changed if you use parentheses. *Operations enclosed in parentheses are executed in BASIC before operations outside parentheses, regardless of the hierarchy of the operators.*

Some examples of arithmetic hierarchy are:

```
3+4/5

4/5+3
```

In both of these examples, the 4 is first divided by the 5 because the divide operation has the highest priority. The .8 result is then added to the 3, giving a final result of 3.8.

Another example is:

(3+4)/5

In this example, the 3 and 4 are first added because they are enclosed in parentheses. The result, 7, is then divided by 5, giving a final result of 1.4.

In the expression

16+23-4+133-8

addition and subtraction occur from left to right in the following sequence:

```
  16
+ 23
  39     Interim result
-  4
  35     Interim result
+133
 168     Interim result
-  8
 160     Final result
```

However, when parentheses are added, the sequence of operations can change:

(16+23)-((4+133)-8)

In this example, the operations occur in the following sequence:

1. Proceeding from left to right, the expression in the first set of parentheses is evaluated.

```
  16
+ 23
  39     Interim result 1
```

2. The parenthetical expression within the second set of parentheses is evaluated next.

```
   4
+133
 137     Interim result 2
```

3. The second set of parentheses is now evaluated.

```
 137
-  8
 129     Interim result 3
```

4. Finally, the subtraction (having the same priority as addition) is performed and the result is displayed.

```
  39     Interim result 1
-129     Interim result 3
 - 90    Final result
```

Although the numbers and operators in this expression are the same as those in the previous example, the parentheses completely change the order of the operations and the final result.

Now, determine the result of the following expression by entering the numbers, then pressing the EXECUTE key:

```
4↑(3*(4-2))
```

The order of the arithmetic operations is: 4-2=2, 3x2=6, and 4^6=4096.

The display screen shows:

```
4↑(3*(4-2))
 4096
```

Figure 5 shows other examples of arithmetic expressions.

This is the Way it Looks as Arithmetic:	This is the Way it Looks in BASIC:	This is What it Means:
$\frac{a+b+c}{2}$	(A+B+C)/2	First add A, B, and C; divide the sum by 2.
$a+\frac{b+c}{2}$	A+(B+C)/2	Add B and C, divide the sum by 2; add the result to A.
3a+4	3* A+4	Multiply A by 3; then add 4.
3(a+4)	3*(A+4)	Add A and 4; multiply the sum by 3.
x^2+7	X↑2+7	Square X and add 7.
$(x+7)^2$	(X+7)↑2	Add X and 7; square the quantity.
$\frac{(x+1)^2}{2}$	(X+1)↑2/2	Add X and 1; square the quantity; divide the result by 2.
$\frac{\left(\frac{x^2}{2}\right)(x+y)}{3}$	(X↑2/2)*(X+Y)/3	Square X and divide by 2; add X and Y and multiply by the previous result; divide that result by 3.

Figure 5. Examples of Arithmetic and BASIC Expressions

As you can see, the more complicated the arithmetic expression looks, the more complicated the BASIC expression looks. When you are writing BASIC expressions, remember that parentheses must always be in balanced pairs–as many right parentheses as left parentheses. If a statement gets too complicated, you can usually break it down into several simpler statements.

Positive/Negative Operators

The plus (+) and minus (-) signs indicate that a number is positive or negative. When used for this purpose, the + and - signs have a higher priority in the arithmetic hierarchy than they have when used for addition and subtraction. In the following example:

```
-2↑2-3
```

the 2 is raised to the second power, and the minus is assigned to the result before the subtraction of the 3.

One rule that you must follow is that *you cannot use two operators sequentially*, except ** which is the same as ↑. Sequential operators must be separated by parentheses. This rule applies to both the arithmetic operators (+, -, *, /, and ↑) and the positive/negative operators (+ and -). For example, enter 7^{-3} as 7↑-3. The flashing display screen shows:

The lower arrow indicates the syntax error. Press ATTN to stop the flashing screen, then enter the corrected expression, 7↑(-3). A complete description of system error messages is included in the *IBM 5110 BASIC Reference Manual*, SA21-9308.

You must use parentheses to separate consecutive operators, as in the following examples:

Invalid	Valid
6+-4	6+(-4)
3*-1.5	3*(-1.5)
8--4	8-(-4)

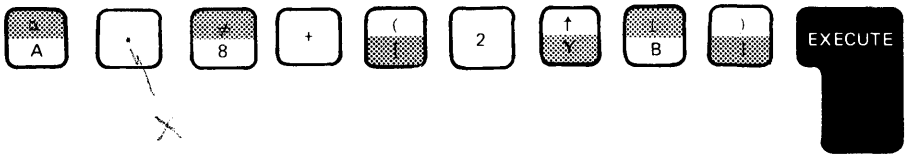
Try to solve the following problems using your system. Remember to press EXECUTE after entering each problem.

```
7*3.5-1.28
 23.22

3.2/(-16*.4)
-.5

12↑3*8/(4-.2)
 3637.894737
```

Answers

The order of arithmetic operators applies when variables are used also. For example, evaluate the following expression, where A=12 and B=4:

A times B plus (2 to the power of B)



The correct answer is 64.

A Note About Numbers

When you use numbers in BASIC, they can be:

- *Integers* (whole numbers) such as:

  2, -76, 842, 10000000, or 999111

- *Decimal numbers* such as:

  -1.5, 3.7772, 0.00081, or -457.25

- Numbers in *exponential format* such as:

  6E7 (meaning 6×10^7) or 5.4E-3 (meaning 5.4×10^{-3})

A Note About Names

When you name numeric variables in BASIC, they can be:

- A single character of the extended BASIC alphabet (A-Z, @, $, or #) such as:

  $, C, or V

- A single character of the extended BASIC alphabet (A-Z, @, $, and #) followed by a single number such as:

  A4, $6, $3, or T3

Variables That Stand for Characters

While you usually think of variables as standing for numbers, in BASIC you can let a variable stand for combinations of characters such as words or names. If a variable is going to represent a word or a name, it is called a *character variable*. You name character variables with one letter of the extended BASIC alphabet (A-Z, @, $, and #) followed by a dollar sign ($). The dollar sign tells the system that the variable is a character variable.

To assign a word or name to a character variable, you enclose the word or name in single quotation marks following the equal sign. When character data is displayed, the single quote marks do not appear. For example:

```
A$='HARVEY SMITH'
```

Here are the general rules:

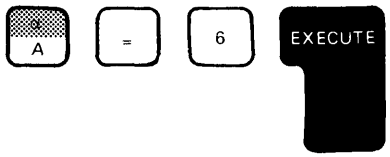
- A character variable is named by a single letter of the extended BASIC alphabet followed by a dollar sign ($).

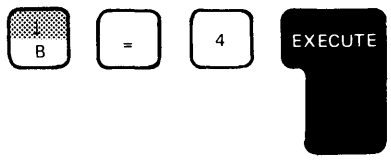
- To assign a value to the character variable, enclose the words or names you are assigning in single quotation marks following the equal sign.

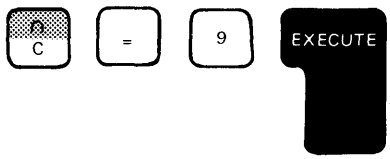
- The maximum length of a character variable is 255 characters (default length is 18 characters).

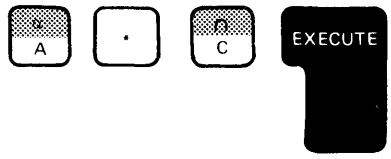
Using Calculation Results

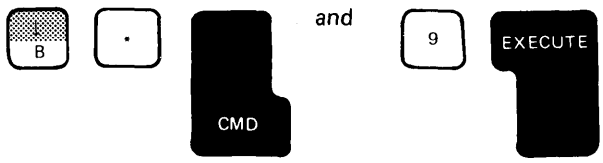
When you are entering a series of expressions in which the result from one expression is used in the next expression, you can use the 9 key (while holding down the CMD key) to insert the result of the last expression. This is the calc result function. Notice that Calc Result is listed above the 9 key. For example, key the following:











The display screen shows:

Notice that the system inserted the result of A*C into the second expression.

The calc result function will always insert the result of the *last* calculator expression (character expressions as well as arithmetic expressions). You must hold down the CMD key while you press  for the calc result function. Arithmetic results are enclosed in parentheses to avoid any conflict with adjacent operators in case the result is negative. Character results are enclosed in single quotation marks, just as they were when they were assigned.

Up to this point, you have been operating the system as a calculator. Any of the operations described thus far can be performed any time the system is waiting for you to enter input, with two exceptions:

- When a BASIC program is in operation and it stops for keyboard input required by an INPUT statement, you cannot perform any calculator operations.

- When you are entering data to create a keyboard-generated data file (explained later), you cannot perform any calculator operations.

Any time, other than the exceptions listed, you can enter any of the calculations described.

You can also stop a program during its operation to display or change the values of variables, then continue the program. This is extremely useful when you are checking a program for proper operation.

The following chapters discuss how to program your system using the BASIC language. Exercises for this and following chapters are provided in Chapter 13.

Chapter 3. Entering, Running, and Storing a Program

A program is your way to communicate with the system to solve a problem. The key words in this statement are *communicate* and *to solve a problem.* All programming is oriented toward problem solving. You can solve a problem by first analyzing the problem, then by formulating the solution. This involves communication. You can communicate with the system using the BASIC or APL language, as opposed to your communicating with other people in the English language. Thus, a program is little more than a means of translating your instructions to solve a problem into a language that the system understands.

ENTERING A PROGRAM

The following discussion shows you how to enter a BASIC program into the system and then how to execute that program. Also in this chapter, you will learn how to save a program on a magnetic tape cartridge or diskette, then load the program back into the system for execution again.

The program you will enter calculates accumulated savings. This calculation requires adding to the initial principal the interest earned at a specified rate for some period of time; in this case we use years. Enter the statements just as they are shown in the following example. Don't forget to press EXECUTE after entering each statement. You can enter the statements character-by-character or use the BASIC statement keyword keys with the CMD key. If the system detects an error in a statement you have entered, the keyboard becomes inactive (except for ATTN and HOLD), and the display will flash. To stop the flashing display, press ATTN, then correct the error.

```
0010 REM PROGRAM TO CALCLATE ACCUMULATED SAVINGS
0020 PRINT 'INITIAL PRINCIPAL'
0030 IPUT P
0040 IF P=0 GOTO 0100
0050 PRINT 'INTEREST RATE? , NUMBER OF YEARS?'
0060 INPUT R,N
0070 S=P*(1+R/100)↑N
0080 PRINT 'ACCUMULATED SAVINGS AFTER ';N;'YEARS IS $';S
0090 GOTO 0020
0100 STOP
```

Now read your entries on the display screen to see if you have entered the program correctly. If you find a keying error, the next paragraph describes how to correct the error before you run the program. If your system has an attached printer, you can hold down the CMD key and press the  key below Copy Display to get a copy of the displayed data. Note that this key is blank on a BASIC-only keyboard. The copy display function provides you with printed copy of all 16 lines of data.

Correcting Your Keying Errors

To correct or change a statement line of a program already entered in the system, use the gray scroll keys ( and ) to position the incorrect line to be changed on the input line right above the READY message line. When pressed momentarily, these keys move all information on the top 15 lines up or down one line position. When you hold these keys down, the display lines will repeatedly move up or down. When the line you want to change is positioned correctly, which is easy to identify because the first character will be flashing, you can use the forward space or backspace key to position the cursor at the character to be corrected. You can then use insert and delete functions to make the change. Remember, these functions are activated only when you hold down the CMD key and press  (Insert) or  (Delete). After all changes have been made to the line, press EXECUTE to reenter the line.

RUNNING THE PROGRAM

After you have entered the statement lines of the sample program, you are ready to run the program. To run the program, enter RUN, then press EXECUTE. Any error during execution causes the display to flash. Press ATTN to stop the flashing screen, then correct the error. You will have to enter RUN again to execute the program. When you run the program, the display screen shows:

```
RUN
INITIAL PRINCIPAL?

?
```

You will recognize the prompting message INITIAL PRINCIPAL? as part of the second statement in the sample program. This is a PRINT statement, which directs information to be displayed.

Note that the bottom question mark is flashing. The flashing question mark is a result of the INPUT statement in the sample program. The INPUT statement causes the question mark to be flashed to indicate that you are to enter information from the keyboard for the program.

Now respond to the request for data to be entered by keying a value for initial principal, then press EXECUTE. You can enter any number of digits you want. The maximum number of significant digits that the system will assign to any variable (your variable is P for principal) is 15 digits. You can include a decimal point, which does not count as a digit entry, but you must not enter commas. (Commas indicate multiple variables to the system.)

If you enter a decimal number with more than six digits to the right of the decimal point, any digits beyond the sixth are rounded when the answer for savings is displayed. The system is initialized to round numbers at the sixth decimal position. However, the rounding position can be changed to any position from 1 to 15 with the RD= command, which sets the rounding position. To set the system to round all displayed or printed results and calculations at the second decimal position, you would enter: RD=2.

The rounding command can also be included with the GO and RUN commands as described in the *IBM 5110 BASIC Reference Manual*, SA21-9308. Remember that whenever you turn the power on or press RESTART, the rounding position is set at 6.

All examples in this manual are run with the rounding position set to 6 digits (RD=6). If you change the rounding position, different results will be displayed.

You must remember that when using any programming language, including BASIC, you are communicating with the machine, telling it what you want it to do. Thus, you should define precisely what the machine does not know to avoid unnecessary problems.

You can enter values for the accumulated savings program as many times as you want. After you enter values for interest and years in response to the flashing question mark and press EXECUTE, the system will display the information you specified and compute the answer.

The statements in the sample program are described in the following paragraphs. In addition, Appendix A contains a short definition of all the BASIC statements used in the system.

Statement	Meaning
10 REM ACCUMULATED SAVINGS	The REM (remark) statement can appear anywhere in the program, but has no effect on program execution. This statement is used to insert comments into the BASIC program.
20 PRINT 'INITIAL PRINCIPAL?'	This PRINT statement specifies that INITIAL PRINCIPAL? be displayed. The single quotation marks around INITIAL PRINCIPAL? indicate that it is a character constant and that the entire character string is to be displayed.
30 INPUT P	The INPUT statement allows you to assign values from the keyboard to variables when your program is running. In this example, the variable P will receive the value you enter. The system displays a question mark in position 1 of the input line to indicate that keyboard input is expected.
40 IF P=0 GOTO 100	The IF statement transfers control to a specified statement when a specified condition is met. In this statement, the program will terminate when you enter 0 for the principal. As long as you want to calculate savings, you can enter values for the principal, interest, and years. When you are finished, however, just enter 0 for principal and the program goes to statement 100 (STOP).

Statement	Meaning
50 PRINT 'INTEREST RATE?, NUMBER OF YEARS?'	This statement displays the character string INTEREST RATE?, NUMBER OF YEARS?
60 INPUT R,N	This statement specifies that the variables R and N will receive the values you enter for interest rate and number of years, respectively. Again, a question mark will be displayed to indicate that keyboard input is expected. Values you enter must be separated by a comma.
70 S=P*(1+R/100)↑N	This statement indicates that the value of S (savings) is equal to the value of P (principal) multiplied by 1 plus the value R (rate divided by 100 to convert to percent), raised to the power of N (number of years).
80 PRINT 'ACCUMULATED SAVINGS AFTER',N;'YEARS IS' ;S	This statement indicates that the characters enclosed in single quotation marks are to be displayed, with the values for S and N inserted where shown. The value of N is the number of years you entered, and the value of S was calculated in statement 70.
90 GOTO 20	The GOTO statement transfers control to a specified line number. In this statement, control is transferred to the statement at line number 20. This provides for a number of savings calculations to be made repetitively.
100 STOP	The STOP statement indicates the end of execution of a BASIC program and terminates operations.

The numbers preceding the statements are called statement numbers. They are necessary so the system knows the proper sequence of your instructions. BASIC statement numbers in the 5110 can have values from 0001 through 9999. You can use consecutive numbers if you wish, but normally you should leave room for expansion between the statement numbers so that changes can be made more easily (see *Making Changes to Your Program*). When you enter statement numbers, you do not need to include the leading zeros. They will be added by the system when the statement is entered.

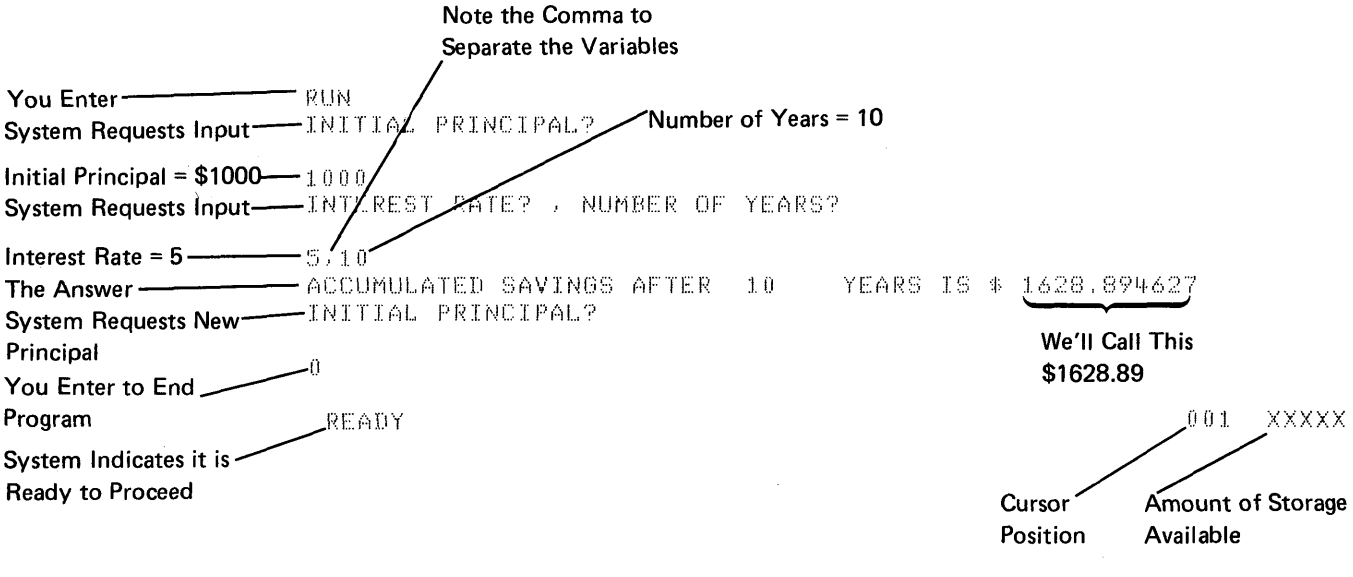
After you have computed your last savings calculation, you can end the program operation by entering 0 for the requested principal and pressing EXECUTE.

Automatic Statement Numbering

Instead of manually entering the statement numbers in a BASIC program, you can instruct the system to provide statement numbers for you. You can do this with the AUTO command. Simply enter AUTO and press EXECUTE. Notice that the word AUTO is displayed above statement number 0010. From this point on, the system numbers your statements in increments of 10. Automatic numbering continues until you enter anything other than an AUTO-numbered BASIC statement (a command word, for example, LIST) in the input line. You can restore automatic numbering by entering AUTO NNNN, where NNNN is the statement number you want to begin with.

Sample Run

Let's use the accumulated savings program to calculate the amount of savings generated by an investment of $1000 for 10 years at a rate of 5%.



USING TAPE OR DISKETTE STORAGE (YOUR LIBRARY)

So far, you have used only the 5110 active work area. The active work area is the part of 5110 internal storage where calculations are performed; it is also the place where your programs are stored. When you set the 5110 POWER ON/OFF switch to OFF, or press RESTART, the data in the active work area is lost. However, before turning the power off or pressing RESTART, you can save the data in your active work area by writing the contents of your active work area on a tape cartridge or diskette. This media (tape or diskette) is like a library; you can write the contents of your active work area on the media (like placing a book on a library shelf) and, at a later time, put the information stored on the media back into the active work area (like taking the book off the library shelf to use it again).

The library consists of one or more files (each file is like a book). Just as each book in a library has a name, each file that contains information on the media also can have a name (file identification).

The IBM 5110 system commands are your means of controlling the active work area and storage media. Look at the labels above the alphameric keyboard. These are the system command keywords, which you can enter by simply pressing the CMD key with the appropriate key below the label. The system command keywords can also be entered character by character. There are some system commands that do not appear on the labels above the keyboard. All of the system commands are described in detail in the *IBM 5110 BASIC Reference Manual*, SA21-9308.

In the following example, you will see how some of the system commands work. First, a tape cartridge or diskette must be inserted into your system.

If you are using a tape cartridge:

1. Be sure that the tape does not contain data required for further use.

2. Check the tape cartridge security arrow in the corner of the cartridge. Figure 6 shows the arrow pointing to SAFE. When the arrow is in this position, *the tape cannot be written on.* To be able to write on the tape, use a paper clip or a coin to turn the arrow away from SAFE.

3. Insert the tape cartridge. Figure 7 shows how a magnetic tape cartridge is inserted into the 5110. Press the cartridge in until it is firmly seated.

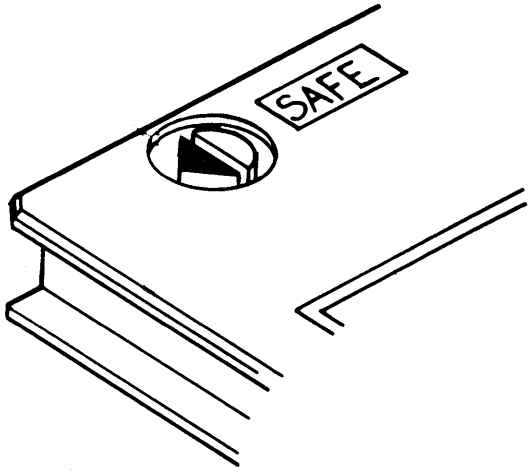


Figure 6. SAFE Arrow

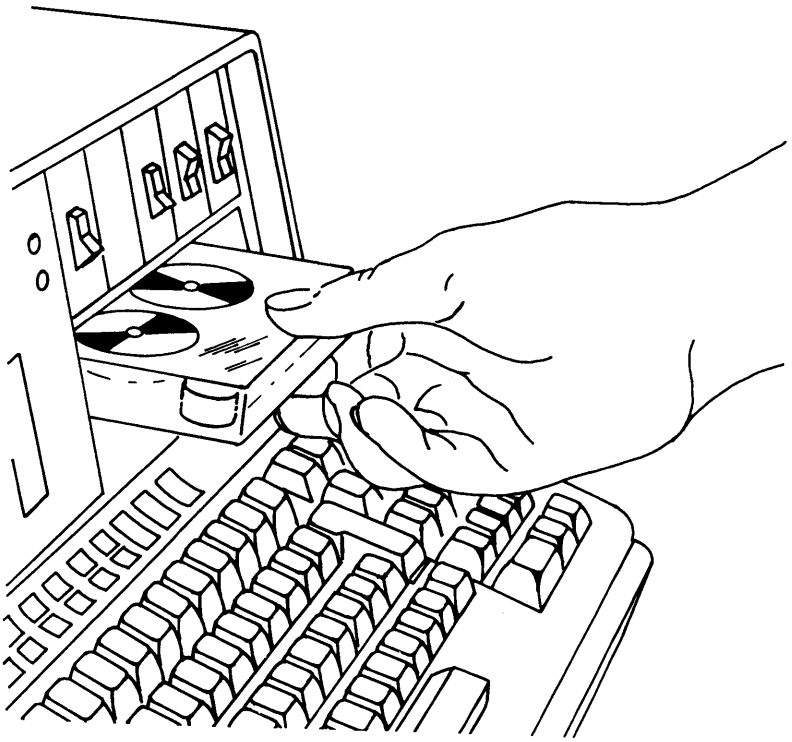


Figure 7. Inserting a Cartridge Into the 5110 Model 1

If you are using a diskette:

1. Be sure the diskette is initialized and contains no data required for other uses.

   Diskette initialization is the process of providing identification labels and other system information on diskettes. The initialization routine is among the customer support functions provided on diskette from IBM. You can select the initialization routine using the LINK command, as shown in the *IBM 5110 Customer Support Functions Reference Manual*, SA21-9311.

   Note: Diskettes supplied by IBM are initialized before they are sent to you.

2. Remove the diskette from the protective envelope (Figure 8).

3. Insert the diskette into diskette drive 1, as shown in Figure 9.

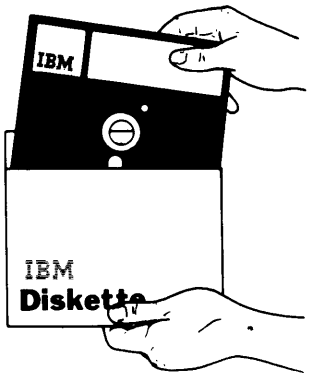


Figure 8. Removing the Diskette from the Protective Envelope

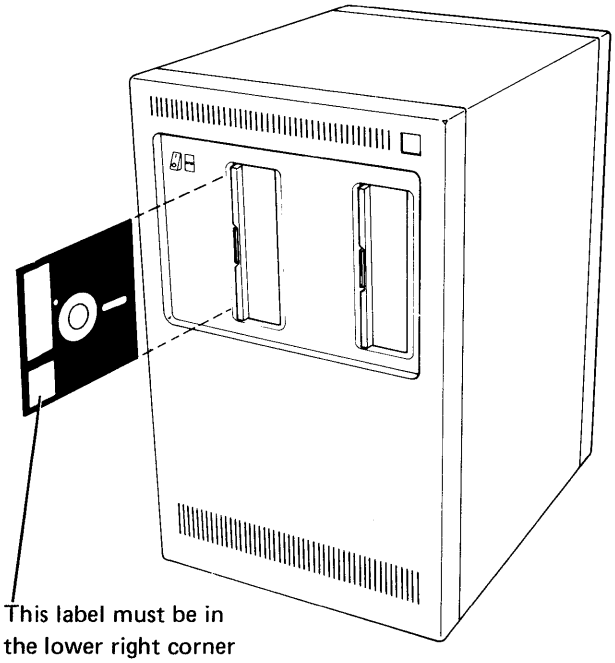


Figure 9. Inserting a Diskette in Diskette Drive 1

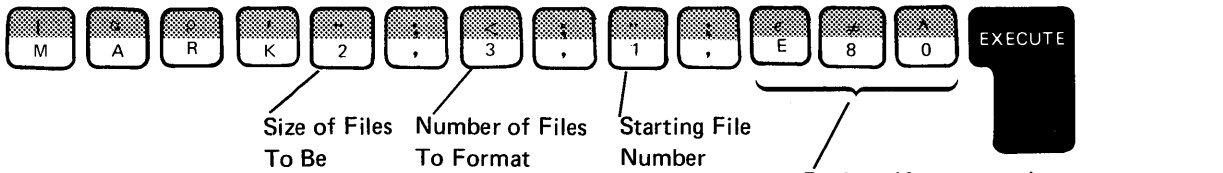
Marking Your Media

To prepare a tape or diskette with one file of any size, or several files of the same size in the same operation, you enter a MARK command. The MARK command can be entered anytime that READY is indicated on the display screen. It will not interfere with your programs. For our exercise, we are assuming that you are beginning a new media (no files have been marked). We will mark three files, each containing 2048 or 2K character positions (K is equal to 1024). This exercise provides enough space on your media to contain three programs (one in each file) of approximately 35 statements each. Tape files for the 5110 are numbered sequentially beginning with 1. Should you later decide to add more files, you can do so as long as you do not exceed the physical limits of your media.

A tape contains space for approximately 200K of storage, minus the leading and trailing data for each data file, which equals 0.5K per file. Thus, a tape can contain approximately 132 1K files, 44 4K files, or any combination of file sizes up to 200K, including the required 0.5K per file. For information on diskette capacities, see the *IBM 5110 BASIC User's Guide*, SA21-9307.

Note: If you are not using a new tape cartridge, you must first ensure that your tape does not contain important data belonging to someone else. This is necessary because any existing data beyond the file you MARK is erased when you MARK the tape again. The system displays a warning message when you attempt a MARK command to a file that is already marked. To continue marking the file, press ATTN to stop the flashing screen, press the scroll up key once to move the display up one line, enter GO in positions 1 and 2, then press EXECUTE. To discontinue the MARK operation, enter GO END, then press EXECUTE.

To mark the tape in our exercise, press the following keys:



Size of Files To Be Formatted

Number of Files To Format

Starting File Number

Device. If you are using tape storage, specify E80. If you are using diskette storage, specify D80 (diskette drive 1). See the *5110 BASIC Reference Manual* for details on device addresses.

You have now marked the media for three files of 2K characters each, starting with file 1. The READY message is displayed when the media preparation is completed. We will now save the savings program on your media. If you want to mark additional files, remember that you should begin with file number 4.

In the following examples, if you are using diskette storage, specify device address D80 (diskette drive 1) instead of device address E80 (built-in tape drive).

SAVE Command

The SAVINGS program can be saved on tape or diskette with the SAVE command.

To save the sample program on tape, enter SAVE, then enter the number of the file you want to save it in, followed by E80, which is the device address of the Model 1 built-in tape unit. You will save the program in file 1, so enter SAVE 1,'SAVINGS',E80. The READY message will be displayed to tell you when the program is saved on tape. (You needn't be concerned with the numbers following the READY message.) While the program is being saved, you will notice the tape in the cartridge moving back and forth. This is normal, because the system is reading each segment of data after it is written. This ensures that the information is saved correctly.

To save the program on diskette, you will need a file number, a file name, and a device address. Use file number 1 and file name SAVINGS. Enter SAVE 1,'SAVINGS',D80. To save the program on diskette, you must enter the device address of diskette drive 1, which is D80. To prove that the program has been saved, and that you can load it back into storage, the program stored in the system must first be erased.

There are three ways to do this:

- Enter LOAD0 and press EXECUTE. This clears machine storage and prepares it to accept input from the keyboard or programs loaded from tape. This is the recommended way to clear machine storage.

- Press the RESTART switch. This restarts the machine to the same status as when the power was turned on. The internal diagnostics are performed again; thus, this method requires 10-15 seconds depending on the amount of storage in your machine. This method is recommended only when the PROCESS CHECK indicator comes on, or when you change from BASIC to APL or APL to BASIC.

- Set the power switch to OFF, then set it back to ON. The same diagnostics are performed as during RESTART.

To clear the machine, enter LOAD0 and press EXECUTE. To prove the program no longer exists in the machine, enter RUN and press EXECUTE. The system will respond with an error message to let you know this cannot be done because there is no program in storage. Press ATTN to continue.

In order to run the program again, it must first be loaded into storage from where it was stored on the tape or diskette. The LOAD command is used to place the program back into storage.

LOAD Command

To load the SAVINGS program back into 5110 storage, enter LOAD, then enter the number and/or name of the file containing the program you want to load. In this example, type in 1 for file 1, then enter device address E80 for tape or D80 for diskette. Complete this sequence by pressing EXECUTE. The READY message tells you that the program is loaded and can be executed again. Run the program again by entering RUN, then pressing EXECUTE.

Practice the SAVE and LOAD commands by changing the file number and file name when you again save the sample program on tape or diskette and load it back into the system.

Listing a Directory of Programs

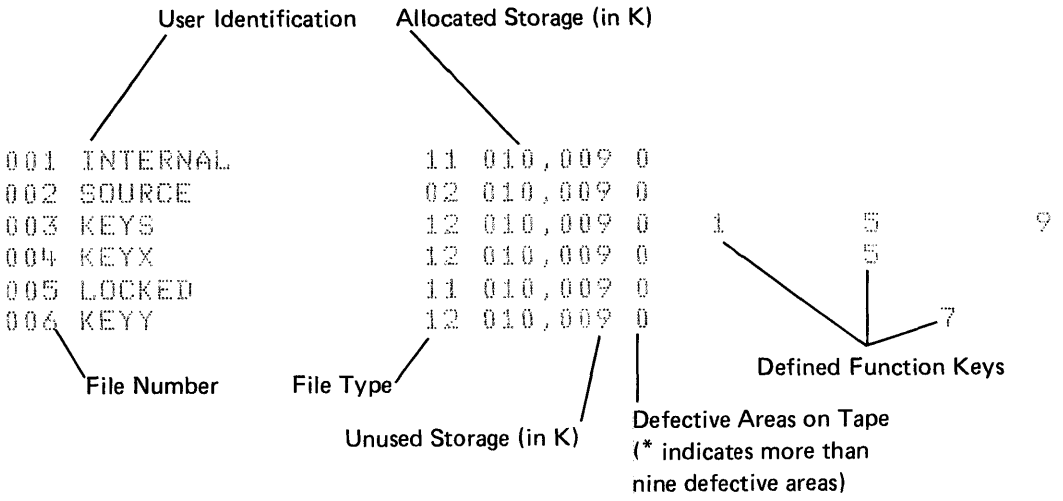
When you save a program, it goes into your collection of programs and data on tape or diskette. You may find it necessary at times to list a directory of the programs you have in the files on tape or diskette. To do this, you use the UTIL command. The UTIL command instructs the system to list the following information about each file on tape or diskette:

- File number
- File identification
- File type
- Amount of storage allocated to the file
- Amount of unused storage in the file
- Number of defective areas (on tape only)
- File protection status (on diskette only)
- Data set starting location (on diskette only)
- Function keys defined in the file

A UTIL command for a typical tape directory is shown below.

```
UTIL DIR1,E80
```

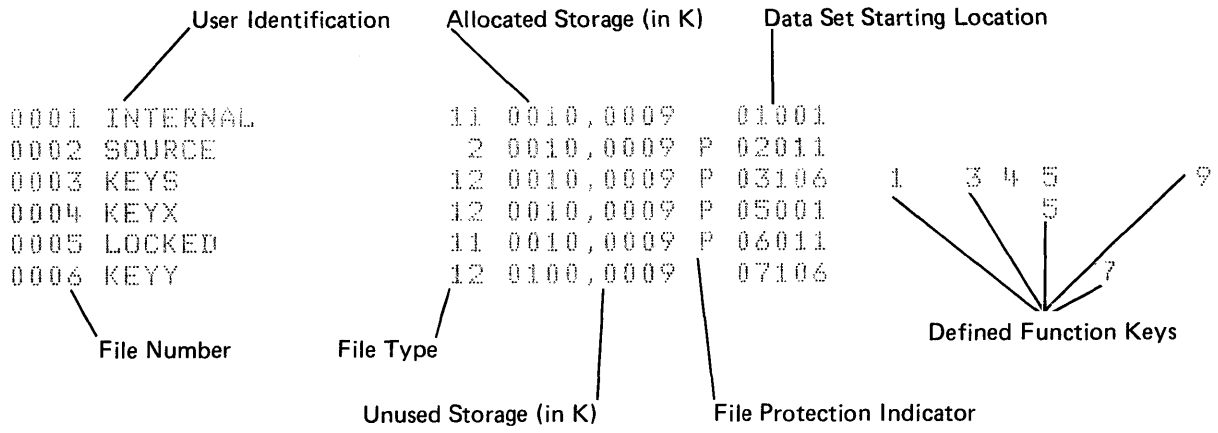
The resulting output for tape is shown in the following example:



For a diskette file directory, a UTIL statement is:

```
UTIL DIR1,D80
```

The resulting output for diskette is shown in the following example:



For additional information about directory listings, see the *IBM 5110 BASIC Reference Manual*, SA21-9308.

The following commands have been discussed in this chapter:

AUTO – Automatically provides BASIC statement numbers

MARK – Prepares a tape cartridge or diskette for data or programs to be saved

LOAD – Loads the 5110 storage with data from tape, diskette, or from the keyboard

SAVE – Saves the BASIC program in 5110 storage on tape or diskette

RD= – Specifies rounding of decimal numbers

RUN – Executes a BASIC program

UTIL – Lists a directory of tape or diskette file information

A REVIEW OF WHAT YOU'VE DONE

After reading this far and doing the exercises described, you should be able to perform the following functions with your system:

- Use as a calculator
  - Addition
  - Subtraction
  - Multiplication
  - Division
  - Exponentiation
  - Use of positive/negative operators
  - Use of parentheses in arithmetic hierarchy
  - Use of arithmetic constants

- Correct keying errors
  - Replace a character
  - Delete a character
  - Insert a character
  - Make corrections in a line

- Use variables
  - Assign values to variables
  - Display variable values

- Enter short, simple programs
  - Change program lines
  - Run programs
  - Erase programs
  - Store programs on tape or diskette
  - Load programs from tape or diskette

- Clear the machine storage

Chapter 4. How to Write a Program

In the following pages, you are going to write more BASIC programs and learn to use some fundamental tools for writing programs. From this point on in the manual we will not show you the keys to press. We will just say to enter and then give you the data you should enter. Exercises for this chapter are provided in Chapter 13.

The LET Statement

A LET statement consists of four parts: a statement number, a variable to the left of an equal sign, an equal sign, and a quantity or a computation (called an *expression*) to the right of the equal sign. In BASIC programming, a LET statement means:

1. Evaluate the expression on the right side of the equal sign, and

2. *Assign* that value to the variable on the left side of the equal sign.

In BASIC, you can have statements such as

```
0030 LET X=X+1
```

while you couldn't in math. In BASIC this statement means to take whatever value X now has, add 1 to it, and replace the old value of X with this new value.

Incidently, you can omit the word LET from a LET statement in a program. These two statements

```
0010 LET X=A+B
```

```
0010 X=A+B
```

mean exactly the same thing. Most of our examples will not show the word LET because it's not necessary to include it.

The following program uses simple arithmetic. Try to look at the program as a step-by-step method for solving a particular problem.

Problem

Last month you went to the dentist and had an examination and X-rays. That cost $25. You had two teeth filled. That cost $24. Your insurance will pay for 75% of everything over $15. How much do you have to pay, and how much does the insurance pay?

What to Do

1. Find the total dentist bill (call it D).

2. Subtract $15 to find the amount eligible for insurance (call it E).

3. Take 75% of the result (call it I). That's how much the insurance pays.

4. Subtract the insurance money from the total bill D to find out how much money you owe (call this M).

5. Display how much you have to pay and how much the insurance will pay (M and I).

The following BASIC statements, which you will enter later, can be used to solve this problem:

```
0010 D=25+24
0020 E=D-15
0030 I=.75*E
0040 M=D-I
0050 PRINT M,I
0060 STOP
```

Notice the PRINT statement. Because you want to know both your payment and the insurance payment, you can specify both M and I in the same statement. Any time you want to display the value of more than one variable, you can use a single PRINT statement if you list the variables and separate them with commas or semicolons.

USING REMARKS

You can make your programs easier to work with, and easier for other people to use, if you include descriptions of what the statements do in the program. These descriptions are known as *remark statements*. You write remarks as if they were statements in the BASIC program, but they don't serve any function in the execution of the program. They are solely for information. You can insert them anywhere in a program.

To include a remark in a program, you write a BASIC statement called REM. It has a line number like any other BASIC statement. Following the line number, you enter the letters REM followed by any remark you want. Examples of REM statements are:

```
40 REM THIS PROGRAM COMPUTES BATTING AVERAGES

70 REM AT THIS POINT, PRINT OUT THE RESULTS

10 REM DENTBILL
```

You'll see other examples of REM statements as you go through this manual.

You should now be ready to enter the program from the system keyboard. To enter and execute the program, follow the instructions below. Remember to press EXECUTE after each line is entered.

Instructions	**Display Screen Shows**
Clear storage	`LOAD0`

Enter the statements	`0010 REM DENTBILL`
	`0020 D= 25*24`
	`0030 E=D-15`
	`0040 I=.75*E`
	`0050 M=D-I`
	`0060 PRINT M,I`
	`0070 STOP`

Run the program	`RUN` `  23.5  25.5`

Remember that the idea in this program, and any other programs you write, is to break down what you want to do into logical sequential steps. It may help to use this tactic: Ask yourself what is the very first thing I have to do? What is the next? And so on. You can make a list of what you have to do and then convert each item in the list to a BASIC statement. Thus, you will always have a sequence of statements that will solve the problem in an orderly step-by-step fashion.

The dentbill program is not a typical program because it works on only one set of data that is a part of the program. Most programs are written to use many sets of data and use data that is not a direct part of the program. Later program examples will explain this in detail.

LISTING PROGRAM CONTENTS

Because the dentbill program is currently in the system work area, we can now list the statements of the program on the display screen with the LIST system command. Enter LIST, then press EXECUTE. The statements of the dentbill program now appear on the display screen.

The LIST command causes the first 14 lines of the program to be displayed. You can then use the scroll up key to view additional lines of the program.

It isn't necessary to list the entire program each time you want to see a particular part of it. You can list any 14-line portion of the program by entering the last line number you want displayed after the command keyword LIST. For example, LIST 30 would display statement numbers 10, 20, and 30 of the dentbill program, with statement 30 on line 1.

BRANCHES

The system normally executes programs line by line according to the line numbers of the statements. However, you can vary this sequential order and transfer control to a line number other than the next sequential one. This is called *branching*. Two of the statements you can use for branching are the GOTO and IF statements.

The GOTO Statement

This statement tells the system to go to a specific line number. A GOTO statement at line 20 of a program that tells the system to go to line number 60 would look like this:

```
0020 GOTO 60
```

The IF Statement

An IF statement can test whether a variable is equal to, greater than, or less than another variable or constant of the same data type. The IF statement includes a GOTO statement. The IF statement operates this way:

1. The IF statement tests the condition you define.

2. If the answer to the test is *yes*, the condition is true; the system will go to the line number that you entered in the IF statement.

3. If the answer is *no*, the system ignores the rest of the IF statement and goes directly to the next sequential line in the program.

Here's an example of an IF statement:

```
0040 IF X=0 GOTO 80
```

In this statement, if X is 0, the system goes to line 80. If X is *not* 0, it goes on to the next line in the program.

The six tests you can make with the IF statement are:

- Equal to, =
- Not equal to, ≠ or < >
- Greater than, >
- Less than, <
- Greater than or equal to, ≥ or > =
- Less than or equal to, ≤ or < =

The system stores < > as ≠, >= as ≥, and < = as ≤; thus, even though you enter < > the system will display ≠ when you list the statements.

Some examples of IF statements are:

This IF Statement:	Means:
0130 IF X>10 GOTO 40	If the value of X is greater than 10, go to line 40.
0190 IF Y<21 GOTO 10	If the value of Y is less than 21, go to line 10.
0010 IF A1≥5 GOTO 60	If the value of A1 is greater than or equal to 5, go to line 60.
0030 IF A2≠X GOTO 75	If the value of A2 is not equal to the value of X, go to line 75.

The following program examples describe more about how to break down a problem into the BASIC statements required to use your system to solve a problem. Again, as opposed to most typical programs, the sample programs will use data internal to the programs. After you've seen how data within a program can be manipulated, you'll be shown how to supply program data from outside the program.

Although it may not be necessary in all instances, it is a good idea to enter a LOAD0 command before entering any program statements. This ensures that the system work area is clear. Remember also that you can use the AUTO command to provide automatic statement numbering.

Program Example 1

You are in charge of billing people for orders of dresses. There are two styles, one at $108 a dozen and one at $136 a dozen. On orders of $500 or over, there is a 10% discount. For the account you are now working on, there are two dozen orders for the first dress and three dozen orders for the second dress.

The program to determine the bill is:

```
0010 REM PROGRAM TO FIGURE OUT DISCOUNTS ON ORDERS
0020 A=2
0030 B=3
0040 T=A*108+B*136
0050 IF T<500 GOTO 0070
0060 D=.1*T
0070 PRINT T,D,T-D
0080 STOP
```

This program solves the problem in the following steps:

1. It finds the total order (line 40).

2. It tests to see if the total is less than $500 (line 50). If it is, the program goes to line 70 and displays the total. The discount D will be 0 in this case, and the totals will be displayed.

3. For orders of $500 or over, the program computes the 10% discount on line 60. Then it continues to line 70 to display the total.

Note: D will be 0 when the order is less than $500 because each time the system starts to execute a program after a RUN command, it automatically sets the value of all numeric variables to 0. Character variables are set to blanks. This is called *initialization*. The values remain zeros or blanks until a statement in the program assigns a different value. This means that you never have any problem with values being left over from the last time you ran the program.

Upon execution of this program, the display screen shows:

```
624                62.4                561.6
```

Thus, the total order is $624, the allowable discount is $62.40, and the amount to be billed is $561.60. The cents columns in the dollar figures do not print because the system has no way of determining how many significant digits you want printed. You will be shown how to have numbers printed in the exact format you want in a later chapter.

Program Example 2

You are moving. You get estimates from two movers and want to know which mover will be cheaper. Mover A charges $40 an hour and estimates that the work will take 5 hours. Mover B charges $32.50 an hour and estimates that the work will take 8 hours. If both movers cost the same, you'll hire Mover B because he has a better reputation. Here is the program:

```
AUTO
0010 A=5*40
0020 B=8*32.5
0030 REM TEST TO SEE WHO IS CHEAPER
0040 IF A<B GOTO 0100
0050 REM GO HERE IF B CHEAPER
0060 PRINT 'B CHEAPER OR EQUAL
0070 PRINT B
0080 STOP
0090 REM GO HERE IF A CHEAPER
0100 PRINT 'A CHEAPER'
0110 PRINT A
0120 STOP
```

Here is how this program works:

1. In lines 10 and 20, it figures the total cost for each mover.

2. In line 40, it tests to determine which one of two paths to take. Either the program will go to line 100, or it will continue with lines 50, 60, 70, and 80. This test determines which mover is cheaper.

3. If Mover A is cheaper, the program goes to line 100. Line 100 lets you know that Mover A has the contract and displays the total price. Notice line 100. It is a PRINT statement, but it has single quotation marks around the words A CHEAPER. You can write a PRINT statement that displays the words entered if you enclose the words in single quotation marks. If line 100 is executed, the words A CHEAPER will be displayed. Line 110 has no quotation marks. It is a PRINT statement for variable A, and will display the value of variable A. After the PRINT statements, the program ends at line 120.

4. If Mover B is cheaper, the program continues with line 50. Line 60 is a PRINT statement containing the words B CHEAPER OR EQUAL in single quotation marks. If line 60 is executed, the words B CHEAPER OR EQUAL will be displayed. Line 70 will display the value of variable B. After the PRINT statements, the program comes to line 80, a STOP statement, which ends the program.

After you run the program, the display screen shows:

```
RUN
? CHEAPER
 200
```

Loops

Here is a new problem. You are a rug salesperson. All your rugs come in rolls 12 feet wide. Your customers buy rugs in varying lengths depending on how long their rooms are. You want to make a chart of how many square yards of rug are required for rooms of different lengths. The most popular room sizes start at 9 feet long (a 12 by 9 foot rug) and increase a foot at a time (12 by 10, 12 by 11, and so on) until they reach 12 by 20.

You will write a program that computes the number of yards in a 12 by 9 rug, then a 12 by 10 rug, on up to a 12 by 20 rug.

To compute the number of square yards in each of the 12 different sized rugs, you have to find the number of square feet and divide by 9. The main computation step is:

```
0020 LET Y=(12*X)/9
```

where Y is the number of square yards, and X is the length of each different rug. To display the value of Y, you would use this statement:

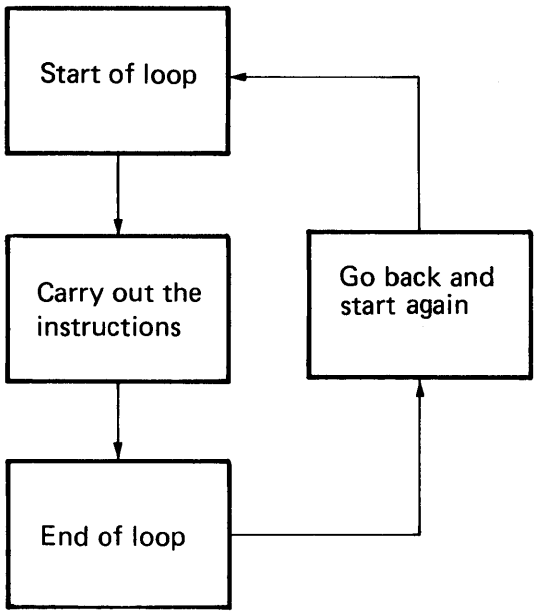
```
0030 PRINT Y
```

The value of X has to increase by 1, from 9 to 10 on up to 20. You could write a program like this:

```
0010 X=9
0020 Y=(12*X)/9
0030 PRINT Y
0040 X=10
0050 Y=(12*X)/9
0060 PRINT Y
   .
   .
   .
```

and so on until X=20

This program uses a LET statement to increase the value of X. However, there is a better way. You can make a loop. A loop is just what it sounds like. It is a series of program steps that are repeated. It looks like this:



Two things have to happen to this loop to make it work. It has to have some way to change the values it uses before it loops up to the top and starts again. And it has to have some way to know when to stop, or the program will run indefinitely.

First Loop Method

So far you have these program statements:

```
0020 Y=(12*X)/9
0030 PRINT Y
```

You want to start with X equal to 9, so put a LET statement ahead of these two statements assigning 9 as the first value of X. It will also help if you print the value of X with the computed value of Y, so the table will be more self-explanatory. Now, the program looks like this:

```
0010 X=9
0020 Y=(12*X)/9
0030 PRINT X,Y
```

To avoid specifying X=10, X=11, and so on, write a general statement that will keep increasing the value of X by 1. That statement is:

```
0040 X=X+1
```

Remember that while this statement looks peculiar in a mathematical sense, it's perfectly valid in BASIC. It says: Assign the value of X to be equal to the old value of X plus 1.

By adding statement 40 to the program, you've changed the value of X and completed the steps required to make the loop operate once. Now you have to add a GOTO statement to go back to the beginning of the loop:

```
0050 GOTO 20
```

You go to line 20 because you only have to go back to the computation step, not to line 10 where you originally set X equal to 9.

After the system goes to line 20, it computes and displays the yardage again, but this time for X equal to 10. It arrives at line 40 again and changes X to 11; then it goes back to line 20 to compute the next yardage. This process continues, increasing the value of X by 1 after each loop.

Ending a Loop

One thing is missing from an otherwise perfect loop. It never ends. Not at X=20, not at X=30, because X just keeps increasing. If you are sitting in front of your system while this program is running, *you* can stop this loop whenever you want to by pressing ATTN. But this is obviously not an ideal method. You can make the loop stop automatically if you build in a test with an IF statement to see when you've processed enough values of X. In this program, you want the loop to stop when the value of X passes 20. Consider this IF statement:

```
0050 IF X>20 GOTO 70
```

Line 70 will be a STOP statement.

The IF test goes *before* the GOTO statement that branches to line 20. If you put it after the GOTO statement, it will never be executed. This is the finished program:

```
0010 X=9
0020 Y=(12*X)/9
0030 PRINT X,Y
0040 X=X+1
0050 IF X>20 GOTO 0070
0060 GOTO 0020
0070 STOP
```

Looking at this program, you should be able to see that lines 50 and 60 can be combined to make a more efficient program that looks like this:

```
0010 X=9
0020 Y=(12*X)/9
0030 PRINT X,Y
0040 X=X+1
0050 IF X≤20 GOTO 0020
0060 STOP
```

Now, enter and run the program. After you run the program, the display screen shows:

```
RUN
 9                    12
 10                   13.333333
 11                   14.666667
 12                   16
 13                   17.333333
 14                   18.666667
 15                   20
 16                   21.333333
 17                   22.666667
 18                   24
 19                   25.333333
 20                   26.666667
```

You can press HOLD to stop the upward movement of the data. To continue, press HOLD a second time.

There is another way to make a loop in a program. At the beginning of the loop, instead of setting X equal to its first value, you enter the entire range of values that X will use. In the rug example, you would write

```
0010 FOR X=9 TO 20
```

Then you write the statements that solve the problem and print the results:

```
0020 Y=(12*X)/9
0030 PRINT X,Y
```

Then you tell the system to go to the next value of X and repeat the loop:

```
0040 NEXT X
```

FOR and NEXT statements always go in pairs: FOR at the beginning of the loop and NEXT at the end. The system automatically repeats the loop as many times as you told it to in the FOR statement. When it finishes, it goes on to the statement following the NEXT statement.

Using the FOR and NEXT statements, the rug program looks like this:

```
0010 FOR X=9 TO 20
0020 Y=(12*X)/9
0030 PRINT X,Y
0040 NEXT X
0050 STOP
```

In a FOR statement, you can name any arithmetic variable to be the control variable, and you can make its range of values anything you want. The control variable is to the left of the equal sign. The range (to the right of the equal sign) doesn't have to be given in numbers. You can use other variables for the range, for example:

```
0060 FOR J=A TO B
   .
   .
   .
0120 NEXT J
```

Steps

When you write a FOR statement, the system increases the value in steps of 1 (for example, 1 to 2 to 3, or 18 to 19 to 20 to 21). However, sometimes you may want to use just even numbers, or odd numbers, or every tenth number. If your loop requires a value other than steps of 1, you can specify the step value whether you are using FOR and NEXT statements or a LET statement to control the loop.

If you write a loop that uses a LET statement, you can write these LET statements:

```
0100 X=X+2     To change X in steps of 2

0050 X=X+10    To change X in steps of 10
```

If you're using FOR and NEXT statements for the loop, you add the word STEP and the size of the step to the FOR statement. For example:

```
0010 FOR X=1 TO 25 STEP 2
```

gives you odd values of X from 1 to 25 (1, 3, 5, 7 . . .).

```
0030 FOR D=10 TO 100 STEP 10
```

gives you 10, 20, 30, up to 100.

For even values of D from 1 to 20, you would write:

```
0020 FOR D=2 TO 20 STEP 2
```

Notice that D is set to 2 because the first even number you want is 2.

If you omit the word STEP and the value from the FOR statement, you automatically get steps of 1. You can also include fractional steps, for example:

```
0030 FOR I=1 TO 3 STEP .1
```

Loops Within Loops

Here's a problem in which two values change: Find the annual amount of interest (A) at the interest rates (I) of 5%, 6%, 7%, 8%, 9%, and 10% on principals (P) ranging from $100 to $1000 in steps of $100.

This problem can be solved by a program that uses two loops–one for changing the interest and one for changing the principal. Do the interest loop first:

```
0030 FOR I=5 TO 10
0040 A=(I/100)*P
0050 PRINT P, I, A
0060 NEXT I
```

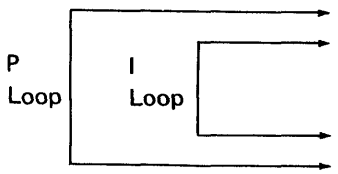
This computes and displays A, P, and I.

This creates a loop for I to vary from 5% to 10%.

Now all that's left is to define P, because the program doesn't know where to find the values for P. The P loop has no computations of its own; it only defines the values for P:

```
0020 FOR P=100 TO 1000 STEP 100
0070 NEXT P
```

The P loop goes *around* the I loop:




```
0020 FOR P=100 TO 1000 STEP 100
0030 FOR I=5 TO 10
0040 A=(I/100)*P
0050 PRINT P,I,A
0060 NEXT I
0070 NEXT P
0080 STOP
```

You must put one loop entirely inside the other so that the system will stay in one loop and finish it completely (compute *all* the values for I for a single value of P) before it goes on to the next value of P. In this program, the system starts with P equal to $100, then it comes to the I loop and sets I equal to 5%. It goes on to compute the interest on $100 at 5%, 6%, 7%, 8%, 9%, and 10% because it keeps repeating the I loop until I equals 10. When it finishes all the different interests on $100, it goes to line 70, which is the bottom of the P loop. Here, control loops back to line 20, which increases P to $200, and starts on the I loop again, this time with P equal to $200 and with I again ranging from 5% to 10%. The program continues in these loops until all of the values of P have been used. Then you have all the amounts of interest you wanted.

To run the program:

When the display screen shows READY, enter the statements:

```
LOAD 0
AUTO
0010 REM INTEREST
0020 FOR P=100 TO 1000 STEP 100
0030 FOR I=5 TO 10
0040 A=(I/100)*P
0050 PRINT P,I,A
0060 NEXT I
0070 NEXT P
0080 STOP
```

Run the program

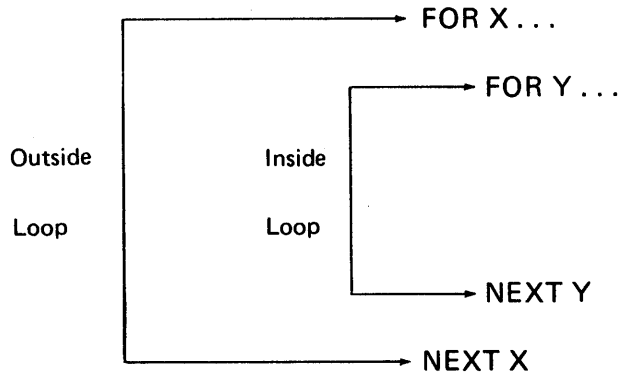
```
RUN
```

To see a portion of the program results, press HOLD. To continue execution, press HOLD again.

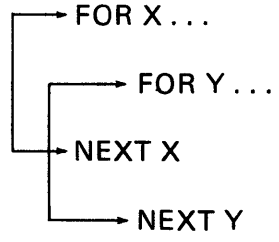
The display screen shows a portion of the output when you pressed HOLD. For example:

```
 800        10         80
 900         5         45
 900         6         54
 900         7         63
 900         8         72
 900         9         81
 900        10         90
1000         5         50
1000         6         60
1000         7         70
1000         8         80
1000         9         90
1000        10        100
```

Loops within loops must always be nested like this:

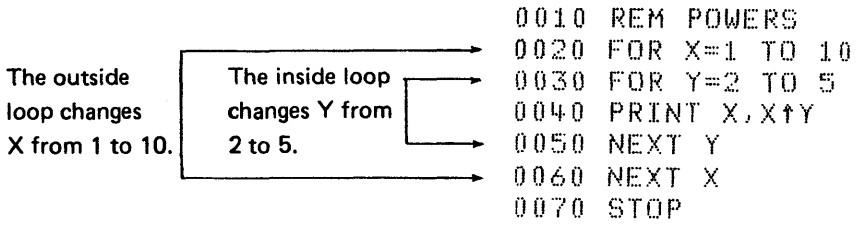


so that the inner loop is fully completed each time before the outside one is begun again. Two loops must never overlap like this:



Remember, one loop must always be completely enclosed by the other.

Here is another example of loops. This is a program to find X^2, X^3, X^4, and X^5 with X equal to 1 to 10.




```
0010 REM POWERS
0020 FOR X=1 TO 10
0030 FOR Y=2 TO 5
0040 PRINT X,X↑Y
0050 NEXT Y
0060 NEXT X
0070 STOP
```

Notice that the inside Y loop is fully contained in the outside X loop. Run the program as shown:

When the display screen shows READY, enter the statements:

```
0010 REM POWERS
0020 FOR X=1 TO 10
0030 FOR Y=2 TO 5
0040 PRINT X,X↑Y
0050 NEXT Y
0060 NEXT X
0070 STOP
```

Run the program

```
RUN
```

The display screen shows (use the HOLD key to display any 15-line portion of the displayed results):

```
 7             16807
 8             64
 8             512
 8             4096
 8             32768
 9             81
 9             729
 9             6561
 9             59049
 10            100
 10            1000
 10            10000
 10            100000
```

Chapter 5. Other Ways to Put Values into Programs

In all the programs we've written, we've tried to:

- Write a program to solve the problem using general expressions.
- Supply specific values for the expressions and run the program with the specific values.

The advantage of programming in this way is that the bulk of the program doesn't change every time you want to solve the same problem with different numbers. You only need to change the numbers, not the programmed expression, when you want to run the program using different numbers.

We are now going to look at other ways to supply specific numbers for programs. Exercises for this chapter are provided in Chapter 13.

THE READ, DATA, AND RESTORE STATEMENTS

To assign 10 values, say the numbers 1 through 10, to 10 variables, you could use 10 LET statements:

```
0010 LET A=1
0020 LET B=2
0030 LET C=3
   .
   .
   .
0100 LET J=10
```

Using 10 LET statements can be tedious. Another way to enter these numbers is with one DATA statement:

```
0200 DATA 1,2,3,4,5,6,7,8,9,10
```

The DATA statement causes values to be placed in an internal data table. You can use one or several DATA statements to do this. Values in DATA statements are put into the data table sequentially, in the order in which they are entered. The values must be separated by commas. The following set of statements would have the same effect as the single preceding DATA statement:

```
0200 DATA 1,2,3
0210 DATA 4,5,6
0220 DATA 7,8,9,10
```

Once the values are in the table, you use the READ statement to assign them to variables. Here's an example:

```
0200 DATA 1,2,3,4,5,6,7,8,9,10
0210 READ A,B,C,D,E,F,G,H,I,J
```

The READ statement locates the values in the data table and assigns them (in order) to the variables–the value 1 to the variable A, 2 to B, 3 to C, and so on.

You don't have to assign all of the values in the data table at one time. For example:

```
0200 DATA 1,2,3,4,5,6,7,8,9,10
0210 READ A,B,C
```

will cause the first three values in the table to be assigned to A, B, and C, respectively. Another READ statement will take up where the last one left off. Thus:

```
0420 READ D,E,F,G
```

will assign the values 4, 5, 6, and 7 to D, E, F, and G, respectively.

You must be careful, though, not to try to read more values than the table contains. For example, still another READ statement:

```
0440 READ H,I,J,K
```

would be requesting values for four variables when only three numbers (8, 9, and 10) are left in the table. This will cause an error.

If you want, you can use the values in the data table more than once. At any point in your program, you can instruct the system to assign from the beginning of the table again, even if you haven't read all the values in the table. To go back to the beginning of the table, use the RESTORE statement:

```
0100 RESTORE
```

Let's assume that you want to assign the values 1, 2, and 3 to three variables A, B, and C, in that order. Then later in the program you want to assign the same values to D, E, and F. These statements will do just that:

```
0030 DATA 1,2,3,4,5,6
   .
   .
   .
0060 READ A,B,C

   .
   .
   .

0100 RESTORE READ FROM START OF DATA TABLE
0110 READ D,E,F
```

Notice that you can include a comment in the RESTORE statement. The words READ FROM START OF DATA TABLE have no effect on what your program is doing; they merely serve as a reminder to you, when you look at the program, of what the RESTORE statement is doing. Your comment can say anything you want it to say, as long as it fits on one line with the RESTORE statement.

It's important to remember, when using READ and DATA statements, that no matter how many DATA statements you include in your program, only *one* data table is created before any READ statement is executed. The table is created from all the DATA statements in your program, regardless of where they appear–at the beginning, at the end, or scattered throughout. Each of the following three sets of statements has the same effect:

```
0200 DATA 1,2,3
0210 DATA 4,5,6
0220 READ H,I,J,K,L,M

0200 READ H,I,J,K,L,M
0210 DATA 1,2,3
0220  DATA 4,5,6

0200 DATA 1,2,3
0210 READ H,I,J,K,L,M
0220 DATA 4,5,6
```

THE INPUT STATEMENT

Both the assignment statement (LET) and the DATA statement use constants–unchanging data items that are part of your program–to assign values to variables. You have to know, at the time you're writing your program, what values you want to assign.

The INPUT statement allows a little more flexibility. This statement names the variables that are to receive values, but allows you to wait until you are running your program to actually supply the values. For example:

```
0050 INPUT X,Y,Z
```

means that you will supply values from the keyboard for X, Y, and Z when your program is run. You'll know when it's time to supply the values because a flashing question mark will be displayed. When you see this, you should enter your values, one for each variable in the INPUT statement–in this case, three. The values are entered all on one line, separated by commas. Thus, when you've entered the information, the display screen shows:

```
185,205,191
```

By entering these numbers, you've assigned 185 to X, 205 to Y, and 191 to Z.

You have to be certain, when entering your values, to enter exactly the same number of values as there are variables in the INPUT statement in your program. The question mark will keep flashing until the correct number of values is entered. If you enter too many values, the excess values are ignored. After the last value is entered, press EXECUTE to continue program execution.

Prompting Your Input

Because a lot of time can elapse between the time you write a program and the time you run it, you may have difficulty remembering exactly how many values you have to enter. This is especially true when your program contains more than one INPUT statement. Then you have to keep track of which one comes first.

You can have your program keep track for you by reminding you what has to be entered. All you have to do is include a PRINT statement immediately before the INPUT statement in your program. For example, if your program averages bowling scores, you could use these statements:

```
0045 PRINT 'ENTER THREE BOWLING SCORES'
0050 INPUT X,Y,Z
```

Then, when the program is run, instead of just a question mark appearing when it's time to enter your values, these lines will be displayed:

```
ENTER THREE BOWLING SCORES

?
```

When you've entered your values, the display screen will show:

```
ENTER THREE BOWLING SCORES

185,205,199
```

You can write any reminder message that you want in the PRINT statement, as long as you enclose it in single quotation marks.

You also have to remember that the PRINT statement has to fit entirely on one line. If your message is so long that it doesn't fit, you might consider using several consecutive PRINT statements:

```
0040 PRINT 'ENTER 12 AVERAGE TEMPERATURES'
0050 PRINT 'FOR JANUARY TO DECEMBER'
0060 INPUT M,N,O,P,Q,R,S,T,U,V,W,X
```

ENTERING CHARACTER VARIABLES INTO PROGRAMS

You've been entering numeric variables into programs in this section, but any of the methods you've used will let you supply values for character variables as well. You have already seen how to do this with a LET statement. For INPUT and READ statements, you just use valid character variables where we've been using numeric variables. Note that you need not enclose character data in single quotation marks when you enter it for a DATA statement or in response to the flashing question mark for an INPUT statement, unless the data must include a quotation mark, a comma, or leading blanks. For example, if you want a program to keep track of a person's height and weight, you can enter the person's name, height, and weight with these READ and DATA statements:

```
0010 READ N$,H,W
0020 DATA TOM JONES,6.1,184
```

You could also use an INPUT statement:

```
0010 INPUT N$,H,W
```

and then respond to the flashing question mark like this:

```
TOM JONES,6.1,184
```

A REVIEW OF WHAT YOU'VE DONE

All of the following methods of assigning values to variables are useful:

- LET statements
- READ, DATA, and RESTORE statements
- INPUT statements with data supplied from the keyboard

You can use a combination of these methods if you have a program where some values don't change, some change occasionally, and others change often.

Chapter 6. Making Changes to Your Programs

It is very important that you be able to make changes to your programs. You may have to change a program to supply values for variables, to make corrections, to add lines, or to remove lines. There are several ways you can change a program, either as you write it or after you write it.

CORRECTING KEYING ERRORS

If you make mistakes while entering your program statements or commands, you already know how to fix them. As you catch the errors, you can:

- Use the backspace or forward space key to position the cursor at the incorrect character, then simply enter the correct character.
- Use the insert or delete function to insert or delete characters.
- Use the scroll up and scroll down keys to position a line to be corrected.
- Press ATTN to delete all characters starting with and to the right of the cursor position.

INSERTING NEW LINES

The following program, called phone, computes charges for local telephone calls. The rate for local calls in this example is 10 cents for the first three minutes or less, and 2 cents for each additional minute or fraction of a minute. We'll write a general program, but we'll purposely omit the actual length of any call. These are the variables we'll use:

T	Total length of the call in whole minutes
T1	Amount of time over 3 minutes
C	Charge for the call

The program is:

```
0010 IF T>3 GOTO 0040
0020 PRINT 'CALL LESS THAN OR EQUAL 3 MIN, 10 CENT CHARGE'
0030 GOTO 0100
0040 T1=T-3
0050 C=.1+.02*T1
0060 PRINT 'LENGTH OF CALL'
0070 PRINT T
0080 PRINT 'CHARGE FOR CALL'
0090 PRINT C
0100 STOP
```

Enter this program. After we add a statement to assign a value to T, you'll be able to run the program.

To assign a value to T, you can use READ and DATA statements:

```
READ T

DATA 8
```

(for an 8 minute call)

You can insert these statements before line 10.

Now enter:

```
5 READ T
```

Press EXECUTE and enter:

```
6 DATA 8
```

and press EXECUTE again.

To see what has been done with these statements, enter the LIST command.

The display screen shows:

```
0005 READ T
0006 DATA 8
0010 IF T>3 GOTO 0040
0020 PRINT 'CALL LESS THAN OR EQUAL 3 MIN. 10 CENT CHARGE'
0030 GOTO 0100
0040 T1=T-3
0050 C=.1+.02*T1
0060 PRINT 'LENGTH OF CALL'
0070 PRINT T
0080 PRINT 'CHARGE FOR CALL'
0090 PRINT C
0100 STOP
```

The system has taken the two lines and inserted them in the program (as lines 5 and 6) before line 10. By entering a line number and any valid BASIC statement, you have given an instruction. This instruction starts with a line number, and tells the system you are adding a line and where to add it.

Now you can see why it is convenient to have the line numbers increase by 10's; it gives you the chance to insert up to nine new lines between every two original lines.

You can now run the phone program by entering the RUN command.

REPLACING ONE LINE WITH ANOTHER

Let's try a different value for T in the phone program. This time T is 21 minutes. You'll have to change line 6, the DATA statement, to use this new value.

Enter the following statement, then press EXECUTE:

```
6 DATA 21
```

If you list the program now, the display screen shows:

```
0005 READ T
0006 DATA 21
0010 IF T>3 GOTO 0040
0020 PRINT 'CALL LESS THAN OR EQUAL 3 MIN, 10 CENT CHARGE'
0030 GOTO 0100
0040 T1=T-3
0050 C=.1+.02*T1
0060 PRINT 'LENGTH OF CALL'
0070 PRINT T
0080 PRINT 'CHARGE FOR CALL'
0090 PRINT C
0100 STOP
```

See what happened? The system replaced the old line 6 with the new line 6.

When you want to replace a line, simply enter the same line number as the line you want to replace and enter the new line. The system replaces the old line in storage with the new one after you press EXECUTE.

Remember that you can use the SAVE command if you want to save the program on tape or diskette.

REMOVING A LINE

In the phone program, we will now include an INPUT statement so we can run the program with many changing values for T. We can replace the READ statement with an INPUT statement, but the DATA statement must be deleted. To do this, first list the program. Now enter the number of the line you want to delete, then enter DEL and press EXECUTE. To delete line 6, enter 6 DEL, then press EXECUTE. Line 5 can be replaced by the following procedure: enter 5 INPUT T and press EXECUTE. List the program again, and the display screen shows:

```
0005 INPUT T
0010 IF T>3 GOTO 0040
0020 PRINT 'CALL LESS THAN OR EQUAL 3 MIN, 10 CENT CHARGE'
0030 GOTO 0100
0040 T1=T-3
0050 C=.1+.02*T1
0060 PRINT 'LENGTH OF CALL'
0070 PRINT T
0080 PRINT 'CHARGE FOR CALL'
0090 PRINT C
0100 STOP
```

The system has replaced line 5 and deleted line 6.

When you want to delete a line, simply enter the statement number, then enter DEL, and press EXECUTE. A new listing of the program will show the line deleted. You can also use the DEL function to delete several lines. For example, you could delete lines 0070 through 0090 by entering:

```
0070 DEL 0090
```

RENUMBERING STATEMENT LINES

In the phone program, the statement numbers are not sequential by 10's. If you want the numbers to start with 0010 and increase by 10, you can simply use the RENUM command. This command will assign the number 0010 to the INPUT T statement and number the remaining statements from 0020 to 0110.

In addition, the GOTO statements (original lines 10 and 30) will be altered to transfer execution to the appropriate renumbered statement. To see the result of a renumber operation, list the phone program, then enter RENUM and press EXECUTE. After you list the program again, the display screen shows:

```
0010 READ T
0020 IF T>3 GOTO 0050
0030 PRINT 'CALL LESS THAN OR EQUAL 3 MIN, 10 CENT CHARGE'
0040 GOTO 0110
0050 T1=T-3
0060 C=.1+.02*T1
0070 PRINT 'LENGTH OF CALL'
0080 PRINT T
0090 PRINT 'CHARGE FOR CALL'
0100 PRINT C
0110 STOP
0120 PRINT A
```

We've seen the PRINT statement used to display the values of variables and to display comments exactly as entered in the statement. We've also seen that to display a comment exactly as you entered it, you must enclose the comment in single quotation marks. You should remember, then, that if you include this line in a program:

```
0050 PRINT 'X'
```

when line 50 is executed, the display screen will show:

```
X
```

and *not* the value of X, which would be displayed if line 50 were

```
0050 PRINT X
```

Within a single PRINT statement, you can mix character and arithmetic variables and constants. You must use commas or semicolons to separate the values to be displayed. These separators (delimiters) control spacing of the displayed data. For this example, no separator is required between the variables and character constants. The only instance in which a comma or semicolon is not required is between a character constant and a variable, as in this example.

Here's an example of a program that computes annual interest for any rate and principal that you enter:

```
0010 INPUT R,P
0020 I=(R/100)*P
0030 PRINT 'THE ANNUAL INTREST AT 'R' PERCENT ON $'P' IS $'I
0040 STOP
```

When you run this program and use values of 7 and 825, here's what you see:

```
RUN
7,825
THE ANNUAL INTREST AT  7  PERCENT ON $ 825  IS $ 57.75
```

If your system has an attached printer, you can specify that the data be printed by entering PRINT FLP in place of PRINT in line 30. All of the capabilities of and restrictions for the PRINT statement also apply to PRINT FLP. A comma must separate the FLP and the first value.

MAKING HEADINGS

Suppose you have a loop in a program that computes mileage allowances (at 12 cents a mile) for company auto trips (of 10 to 50 miles in steps of 5 miles):

```
0010 FOR X=10 TO 50 STEP 5
0020 PRINT X, .12*X
0030 NEXT X
0040 STOP
```

When you run this program, the display screen shows:

```
RUN
 10            1.2
 15            1.8
 20            2.4
 25            3
 30            3.6
 35            4.2
 40            4.8
 45            5.4
 50            6
```

You can make headings for these columns by entering a PRINT statement before the loop:

```
0005 PRINT 'MILES','MILEAGE ALLOWANCE'
```

When you run the program again, the display screen shows:

```
RUN
MILES          MILEAGE ALLOWANCE
 10             1.2
 15             1.8
 20             2.4
 25             3
 30             3.6
 35             4.2
 40             4.8
 45             5.4
 50             6
```

You will be shown later how you can change the mileage allowance column to include trailing zeros, which will make it more readable as dollars and cents.

MATH CALCULATIONS IN PRINT STATEMENTS

The PRINT statement allows you to include math calculations along with variables and words. Therefore, if you just want a calculation done and the result displayed, you can do it in a single PRINT statement. For example, you can write

```
0010 INPUT X
0020 PRINT X,X↑2
```

instead of writing

```
0010 INPUT X
0020 Y=X↑2
0030 PRINT X,Y
```

Chapter 8. Setting Up Your Own Format–PRINT USING and Image Statements

A more flexible way to display results is to use a statement called PRINT USING. This statement allows you to display variables using a particular format. You specify the format in a separate BASIC statement called an image or FORM statement, or you can assign the format to a character variable in an assignment statement. The PRINT USING statement is used together with an image or FORM statement or character variable. The image/FORM statement can appear anywhere in a program and can be used by any number of PRINT USING statements. A character variable used with a PRINT USING statement must have been previously defined with a DIM and/or assignment statement. For details on the FORM statement, see the *IBM 5110 BASIC Reference Manual*, SA21-9308. Exercises for this chapter are provided in Chapter 13.

The image statement is a BASIC statement, but it looks different from the other BASIC statements. Following the statement number is a colon (:). After the colon, you enter the exact wording and format that you want your results to have. You leave room for any variable values by using # signs where the values belong. When the program is run, the system substitutes real values for the # signs and displays the data in the image statement (including any blank spaces you leave). A sample PRINT USING statement and its image statement are:

```
0020 PRINT USING 30,G,N
0030 :GROSS SALES ARE ###.## AND NET PROFITS ARE ###.##
```

Notice that single quotation marks were not used around the character data in line 0030. This is true of all image statements, unless, of course, you want quotation marks to appear in the displayed or printed data.

This is the way these two statements work together:

```
0020 PRINT USING 30,G,N
```

The line number of the image statement.

A comma, which is required.

The variables, separated by commas, whose values are going to be displayed or printed in the format specified in the image statement.

```
0030 :GROSS SALES ARE ###.## AND NET PROFITS ARE ###.##
```

This identifies the image statement; it does not appear when the statement is executed.

The value of G will be inserted here when the line is displayed or printed instead of ###.##.

The value of N will be inserted here when the line is displayed or printed instead of ###.##.

If these two statements were part of a program, with G equal to 103.72 and N equal to 21.45, at the time you ran the program, the system would display:

```
GROSS SALES ARE 103.72 AND NET PROFITS ARE 21.45
```

Display Position 1

You can also specify a character variable in place of the line number of an image or FORM statement in the PRINT USING statement. For example:

```
0020 A$='GROSS SALES ARE ###.## AND NET PROFITS ARE ###.##'
0030 PRINT USING A$,G,N
```

Before being used in this manner, character variables must have been previously dimensioned to the length of the image (see *Arrays*, Chapter 11) in a DIM statement.

An image statement always begins with a colon. When you enter the # signs, as stand-ins for variable values, you are really telling the 5110 how many spaces to leave for the values. If a value needs more space than you indicated, a row of asterisks will be displayed or printed instead of the value when the statement is executed. In these statements:

```
0040 PRINT USING 50,Z,Z↑3
0050 :Z IS ## AND Z CUBED IS ##
```

there are only two spaces indicated for each value. If Z is 3, the displayed result would be:

```
Z IS 3 AND Z CUBED IS 27
```

But if Z is 5, then 5^3 is 125, which is three spaces long. The displayed result would be:

```
Z IS 5 AND Z CUBED IS **
```

The asterisks mean that the answer was too long for the space you indicated.

When you use # signs to indicate space for your variable values in an image statement, you can also control how many decimal places you want the value to have when it is displayed or printed. You do this by inserting a decimal point in the string of # signs wherever you want the decimal point to go in the result. For example, when you are using dollars and cents, insert a decimal point two places from the right, as in this example:

```
0100 PRINT USING 110,S
0110 :THE SERVICE CHARGE FOR YOUR ACCOUNT IS $###.##
```

If S were 23.47, the result would be displayed as follows when you ran the program:

```
THE SERVICE CHARGE FOR YOUR ACCOUNT IS $ 23.47
```

If your system has a printer, you can use the PRINT USING FLP statement to format your data. All the capabilities of the PRINT USING statement also apply to the PRINT USING FLP statement.

Example of Printing

There are three sales representatives in your department, and you are responsible for making a monthly report showing their sales figures for each week of the month. You can write a program that will automatically print the report for you in an attractive format.

This is how the variables in the program are named. Each sales representative is assigned a letter: A, B, and C. Starting with the first sales representative, the variables are:

A$ – Name
A1 – Sales total for week 1
A2 – Sales total for week 2
A3 – Sales total for week 3
A4 – Sales total for week 4
A5 – Sales total for the month
.
.
.
and so on for sales representatives B and C

There are these other variables:

M$	–	Name of the month
W$, X$, Y$, and Z$	–	Last day of each week in the month
T1, T2, T3, and T4	–	Totals for everybody for each week
T5	–	Total for everybody for the month

For this program, you have to supply the month's sales figures, the name of the month, and the last day of each week covered. The report is then printed automatically. Now enter the following program:

```
0010 REM PROGRAM FOR PRINTING MONTHLY SALES REPORT
0020 A$='ADLER'                                          Supply sales
0030 B$='BIPPLE'                                         representatives
0040 C$='CUBBINS'                                        names.
0050 PRINT 'ENTER MONTH, LAST DAY OF EACH WEEK'          Enter dates
0060 INPUT M$,W$,X$,Y$,Z$                                for the weeks.
0070 PRINT 'ENTER FIGURES FOR ADLER'
0080 INPUT A1,A2,A3,A4
0090 PRINT 'ENTER FIGURES FOR BIPPLE'                    Supply sales
0100 INPUT B1,B2,B3,B4                                   figures.
0110 PRINT 'ENTER FIGURES FOR CUBBINS'
0120 INPUT C1,C2,C3,C4
0130 A5=A1+A2+A3+A4                                      Compute
0140 B5=B1+B2+B3+B4                                      monthly
0150 C5=C1+C2+C3+C4                                      totals.
0160 PRINT USING 0170,FLP,M$
0170 :MONTHLY SALES REPORT FOR MONTH OF ############
0180 PRINT FLP
0190 PRINT USING 0200,FLP
0200 :SALESMAN              WEEK ENDING                          TOTAL
0210 PRINT USING 0220,FLP,W$,X$,Y$,Z$
0220 :                ##          ##          ##          ##
0230 PRINT FLP
0240 PRINT USING 0270,FLP,A$,A1,A2,A3,A4                 Print figures
0250 PRINT USING 0270,FLP,B$,B1,B2,B3,B4                 and sales
0260 PRINT USING 0270,FLP,C$,C1,C2,C3,C4                 representatives
0270 : ########  #####.##   #####.##   #####.##   #####.##   #####.##  names.
0280 T1=A1+B1+C1                                         Compute
0290 T2=A2+B2+C2                                         weekly and
0300 T3=A3+B3+C3                                         grand totals.
0310 T4=A4+B4+C4
0320 T5=A5+B5+C5
0330 PRINT FLP                                           Print totals.
0340 PRINT USING 0350,FLP,T1,T2,T3,T4,T5
0350 :TOTALS   ######.##  ######.##  ######.##  ######.##  ######.##
0360 STOP
```

Note that a PRINT FLP statement with nothing after it causes a blank line to be printed. This form of the statement is used in lines 0180, 0230, and 0330 to include blank lines in the printed report.

If your system has a printer, you can run the program by entering a RUN command and pressing EXECUTE. If your system does not have a printer, enter a RUN command, then enter P=D. For example: RUN P=D, then press EXECUTE. This command directs all printed output to the display screen.

During a sample running of this program, the display screen showed:

```
RUN
ENTER MONTH, LAST DAY OF EACH WEEK

 JULY , 7 , 14 , 21 , 28
ENTER FIGURES FOR ADLER

12.50,500.00,400.00,895.50
ENTER FIGURES FOR BIPPLE

34.50,78.90,500.00,100.00
ENTER FIGURES FOR CUBBINS

300.00,800.00,700.00,43.25
```

For this program the printed or displayed output was:

```
MONTHLY SALES REPORT FOR MONTH OF JULY

SALESMAN                WEEK ENDING                          TOTAL
                 7          14          21          28

 ADLER        12.50      500.00      400.00      895.50     1808.00
 BIPPLE       34.50       78.90      500.00      100.00      713.40
 CUBBINS     300.00      800.00      700.00       43.25     1843.25

TOTALS       347.00     1378.90     1600.00     1038.75     4364.65
```

Chapter 9. Data Files

A file is a collection of related data items that are stored together. Your system can process both stream-oriented files and record-oriented files. In a stream-oriented file, all the data items are stored as a sequential stream of data, in the order in which they are entered. Data items in a record-oriented file can also be stored sequentially in the order they are entered, but they can be retrieved according to an identification field called a key or logical record number.

For a complete description of stream/record-oriented files, see the *IBM 5110 BASIC User's Guide*, SA21-9307.

ACTIVATING AND DEACTIVATING FILES

Files must be activated or *opened* before they can be used within a program. A stream-oriented file must be opened by an OPEN statement in a program. A record-oriented file must be opened by an OPEN FILE statement. The following example shows the format of an OPEN statement:

```
0050 OPEN FL1,'E80',2'INVTY',OUT
```

FL1 is the file reference, which can be from FL0 to FL9, but must be the same as the file reference in the GET or PUT statement. 'E80' is the device address of the tape unit built into the 5110 Model 1. The number 2 specifies which physical file on tape is going to be used. This number can be specified as a variable. The word OUT indicates that the file is to be used for storing data items (with PUT statements) in the file for use in the program.

If you are opening a file on diskette, replace 'E80' in the sample statements with 'D80' (device address of diskette drive 1) and enter the name of the file on diskette. The file name is required when you create a file on diskette.

If a record-oriented file were to be created with WRITE FILE statements, it could be opened as an output file with this statement:

```
0100 OPEN FILE FL1,'D80',2,'INVTY',OUT,RECL=128
```

Note: See the *IBM 5110 BASIC Reference Manual*, SA21-9308, for a complete description of the OPEN statement.

Normally, a file is deactivated or *closed* by the system after execution of your program. However, if you want to switch an input file to output (or vice versa) and continue to use it in the same program, you must deactivate it by using the CLOSE [FILE] statement before reopening it. (If you did not use the CLOSE statement and attempted to use an output file for input or vice versa, execution of your program would be terminated.) The CLOSE statement deactivates the file; a subsequent OPEN statement opens (reactivates) the file for its new use and repositions it at its beginning. Under ordinary circumstances, the CLOSE statement is optional, and the system will close a file at the end of program execution.

The CLOSE statement is required, however, if you use the same file for *both* input and output operations in the same program. This does not apply to record-oriented files opened with the ALL parameter (see *OPEN FILE Statement* in the *IBM 5110 BASIC Reference Manual*, SA21-9308). The CLOSE statement is also required if several different files are accessed by the same program.

CREATING A TAPE OR DISKETTE FILE

The following compound interest program can be used to produce an output listing containing 600 values in 200 lines:

```
0100 PRINT 'ENTER PRINCIPAL'
0020 INPUT P
0030 PRINT FLP, 'TIME', 'RATE', 'AMOUNT'
0040 FOR T=1 TO 10
0050 FOR R=1 TO 20
0060 A=P*(1+R/100)↑T
0070 PRINT FLP, T,R,A
0080 NEXT R
0090 NEXT T
0100 STOP
```

The PRINT statement in this program is executed 200 times to produce an output listing containing the values. These values could be grouped as an output file on tape or diskette. In fact, instead of printing them, you could store them in the file and use them later. By adding an OPEN statement and substituting a PUT statement for the PRINT statement (line 0070), you can create a tape or diskette file; for example:

```
0025 OPEN FL1,'E80',2,OUT

0070 PUT FL1,T,R,A
```

This PUT statement instructs the system to put the values contained in the variables T, R, and A into the file that is defined in the OPEN statement with the same file reference (FL1). As far as the system is concerned, both PUT and PRINT mean output; the only difference is whether the output goes to a tape or diskette file, or to the printer or display screen. Semicolons cannot be used to separate variables in a PUT statement; use only commas.

RETRIEVING A FILE

To access data in a tape or diskette file, you use the GET statement, which is the input counterpart to the PUT statement. The file must first be opened, as shown below.

```
0050 OPEN FL1,'E80',2,IN
```

To access the first set of values from the file created with the preceding PUT statement (statement 0070, above), you can use the following GET statement. The file reference (FL1) must be the same as the file reference in the OPEN statement that defines the specific file.

```
0070 GET FL1,T,R,A
```

This statement assigns the first three values contained in the file to the variables T, R, and A. It is not necessary to use the same variable names that were used when the file was created; for example, we could assign these values to variables X, Y, and Z. The important requirement is that the values in the file and the variables to which they are assigned must be the same type–arithmetic variables for arithmetic values, and character variables for character values.

After the first GET statement is executed, the file is positioned at the next value. Thus, a second GET statement referring to FL1 would access the next three values in the file. If we wanted to access all the values stored previously, we could issue the GET statement 200 times, or enclose one GET statement in a loop as follows:

```
0050 OPEN FL1,'E80',2,IN
0060 FOR X=1 TO 200
0070 GET FL1,T,R,A
0080 PRINT T,R,A
0090 NEXT X
```

These statements would print the 200 values for each T, R, and A.

REPOSITIONING FILES

You may have an occasion to use an input file or an output file more than once in the same program. The RESET statement allows you to reposition the file without deactivating it (deactivation is necessary only when the function of a file is changed from input to output or vice versa). For example:

```
0020 OPEN FL4,'E80',4,IN
0030 GET FL4,X,Y,Z,Q,R,S
   .
   .
   .
0100 RESET FL4
0110 GET FL4,X,Y,Z,Q,R,S
   .
   .
   .
0150 RESET FL4
0160 GET FL4,X,Y,Z,Q,R,S
   .
   .
   .
```

Between statements 0030 and 0100, the variables X, Y, Z, Q, R, and S could be used in one set of calculations and their values changed. By repositioning the file, the original values in the file could again be made available and put into variables X, Y, Z, Q, R, and S for different calculations or uses between statements 0110 and 0150, and again between statement 0160 and the end of the program. Actually, the RESET statement used in this way functions for files in the same way that the RESTORE statement functions for the data table created by the DATA statement.

To add data to the end of the file, you can reset it to its end by using the RESET statement with the END keyword:

```
0200 RESET FL1 END
```

This statement positions FL1 to the end of the last data item in the file. PUT statements appearing after statement 0200 will place additional values in the file. In effect, RESET END allows you to build onto a file. (See *RESET Statement* in the *IBM 5110 BASIC Reference Manual*, SA21-9308.)

Chapter 10. Arrays

With the BASIC language, you can keep groups of similar data (arithmetic or character) together by organizing them into arrays. An array is a collection of data items that is referred to by a single name. Exercises for this chapter are provided in Chapter 13.

Arithmetic arrays are named by a single letter of the extended alphabet. Thus, the letter A can stand for a single arithmetic variable or an arithmetic array or both, while the symbol A2 can only stand for a single arithmetic variable. A single letter stands for an array when it has been defined in a DIM (dimension) statement, which is described later. All elements of an arithmetic array are initially set to 0 when the program is executed.

Character arrays, like simple character variables, are named by a single letter of the extended alphabet followed by a dollar sign ($). Each element of a character array can be up to 255 characters in length. Each element is initially set to blank characters when program execution begins.

BASIC arrays can be either one or two dimensions. A one-dimensional array can be thought of as a row of successive data items. A two-dimensional array can be thought of as a rectangular matrix of rows and columns. A representation of a one-dimensional array A containing four elements is:

Array A

A(1)	A(2)	A(3)	A(4)

A representation of a two-dimensional array B with four rows and three columns is:

Array B

B(1,1)	B(1,2)	B(1,3)
B(2,1)	B(2,2)	B(2,3)
B(3,1)	B(3,2)	B(3,3)
B(4,1)	B(4,2)	B(4,3)

To illustrate the use of one- and two-dimensional arrays, suppose you are keeping weather statistics on the average temperature and the inches of rainfall for 12 months. You can write a program to keep each set of that data in arrays:

- Names of the months
- Average temperature for each month
- Total rainfall for each month

You can arrange the data as three one-dimensional arrays:

Array 1 Names of Months	Array 2 Average Temperature	Array 3 Rainfall
January	28	3.47
February	31	2.10
March	35	2.95
April	49	4.82
May	60	3.02
June	64	2.87
July	75	2.04
August	81	1.89
September	71	2.74
October	59	2.90
November	46	1.85
December	37	2.35

Or as one two-dimensional array:

Array 1

Month	Temp	Rainfall
1	28	3.47
2	31	2.10
3	35	2.95
4	49	4.82
5	60	3.02
6	64	2.87
7	75	2.04
8	81	1.89
9	71	2.74
10	59	2.90
11	46	1.85
12	37	2.35

The second example is really a modified combination of the three one-dimensional arrays. The first column has been changed to the numeric representation of the months because the names of the months (character data) cannot be included in the same array with numeric data.

It will be much easier to use the weather data if we keep it together in three one-dimensional arrays, or in one two-dimensional array, than it would be if we considered it as 36 separate variables. This chapter will show you how to work with arrays in BASIC programs.

DEFINING AN ARRAY

When you want to work with an array, you must first tell the system that you are using an array and not ordinary variables. This is called *defining* your array. Defining the array merely involves telling the system how big the array is going to be so the system can leave room for it, and telling the system what kind of data will be in it. (Later on you enter the data, but this is not part of defining the array.)

The data for your arrays can be numeric data or character data. You can define an array to contain either kind of data, but it must contain *only* one kind of data. You can't mix characters and numbers in a single array. That's why we used the numbers of the months instead of their names when we put the weather data in a two-dimensional array.

An array composed of numbers is called an *arithmetic array*. It is named by a single letter of the extended alphabet such as A or T.

An array composed of character data is called a *character array*. It is named by a single letter of the extended alphabet followed by a dollar sign ($); for example, N$ or Q$.

To define either kind of array, you use a statement called DIM. In the DIM statement, you name the array and include the size of it in parentheses after the name.

DIM Statement for One-Dimensional Arrays

For a one-dimensional array, the size is a single number. Thus, to define an arithmetic one-dimensional array A with 12 elements, your DIM statement is:

```
0010 DIM A(12)
```

To define character array N$ with 20 elements, your statement is:

```
0010 DIM N$(20)
```

To define both together, your statement is:

```
0010 DIM A(12), N$(20)
```

DIM Statement for Two-Dimensional Arrays

For two-dimensional arrays, the size is two numbers, one for each dimension. The first number is the number of rows in the array; the second number is the number of columns in the array.

To define array W with 12 rows and 3 columns, the DIM statement is:

```
0010 DIM W(12,3)
```

Character array A$ with 3 rows and 4 columns is defined by:

```
0010 DIM A$(3,4)
```

You can define all your arrays in a single DIM statement. You can also mix definitions of one- and two-dimensional arrays in a single DIM statement.

DIM Statement for Character Variables

The DIM statement is also used to define the length of character variables to be referenced in a program (such as by a PRINT USING statement, as previously described). The DIM statement to define a character variable has the following format:

```
0010 DIM A$49,B$8(10)
```

In this example, character variable A$ has been assigned a length of 49 characters, and character array B$ has been assigned 10 elements of 8 characters each. Once you have defined a length for a character variable, you can assign a value of that length. If you do not define a length in a DIM statement, the character variable has an assumed (default) length of 18 characters.

ELEMENTS OF ARRAYS

Each individual item in an array is called an *element* of the array. When you want to refer to a particular element of an array, instead of the whole array, you talk about the position of that element in the array. For example, if you want to refer to the third element of one-dimensional array H, you would refer to it as H(3). To refer to the element in the first row and third column of array W, you use W(1,3). The position goes in parentheses after the name of the array. For two-dimensional arrays, the first number is always the number of the row, the second number is always the number of the column.

If we look at the weather example as three one-dimensional arrays, we can call the array with the names of the months M$, the array of temperature data T, and the array of rainfall data R. If we consider the weather data as one two-dimensional array, called W, the numbers of the months are in column 1, the temperature data is in column 2, and the rainfall data is in column 3. If you wanted to refer to January in a program statement, you would refer to either M$(1) or W(1,1).

Here are all the months and the way you refer to them in arrays M$ and W:

	Is in this Position:	
This Month:	**In Array M$**	**In Array W:**
1 (January)	M$(1)	W(1,1)
2 (February)	M$(2)	W(2,1)
3 (March)	M$(3)	W(3,1)
4 (April)	M$(4)	W(4,1)
5 (May)	M$(5)	W(5,1)
6 (June)	M$(6)	W(6,1)
7 (July)	M$(7)	W(7,1)
8 (August)	M$(8)	W(8,1)
9 (September)	M$(9)	W(9,1)
10 (October)	M$(10)	W(10,1)
11 (November)	M$(11)	W(11,1)
12 (December)	M$(12)	W(12,1)

Note that the month names are not used in array W, although the corresponding month number is used.

If we include the temperature and rainfall data, the first element in each one-dimensional array–M$(1), T(1), R(1)–or the first row in array W–W(1,1), W(1,2), W(1,3)–will be data for January; the second element in each one-dimensional array, or the second row in W, will be data for February; and so on.

So far, however, there is no data in any of the arrays. We have only defined the names and sizes. After you define an array, the system sets the values of *all* its elements to 0 (for arithmetic arrays) or blanks (for character arrays).

Assigning Values to Array Elements

To assign values to array elements (the names of the months, the temperatures, or the rainfall), you use the methods of assigning values that you've been using all along.

LET Statements

You can use a LET statement to assign a value to an element of an array. So if the average temperature for January is 28°, you could write either of these statements:

```
0020 LET T(1)=28
0020 LET W(1,2)=28
```

This method is acceptable if you only have a few values to assign, but it is time-consuming if the array is large. In the weather example, we would need 36 separate LET statements to assign all the data to the arrays. Nevertheless, the LET statement is handy if you only want to assign a few values, or if you want to change a value you have already assigned.

Remember that if you are assigning a value to an element of a *character array* in an assignment statement, you enclose the characters you are assigning in single quotation marks. For example:

```
0020 LET M$(1)='JANUARY'
```

DATA and READ Statements

Another way to assign values is to use DATA and READ statements. You use these the same way you do for variables. For example:

```
0020 READ M$(1),M$(2),M$(3)
0030 DATA JANUARY,FEBRUARY,MARCH
```

or

```
0020 READ W(1,1),W(2,1),W(3,1)
0030 DATA 1,2,3
```

Again, when you are using large amounts of data, listing them all separately in a READ statement is not practical. In this example, you can take advantage of a FOR-NEXT loop to assign values:

```
0020 FOR I=1 TO 12
0030 READ T(I)
0040 NEXT I
0050 DATA 28,31,35,49,60,64,75,81,71,59,46,37
```

or

```
0020 FOR I=1 TO 12
0030 READ W(I,2)
0040 NEXT I
0050 DATA 28,31,35,49,60,64,75,81,71,59,46,37
```

These statements assign all the average temperature data to array T or to the second column of array W. (For array W, since we are assigning values *only* to the second column, we used a constant of 2 in the READ statement.) You can't avoid specifying 12 values in the DATA statement, but a loop like this makes the READ statement easier to handle.

When assigning values to array W, you could, in fact, use one READ statement and two loops to assign *all* the data at once. It would look like this:

```
0020 FOR I=1 TO 12
0030 FOR J=1 TO 3
0040 READ W(I,J)
0050 NEXT J
0060 NEXT I
```

Arranged this way, the loops let you enter the data for each row of the array in succession. Your DATA statements might look like this:

```
0070 DATA 1,28,3.47
0080 DATA 2,31,2.10
0090 DATA 3,35,2.95
```

We've entered the data for each row of array W in a separate DATA statement because it is easier to visualize the data that way. You could, however, string out the data so that more than one row appears in a DATA statement like this:

```
0070 DATA 1,28,3.47,2,31,2.10,3,35,2.95 . . .
```

This way you could enter as many data items in each DATA statement as will fit on a line. The important thing is that the data must appear in the same *order* as if you were entering it row by row.

INPUT Statements

You can use INPUT statements to assign values from the keyboard to array elements. You can list all the array element names in the INPUT statement, or you can write a FOR and NEXT loop–similar to the ones for READ–to specify the names of the elements that are to recieve values.

For example, you can assign values to the one-dimensional rainfall array R with this statement:

```
0020 INPUT R(1),R(2),R(3),R(4),R(5)
```

or with these statements:

```
0020 FOR I=1 TO 12
0030 INPUT R(I)
0040 NEXT I
```

You can assign the rainfall data to the third column of array W with these statements:

```
0020 FOR I=1 TO 12
0030 INPUT W(I,3)
0040 NEXT I
```

As with the READ statement, you can write a double loop for an INPUT statement so that you can supply all the data for array W at once. In all instances, the system flashes a question mark on the display screen when the system is ready for you to enter the data from the keyboard. However, if your INPUT statement is in a loop, the system flashes a question mark each time the loop is executed. This means you supply one item of data, wait for the next question mark, supply the next item of data, and so on. You will have to enter the data one item at a time, waiting for a question mark after each entry.

Another Way to Assign Values to Arrays

Instead of using a loop with a READ or INPUT statement to assign values, you can write a READ or INPUT statement such as:

```
0020 MAT READ M$
0030 MAT INPUT N
```

These statements tell the system to read in values for the *entire* array. The letters MAT stand for the word matrix.

This method of assigning values with a MAT READ statement has no effect on your DATA statements. Thus, to assign the temperature data to one-dimensional array T, you could write these statements:

```
0020 MAT READ T
0030 DATA 28,31,35,49,60,64,75,81,71,59,46,37
```

If you use a MAT READ W statement, you would have to enter the data for the entire array in DATA statements. You assign the data row by row with these statements:

```
0020 MAT READ W
0030 DATA 1,28,3.47
0040 DATA 2,31,2.10
0050 DATA 3,35,2.95
```

or with these statements:

```
0020 MAT READ W
0030 DATA 1,28,3.47,2,31,2.10,3,35,2.95
```

If you use a MAT INPUT statement to assign values to an array, the system will signal you with a flashing question mark, as usual, when it is ready for you to enter data from the keyboard. If you are supplying values for a one-dimensional array, just type in all the values on a single line. If you are supplying values for a two-dimensional array, type in all the data row by row. Remember that the values must be separated by commas.

Assigning Values to an Entire Array at Once

If you want every element of an array to have the same value, such as all 1's or all 0's, you can assign that value to each element of the array with the following statement:

```
0030 MAT A=(0)
```

You could also assign to every element of an array the value of a variable or the value of an arithmetic expression with this statement:

```
0050 MAT T=(X)
```

or this statement:

```
0060 MAT M=(X+Y*Z)
```

The value you are assigning must be enclosed in parentheses so that the system knows it is not the name of another array.

If you omit the parentheses, you can make one array an identical copy of another array by using this statement:

```
0070 MAT R=S
```

In this statement, you don't use parentheses because you are, in fact, referring to another array in this assignment statement.

This method of assigning values is limited, however. You can't use the following statement:

```
0040 MAT R=-S
```

to set the values of the elements of array R equal to the negative values of the elements of array S. To do that, you would have to write this statement:

```
0040 MAT R=(-1)*S
```

(See *Arithmetic with Arrays* later in this chapter for more information.)

Working with Elements of Arrays

After you assign values to elements of arrays, you can perform calculations with individual array elements. You use elements of arrays just as you use any variable in any BASIC statement. Nothing is different except that you are keeping a set of variables together for your own convenience in organizing data. Each element still has a value and can act as an independent variable.

Printing Arrays

Elements of arrays, like ordinary variables, can be used in any PRINT or PRINT FLP statement. Some examples of PRINT and PRINT USING statements that include array elements are:

```
0020 PRINT T(3),T(4),M$(2),W(10,2),X,Y,Z
0030 PRINT FLP,'THE AVERAGE RAINFALL FOR JANUARY IS:',W(1,3)
0080 PRINT USING 90 FLP,M$(3),R(3)
0090 :FOR THE MONTH OF ########## THE RAINFALL WAS #.##
```

In addition, you can print an entire array if you insert MAT before the PRINT statement. For example, the statement

```
0090 MAT PRINT FLP,T
```

will print the entire one-dimensional temperature array T. The statement

```
0060 MAT PRINT FLP,W
```

will print the entire two-dimensional weather array W. It will be printed row by row.

You cannot enter arrays and ordinary variables together in a MAT PRINT statement, although they can be intermixed in a simple PRINT statement if the array reference is preceded by MAT.

Putting One-Dimensional Arrays Together in a Program

Now we'll put the three one-dimensional weather arrays, M$, T, and R, together in a sample program that will keep all the data and display it when you run the program:

```
0010 REM THIS PROGRAM KEEPS WEATHER DATA
0020 DIM M$18(12),T(12),R(12)
0030 FOR I=1 TO 12
0040 READ M$(I),T(I),R(I)
0050 NEXT I
0060 DATA JANUARY ,28,3.47'
0070 DATA FEBRUARY ,31,2.1
0080 DATA MARCH ,35,2.95
0090 DATA APRIL ,49,4,4.82
0100 DATA MAY ,60,3.02
0110 DATA JUNE ,64,2.87
0120 DATA JULY ,75,2.04
0130 DATA AUGUST ,81,1.89
0140 DATA SEPTEMBER ,71,2.74
0150 DATA OCTOBER ,59,2.9
0160 DATA NOVEMBER ,46,1.85
0170 DATA DECEMBER ,37,2.35
0180 PRINT USING 0190,FLP
0190 :    MONTH            AVG TEMP              RAINFALL     FOR 1974
0200 FOR I=1 TO 12
0210 PRINT USING 0230,FLP,M$(I),T(I),R(I)
0220 NEXT I
0230 : ############        ###                   ##.#
0240 STOP
```

This program uses FOR and NEXT loops to simplify handling the large number of values involved in these arrays. Notice that instead of writing a FOR and NEXT loop for each array when we were assigning values to the members, we wrote a single loop that worked across the three arrays instead of completing each 12-element array individually. Of course, the DATA statements had to have their data in the same order.

We also used a loop to display the data. It lets us use a single PRINT USING statement with a single image statement to print out 12 lines of data.

After you enter the statements and run the program, the display screen shows:

```
   MONTH            AVG TEMP              RAINFALL      FOR 1974
JANUARY              28                    3.5
FEBRUARY             31                    2.1
MARCH                35                    3.0
APRIL                49                    4.8
MAY                  60                    3.0
JUNE                 64                    2.9
JULY                 75                    2.0
AUGUST               81                    1.9
SEPTEMBER            71                    2.7
OCTOBER              59                    2.9
NOVEMBER             46                    1.9
DECEMBER             37                    2.4
```

Two-Dimensional Array

Now we'll do the same thing with the two-dimensional array W. This time we'll use MAT READ W and MAT PRINT FLP, W statements instead of using loops to assign the weather data and print it. If your system does not have a printer, skip this program because the output exceeds the limits of the screen:

```
AUTO
0010 REM THIS PROGRAM KEEPS DATA IN A 2 DIM ARRAY
0020 DIM W(12,3)
0030 MAT READ W
0040 DATA 1,28,3.5,2,31,2.1,3,35,3.,4,49,4.8,5,60,3.
0050 DATA 6,64,2.9,7,75,2.,8,81,1.9,9,71,2.7,10,59,2.9
0060 DATA 11,46,1.9,12,37,2.4
0070 PRINT USING 0080,FLP
0080 : MONTH            AVG TEMP              RAINFALL    FOR 1974
0090 MAT PRINT FLP,W
0100 STOP
```

The printed output is:

MONTH	AVG TEMP	RAINFALL FOR 1974
1	28	3.5
2	31	2.1
3	35	3
4	49	4.8
5	60	3
6	64	2.9
7	75	2
8	81	1.9
9	71	2.7
10	59	2.9
11	46	1.9
12	37	2.4

ARITHMETIC WITH ARRAYS

Suppose, instead of weather data for one year, you have weather data for two years. This data can be in two arrays. You are interested in averaging the temperatures and rainfall over the two years and making new arrays to contain the two-year averages. To see how to do this, let's look at the two sets of temperature data. If you assume that they are in two one-dimensional arrays called A and B, then to find the average temperature for each month over the two years, you have to add the two temperatures for January and divide by 2, add the temperatures for February and divide by 2, and so on.

Addition and Subtraction with Arrays

You can do all the addition in one step, adding the entire array A to the entire array B, with this statement:

```
0010 MAT C=A+B
```

Again, the letters MAT stand for matrix. The preceding statement causes each element of array A to be added to the corresponding element of array B and the result to be stored in the corresponding element of array C.

The same kind of addition statement works if you want to add two-dimensional arrays. If all the weather data for the first year is in two-dimensional array T and for the second year in two-dimensional array U, and you want the result in array V, the statement is:

```
0040 MAT V=T+U
```

Each element of array T is added to the corresponding element of array U. This includes the columns with the numbers of the months and the columns with the rainfall.

Similarly, if you want to subtract each element of an array from the corresponding element of another array, you would write this statement:

```
0050 MAT C=A-B
```

The letters MAT always tell the system to work with an entire array. Just remember that you must define all the arrays, including the one which is receiving the results, in a DIM statement at the start of your program. Also, you can only add or subtract when all the arrays named have the same dimensions. You can't, for example, add a 14-element array to a 12-element array.

Multiplication and Division

We have seen how to add and subtract array elements. Now what about dividing by 2? Before we can divide, we must see how to multiply, because BASIC doesn't let you divide arrays directly; you can only multiply. You can multiply each element of an array (called A, for example) by a constant, a single variable, or an arithmetic expression with this statement:

```
0030 MAT C=(2)*A
```

The multiplier *always* goes in parentheses so the system knows it is not another array, and it must always go *before* the *. For division, you merely multiply the array by 1 over the divisor, or by a decimal number such as 0.5. Therefore, to divide each element of array A by 2, you would use this statement:

```
0080 C=(1/2)*A
```

Averaging Two Sets of One-Dimensional Arrays

If the weather data is kept in two sets of one-dimensional arrays, A and B for temperature and C and D for rainfall, a program for averaging the two sets of data and assigning the results to master arrays T and R might look like this:

```
0010 DIM M$18(12),A(12),B(12),C(12),D(12),T(12),R(12)
0020 MAT READ M$
0030 DATA  JAN , FEB , MAR , ARP , MAY , JUNE , JULY , AUG
0040 DATA  SEPT , OCT , NOV , DEC
0050 MAT READ A,B,C,D
0060 DATA 20,21,22,23,24,25,2,27,28,29,30,31
0070 DATA 10,12,14,16,18 20,22,24,28,30,31
0080 DATA 2,2,2,2,3,3,3,4,4,4,5,5
0090 DATA 5,5,5,4,4,4,2,2,2,1,3,2
0100 MAT T=A+B
0110 MAT T=(1/2)*T
0120 MAT R=C+D
0130 MAT R=(1/2)*R
0140 FOR I=1 TO 12
0150 PRINT FLP,M$(I),T(I),R(I)
0160 NEXT I
0170 STOP
```

We defined arrays T and R in the DIM statement on line 10, as well as arrays M$, A, B, C, and D. Note that we only need one array for the names of the months, no matter how many years of data we have stored in other arrays.

Averaging Two-Dimensional Arrays

If the two sets of weather data are stored in two-dimensional arrays X and Y, a program for averaging the data might look like this:

```
0010 DIM X(12,3),Y(12,3),W(12,3)
0020 MAT READ X,Y
0030 DATA
0040 DATA
0050 DATA
0060 DATA
0070 MAT W=X+Y
0080 MAT W=(1/2)*W
0090 MAT PRINT FLP:W
0010 END
```

Data for Arrays X and Y

We defined array W along with arrays X and Y in the DIM statement at the start of the program. Note that the numbers of the months, which are in column 1 of both arrays X and Y, are added in statement 0070 along with the rest of the data in arrays X and Y. But when we divide by 2 in statement 0080, we get back the original numbers 1 through 12.

Matrix Multiplication

To multiply a matrix by another matrix, first be sure that the two matrices have compatible dimensions: if A is a 4x5 matrix, then B must have 5 rows; for example, B can be a 5x7 matrix. This multiplication looks like:

```
30 MAT C=A*B
```

There are no parentheses because both A and B are matrices. Matrices A, B, and C should all be defined in a DIM statement before you use the multiplication statement. Matrix C must have dimensions equal to the number of rows in matrix A and the number of columns in matrix B.

Taking a Matrix Transpose

You can transpose a matrix (interchange its rows and columns) with a single statement, like this:

```
40 MAT B=TRN(A)
```

Matrix B assumes a value equal to the transpose of matrix A. When defining matrix B, remember that it must have dimensions opposite to those of A.

The Identity Matrix

You can make any square matrix (a 2x2 matrix, or a 12x12 matrix, for example) into an identity matrix (a matrix with ones on its diagonal and zeros everywhere else). You start by defining a square matrix, called I, for example:

```
10 DIM I(4,4)
```

Then you write this statement:

```
20 MAT I=IDN
```

This creates a matrix I that looks like this:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From then on in your program, you can use I to do calculations that require a 4x4 identity matrix.

Taking the Inverse of a Matrix

If you want to find the inverse of matrix A (the inverse is the matrix that gives you the identity matrix when you multiply it by A), you do it with a single program statement:

```
80 MAT B=INV(A)
```

Remember that for a matrix to have an inverse, it must be square (2x2, 12x12, etc.), and its determinant must not be equal to zero. You can check the determinant by using a built-in function called DET. DET(A) computes the determinant of matrix A. Before you use INV, then, you can test DET(A) with an IF statement like this:

```
65 IF DET(A)=0 GOTO 85
```

If you try to take the inverse when the determinant is zero, you will cause an error that will stop your program from running.

FINDING SQUARE ROOTS

You can determine square roots automatically with your 5110. Instead of writing your own formula for determining the square root of a number, you use the letters SQR, followed by the number whose square root you want to know enclosed in parentheses. For example, SQR (X) finds the square root of X, where X is 0 or a positive number.

You can use SQR in any of your arithmetic expressions, and the expressions inside the parentheses can involve any kind of arithmetic. For example:

$\sqrt{X+Y}$	is entered as	SQR (X+Y)
$\sqrt{\frac{X+Y+Z}{5}}$	is entered as	SQR ((X+Y+Z)/5)
$\sqrt{\frac{A+X}{2}}$	is entered as	SQR (A+X/2)

Other conversion and trigonometric functions and conversion constants in BASIC are discussed in this chapter.

SOME GENERAL SYSTEM FUNCTIONS

Your system provides you with the following general functions in addition to square root. To use these functions, simply substitute the name of your own variable for the character inside the parentheses. Exercises for this chapter are provided in Chapter 13.

ABS(X)	Gives the absolute value of X.
IDX(C_1,C_2)	Gives the location of the C_2 character string within the C_1 character string (relative to position 1). If C_2 does not appear in C_1, the value of IDX is zero.
LEN(C)	Gives the length of the C character string, less trailing blanks. If C is all blanks, the value of LEN is zero.
INT(X)	Gives the integer part of X.
RND or RND(X)	Generates a random number between 0 and 1.
SGN(X)	Determines the sign of variable X, and returns a value of -1, 0, or +1, depending on whether X is negative, zero, or positive.

You can also include expressions inside the parentheses, for example:

```
INT(X↑2+Y*12)
```

You might use SGN(X) to find out if X is positive:

```
SGN(X)
```

The RND function is a little different from the other functions. You can use RND alone, without a value, to generate a random number between 0 and 1. Each subsequent use of RND in the program will generate a new random number. However, if you rerun the program with a new RUN command, the random numbers generated will be the same as the numbers generated the first time you ran the program. To avoid this, you can use RND(X) to start different sets of random numbers each time you run your program. The value of X is used by the process that develops the random number. If you want a random number that is a whole integer instead of a decimal number between 0 and 1, multiply the result of RND or RND(X) by a constant (depending on what range you want the random numbers to have); and then use the INT function to make the result an integer.

For example:

INT(RND*10)	Produces a random integer between 0 and 9.
INT(RND*100)	Produces a random integer between 0 and 99.
INT(RND*1000)	Produces a random integer between 0 and 999.

ARITHMETIC CONSTANTS

There are special constants in BASIC that are used to convert pounds, inches, and gallons to metric kilograms, centimeters, and liters respectively. Normally, when you want to switch from the U.S. measuring system to the metric system, you multiply the measured quantity by a fixed constant to obtain the equivalent measurement in the metric system. For example, 1 lb equals 0.454 . . . kg, so 2 lbs equal 2*0.454 . . . kg. With BASIC, instead of remembering what the conversion multipliers are, the 5110 can provide them for you. These constants are:

- & INCM, which has a value of 2.54 (centimeters per inch)
- & LBKG, which has a value of 0.45359237 (kilograms per pound)
- & GALI, which has a value of 3.785411784 (liters per gallon)

In addition, BASIC has three built-in arithmetic constants to represent the values of:

1. e (natural log) = 2.718281828459045
2. π (pi) = 3.141592653589793
3. $\sqrt{2}$ (square root of 2) = 1.414213562373095

For example, if you want to use π in an equation, you don't need to type in 3.14 . . . . You just use the special BASIC constant. Here are the special symbols that BASIC recognizes for these constants:

For This Constant:	Use This Symbol:
e (natural log)	& E
π (pi)	& PI
$\sqrt{2}$ (square root of 2)	& SQR2

You might use one of these constants in a program that calculates the area of a circle (AREA = πR^2 is the formula). Your program statement would read:

```
50 LET A= & PI*R↑2
```

You can use these constants anywhere in your programs.

CONVERSION FUNCTIONS

BASIC has some built-in ways to convert values from one measuring system to another.

DEG(X)	Gives the number of degrees in X radians.
RAD(X)	Gives the number of radians in X degrees.
CHR(X)	Gives the equivalent character string of arithmetic expression X.
CEN(X)	Gives the centigrade (Celsius) equivalent of X degrees Fahrenheit.
FAH(X)	Gives the Fahrenheit equivalent of X degrees centigrade (Celsius).
NUM(C)	Gives the arithmetic value of character string C.

ARRAY/MATRIX FUNCTIONS

Several BASIC functions allow you to operate on all the elements of an array by a single reference to the array. These functions are as follows:

AIDX(A)	Gives the indexed locations of the elements in array A in ascending order.
DET(A)	Gives the determinant of array A.
DIDX(A)	Gives the indexed locations of the elements in array A in descending order.
(A)=IDN	Gives the identity matrix of array A.
INV(A)	Gives the inverse of matrix A.
PRD(A)	Gives the product of all the elements of array A.
SUM(A)	Gives the sum of the elements of array A.
TRN(A)	Gives the transpose matrix of array A.

RECORD FILE FUNCTIONS

BASIC provides three functions that allow you to inquire about record-oriented file status where C='FL0-FL9'. These functions are:

KLN(C)	Gives the length (in bytes) of the key field in file C.
KPS(C)	Gives the beginning position of the key field in file C.
RLN(C)	Gives the last record length for file C.

TRIGONOMETRIC FUNCTIONS

BASIC has functions that automatically perform trigonometric operations for you. Just substitute your own variable or expression where the variable X appears in the following list:

SIN(X)	Gives the sine of X radians.
COS(X)	Gives the cosine of X radians.
TAN(X)	Gives the tangent of X radians.
COT(X)	Gives the cotangent of X radians.
SEC(X)	Gives the secant of X radians.
CSC(X)	Gives the cosecant of X radians.
ASN(X)	Gives the arc sine (in radians) of X.
ACS(X)	Gives the arc cosine (in radians) of X.
ATN(X)	Gives the arc tangent (in radians) of X.
HCS(X)	Gives the hyperbolic cosine of the real number X.
HSN(X)	Gives the hyperbolic sine of the real number X.
HTN(X)	Gives the hyperbolic tangent of the real number X.

These functions deal in radians. If your program measures angles in degrees instead of radians, combine the RAD or DEG functions with these functions to keep the results in degrees. For example, to find the sine of D degrees, you can use this statement:

```
0040 LET S=SIN(RAD(D))
```

Or to find the arc sine of X in degrees instead of radians, you can use this statement:

```
0070 LET A=DEG(ASN(X))
```

LOGARITHMS AND EXPONENTS

BASIC also has functions that automatically take logarithms and calculate exponents for you:

EXP(X)	Gives the natural exponent of X (e^x).
LGT(X)	Gives the logarithm of X to the base 10.
LOG(X)	Gives the logarithm of X to the base e.
LTW(X)	Gives the logarithm of X to the base 2.

OTHER FUNCTIONS

MAX(X_1, . . .)	Gives the maximum value of the character or numeric scalars X_1, . . . (scalars must be all character or all numeric).
MIN(X_1, . . .)	Gives the minimum value of the character or numeric scalars X_1, . . . (scalars must be all character or numeric).

Chapter 12. If You Have Trouble

This chapter discusses some common problems and errors that you may encounter when you write your BASIC programs.

FORGETTING TO SAVE CORRECTED PROGRAMS

Problem: You've written a program and it doesn't work; so you go back and make corrections to it. The next day you run the program and you get the same incorrect results.

What To Do: You probably forgot to save the corrected version of the program. Correct the program again, then save the new version of the program. If you forget to save an original program, you must reenter the entire program.

ENDLESS LOOPS OR OUTPUT

Problem: Your program is executing a loop and you didn't include any way to end the loop in the program. (You may suspect something is wrong if you are waiting a long time for results and nothing seems to be happening.) Or you might be getting large amounts of output that you don't want. It doesn't have to go on forever.

What to Do: Press ATTN. Processing will stop immediately after execution of the current statement. If the problem is with a loop, list the program and insert a statement to end the loop when you've processed enough data. Enter GO or press EXECUTE to continue.

NUMBERS ARE NOT WHAT THEY SEEM TO BE

Problem: You are puzzled because you know a number should be a whole number, like 1, and the system prints out an answer as a decimal number, like .999999. Or else you test two numbers you think are equal and the system says they are not. Or you add a table of numbers across the rows and down the columns and then compare the grand totals you derive both ways and the totals do not match. Yet you can't find anything wrong with your program.

What Happened: This problem occurred because the system sees numbers in a different way than we do. Internally, it translates the numbers you enter into a specialized form that it can deal with, and, in the translation, the number is often changed slightly. It may be changed because the system rounds it off, or it may be changed because the system can only approximate some numbers that we think of as concrete and whole. Consider this: You are adding 1/3 + 1/3 + 1/3. You know the answer is 1. But if you had to use only decimal notation, you would have to add .33333 + .33333 + .33333 and your answer would be .99999 no matter how long you made the strings of 3s. The system suffers from this kind of limitation. It knows how to represent numbers in very few ways, so it can't change its viewpoint to suit the problem. There's nothing much you can do about this except learn to interpret the system answers as *approximate*. Fortunately, the approximation is generally extremely close to the *real* answer, and in many cases, you're not aware of any difference at all. Think of it as evidence of the human brain's superiority over the system; human beings can be flexible.

There are some programming techniques that can sometimes be used to circumvent this problem. If a PRINT statement seems to be giving you trouble, try substituting a PRINT USING and image combination to suppress extraneous digits or force an integer value when you might be getting an exponential value.

If an arithmetic expression is not yielding expected results, see if it combines both multiplication and division operations; the problem usually arises if division precedes multiplication in an expression evaluation. Try to multiply first and divide second (provided you can do so without changing the meaning of the expression) and you should get better results. For example, if we changed

```
0040 A = (I/100)*P
```

to

```
0040 A = P*I/100
```

in the interest example in Chapter 4 (see *Loops Within Loops*), we would get the integer results we would expect to get and not the odd decimal numbers that we had been getting.

HOW CAN A VAGUE IDEA BECOME A PROGRAM?

First decide on your ultimate goal, then simplify it. Pick a goal that's less ambitious and more realistic. Draw a picture, find the output lines generated by the computer; they become the PRINT statements in your program. Find the lines typed by the human; they become the INPUT statements. Write the PRINT and INPUT statements on paper, with a pencil, and leave blank lines between them. Fill in the blanks later by asking yourself how you'd get the answer if you didn't have a computer.

- Would you use a mathematical formula?

  Get it into your program, but remember that the left side of the equation must have just one variable and that BASIC executes according to the rule of hierarchy.

- Would you use a memorized list, such as English-French dictionary, the population of each state, or the weight of each chemical element?

  That list becomes your data and you need to read it. If you'll use the list more than once, you may restore or use subscripts.

- Would your reasoning repeat?

  If you know how often to repeat, say FOR NEXT. If you are not sure how often, say GO TO or IF THEN. If the repetition occurs only after several other activities have intervened, call the repeated part a subroutine, put it at the end of your program, and say GOSUB.

- Would you choose among several alternatives?

  Say IF THEN. To make the computer choose arbitrarily, say IF RND < .5 THEN . . . .

- Would you compare two things (A and B)?

  Say IF A=B THEN . . . .

Trim your program. Skim through it and eliminate any lines that are silly, such as a GOTO that goes to the next line, a GOTO that goes to STOP, a GOTO that goes to a GOTO, a THEN that goes to the next line, a THEN that just skips to a GOTO, or an IF followed by an IF that has the opposite condition.

Chapter 13. Exercises

This chapter contains exercises for you to study and enter with your system if you wish. The exercises are arranged by chapter, and give you practice with the statements that you learn in each chapter. Do only the exercises for the chapter you have just studied; exercises for more advanced chapters contain statements you haven't covered yet.

After you finish the book, you can look back at exercises from earlier chapters and see that with more advanced BASIC techniques, some of the earlier programs can be modified and made more efficient.

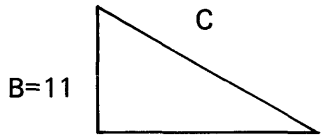
Feel free to make up examples of your own. Practice is the best way to become a proficient programmer. Have fun! If you need to make corrections to your own examples, refer to Chapter 6.

Note: Not all chapters are represented in this exercise section. This is because some chapters do not contain material for which exercises are relevant.

Exercises for Chapter 2. BASIC Arithmetic

Problem 1

Find the length of side C in this right triangle:



($C=\sqrt{A^2+B^2}$ is the formula)

The Program

```
0010 A=17
0020 B=11
0030 C=SQR(A↑2+B↑2)
0040 PRINT C
0050 STOP
```

Problem 2

Last season, Slugs Hoolihan, the baseball player, was up at bat 521 times. He made 130 base hits–15 home runs, 6 triples, 21 doubles and the rest singles. What is Slugs' batting average? What about his slugging average?

Note: The batting average is the number of hits divided by the number of times at bat; the slugging average is:

$$\frac{(\text{home runs}*4)+(\text{triples}*3)+(\text{doubles}*2)+(\text{singles}*1)}{\text{number of times at bat}}$$

The Program

```
0010 REM FIND AND PRINT HIS BATTING AVERAGE
0020 A=130/521
0030 PRINT A
0040 REM FIND AND PRINT THE SLUGGING AVERAGE
0050 S1=15*4+6*3+21*2+(130-15-6-21)
0060 S=S1/521
0070 PRINT S
0080 STOP
```

Problem 3

Here are the results of an election:

Candidate	District 1	District 2	District 3
Adams	4106	2890	2981
Biller	3729	3515	2493
Campbell	4002	3131	3012

Figure out the total vote and what percent of the total each candidate received.

The Program

```
0010 A$='ADAMS'
0020 A=4106+2809+2981
0030 B$='BILLER'
0040 B=3729+3515+2493
0050 C$='CAMPBELL'
0060 C=4002+3131+3012
0070 T=A+B+C
0080 A1=A/T
0090 B1=B/T
0100 C1=C/T
0110 REM PRINT TOTAL VOTE
0120 PRINT T
0130 PRINT A$,A,A1
0140 PRINT B$,B,B1
0150 PRINT C$,C,C1
```

Problem 4

You are shopping for an air conditioner. You need one between 5000 and 6000 BTUs. You want to find the one with the highest energy efficiency ratio (EER) because it will be the most economical to operate. The EER is defined as:

$$\frac{\text{number of BTUs}}{\text{number of watts}}$$

Here are the air conditioners you are considering:

A	5000 BTUs	820 watts
B	6000 BTUs	910 watts
C	5500 BTUs	850 watts

Which model should you buy?

The Program

```
0010 A=5000/820
0020 B=6000/910
0030 C=5500/850
0040 PRINT A,B,C
0050 STOP
```

Problem 5

You are renting a car to make a one-day trip of about 275 miles. Here are the rates for three companies:

Acme Rent-a-Car	$17 a day	17¢ a mile (includes gas)
Better Deals Rentals	$12 a day	22¢ a mile (includes gas)
Cheap Car	$8 a day	10¢ a mile (not including gas)

Figure the gas at 14 MPG and 58¢ a gallon. Which car is the best deal?

The Program

```
0010 A$='ACME'
0020 A=17+.17*275
0030 PRINT A$,A
0040 B$='BETTER DEALS'
0050 B=12+.22*25
0060 PRINT B$,B
0070 C$='CHEAP CAR'
0080 C=8+.1*275+(275/14)*.58
0090 PRINT C$,C
0100 STOP
```

Problem 6

Equations of this form:

$$AX^2+BX+C=0$$

(quadratic equations) can be solved for the two values of X with these formulas:

$$X=\frac{-B+\sqrt{B^2-4AC}}{2A} \qquad X=\frac{-B-\sqrt{B^2-4AC}}{2A}$$

Write a program for these formulas. Using it, solve this equation for X:

$$6X^2-29X+28=0$$

The Program

```
0010 A=6
0020 B=-29
0030 C=28
0040 X1=(-B+SQR(B↑2-4*A*C))/(2*A)
0050 X2=(-B-SQR(B↑2-4*A*C))/(2*A)
0060 PRINT X1,X2
0070 STOP
```

Try it for other equations by changing the values of A, B, and C. Note that this program will not work for some values of A, B, and C. Try it for $X^2+X+1=0$, or A=1, B=1, C=1. The computer will tell you why it doesn't work, if you don't already know.

Exercises for Chapter 4. How to Write a Program

Problem 1

Dr. Lemming sees an average of 60 patients a week in his office. Before he sees each patient, he washes his hands with a liquid antiseptic soap that he buys in gallon jugs. Each time he washes his hands, he uses a teaspoon of soap (there are 6 teaspoons in an ounce). How many weeks will the gallon last? If he also uses a half ounce of hand cream at the end of each day, seven days a week, to keep his hands from chapping from all the washing, how much hand cream does he use up with each gallon of soap?

The Program

```
0010 REM FIGURE THE TOTAL NUMBER OF WASHES IN A GALLON
0020 T=32*4*6
0030 REM DIVIDE BY 60 WASHES A WEEK TO FIND NUMBER OF WEEKS
0040 W=T/60
0050 PRINT W
0060 REM FIND HOW MUCH HAND CREAM NEEDED IN W WEEKS
0070 C=.5*7*W
0080 PRINT C
0090 STOP
```

Note: The calculation in line 20 is based on six teaspoons an ounce, 32 ounces a quart, 4 quarts a gallon. The calculation in line 70 is based on half an ounce a day, seven days a week, for *W* weeks.

Problem 2

Figure out the earned-run average for a baseball pitcher who pitched 187 innings and allowed 61 earned runs. An earned run average is calculated by dividing the number of earned runs by the number of innings pitched, and then multiplying the result by 9 (for 9 innings a game).

The Program

```
0010 REM SUPPLY THE DATA
0020 E=61
0030 I=187
0040 REM FIND THE AVERAGE AND PRINT IT
0050 E1=E/I
0060 A=E1*9
0070 PRINT A
0080 STOP
```

Try this program for a pitcher who pitched four innings and allowed eight runs; a pitcher who pitched 20-1/3 innings and allowed four runs (remember that 20-1/3 must be entered as 61/3); try your own numbers.

Problem 3

Find out how much sod Mr. Harris has to buy to cover his 70x28 foot lawn. If sod is 11.5¢ a square foot, how much will it cost?

The Program

```
0010 REM FIND THE NUMBER OF SQUARE FEET
0020 F=70*28
0030 REM FIND THE COST
0040 C=F*.115
0050 PRINT C,F
0060 STOP
```

Problem 4

You are supplying hamburgers for a party. There will be 17 adults and 14 children. Each hamburger will have three ounces of meat in it. Kids can eat two hamburgers each, adults three. How much meat do you have to buy?

The Program

```
0010 REM HOW MANY HAMBURGERS TOTAL?
0020 H=17*3+14*2
0030 REM HOW MANY OUNCES IS THAT?
0040 O=3*H
0050 REM HOW MANY POUNDS?
0060 P=O/16
0070 PRINT P,H
0080 STOP
```

Problem 5

You are arranging a charter flight. The plane costs $10,000 to hire. There are 117 seats on it. You want to add on a 5% surcharge for profit and a $2.50 per person airport tax. How much is each ticket?

```
0010 REM FIND THE BASE COST OF EACH SEAT
0020 S=10000/117
0030 REM ADD ON THE PROFIT AND THE TAX
0040 T=S+.05*S+2.5
0050 PRINT T
0060 STOP
```

Exercises for Branching (Chapter 4)

Problem 1

The Smiths and the Hoopers live next door to each other. They are both driving to Cape Cod, which is 350 miles away, for a vacation. The Smiths leave at 8 a.m. and average 48 miles an hour. The Hoopers leave at 10:30 a.m. and they drive at an average of 52 miles an hour. Who gets there first?

The Program

```
0010 S=350/48
0020 H=350/52
0030 IF S<H+2.5 GOTO 0090
0040 IF S=H+2.5 GOTO 0120
0050 REM HOOPERS GET THERE FIRST
0060 PRINT 'THE HOOPERS GET THERE FIRST'
0070 GOTO 0130
0080 REM THE SMITHS GET THERE FIRST
0090 PRINT 'THE SMITHS GET THERE FIRST'
0100 GOTO 0130
0110 REM THEY ARRIVE AT THE SAME TIME
0120 PRINT 'THE SMITHS AND HOOPERS ARRIVE TOGETHER'
0130 STOP
```

Problem 2

You are looking for an apartment. You find two, one on your own, and one through an agent. You like them equally, so your choice will be based strictly on cost. Here are the details:

Apartment 1: The rent is $225 a month and does not include utilities. You figure $15 a month for gas and electric. There is a two-year lease.

Apartment 2: The rent is $230 a month. It includes gas and electricity. You must pay the agent 10% of a year's rent. The lease is for two years.

Which should you take? What will be your average monthly cost over the two years?

The Program

```
0010 A1=24*(225+15)
0020 A2=24*230+.1*(12*230)
0030 IF A1≤A2 GOTO 0080
0040 REM APARTMENT 2 IS CHEAPER
0050 M=A2/24
0060 PRINT 'APARTMENT 2 CHEAPER, MONTHLY COST IS:'
0070 GOTO 0150
0080 M=A1/24
0090 IF A2=A1 GOTO 0140
0100 REM APARTMENT 1 IS CHEAPER
0110 PRINT APARTMENT 1 CHEAPER, MONTHLY COST IS:'
0120 GOTO 0150
0130 REM THEY COST THE SAME
0140 PRINT 'THEY BOTH HAVE A MONTHLY COST OF:'
0150 PRINT M
0160 STOP
```

Problem 3

Mr. and Mrs. Richards are figuring out their income tax and they are trying to decide whether to file joint or separate returns. Mr. Richards' taxable income is $8,750. Mrs. Richards' taxable income is $10,312.

For separate returns, this is the schedule:

taxable income $8,000-10,000, pay $1,630 + 28% of amount over $8,000

taxable income $10,000-12,000 pay $2,190 + 32% of amount over $10,000

For a joint return, this is the schedule:

taxable income $16,000-20,000, pay $3,260 + 28% of amount over $16,000

The Program

```
0010 REM FIGURE COST OF SEPARATE RETURNS
0020 I1=8750
0030 I2=10312
0040 T1=1630+.28*(I1-8000)
0050 T2=2190+.32*(I2-10000)
0060 T3=T1+T2
0070 REM FIGURE COST OF JOINT RETURN
0080 I3=I1+I2
0090 T4=3260+.28*(I3-16000)
0100 REM SEE WHICH IS CHEAPER
0110 IF T3<T4 GOTO 0140
0120 PRINT 'COST OF JOINT RETURN IS LESS'
0130 GOTO 0150
0140 PRINT 'COST OF SEPARATE RETURNS IS LESS'
0150 PRINT 'JOINT RETURN COSTS'
0160 PRINT T4
0170 PRINT 'SEPARATE RETURNS COST'
0180 PRINT T3
0190 STOP
```

Exercises for Loops (Chapter 4)

Problem 1

In 1960, the population of Alpha City was 753,580. Its growth rate was 3.7% a year. In 1960, the population of Betaville was 529,430. Its growth rate was 5.1% a year. If the growth rates keep steady, when can you expect the population of Betaville to surpass that of Alpha City?

The Program

```
0010 Y=0
0020 A=753580
0030 B=529430
0040 Y=Y+1
0050 A=A+.037*A
0060 B=B+.051*B
0070 IF B≤A GOTO 0040
0080 Y=Y+1960
0090 PRINT 'BETAVILLE WILL HAVE MORE PEOPLE THAN ALPHA CITY IN:'
0100 PRINT Y
0110 STOP
```

Note: Line 10 assigns an initial value of zero to variable Y. As we mentioned in the text, this line is not really necessary; the system gives an initial value of zero to *all* numeric variables before it begins to execute any BASIC program.

Problem 2

If you make a sum of integers like this:

1+2+3+4+5+6+7+8 . . .

how many numbers can you add before you exceed 1000?

The Program

```
0010 N=N+1
0020 S=S+N
0030 IF S<1000 GOTO 0010
0040 PRINT 'YOU EXCEED 1000 WHEN YOU TRY TO ADD THE NUMBER:'
0050 PRINT N
0060 STOP
```

Problem 3

Make a sales-tax reference chart for yourself. The sales tax in your state is 7%, and you want your chart to go from $0 to $10 in steps of 10¢.

The Program

```
0010 FOR X=0 TO 10 STEP .1
0020 PRINT X,.07*X
0030 NEXT X
0040 STOP
```

Problem 4

A bank teller decides to do an experiment to liven up his day. Every time a customer asks for change of a dollar, he makes change with a different combination of coins. Using half dollars, quarters, dimes, nickels, and pennies, how many customers can he give change to before he has to repeat a combination? Print out the combination and the total number of combinations.

The Program

```
0010 C=0
0020 PRINT 'HALVES','QUARTERS','DIMES','NICKELS','PENNIES'
0030 FOR P=0 TO 100 STEP 5
0040 FOR N=0 TO 20
0050 FOR D=0 TO 10
0060 FOR Q=0 TO 4
0070 FOR H=0 TO 2
0080 S=H*50+Q*25+D*10+N*5+P
0090 IF S≠100 GOTO 120
0100 C=C+1
0110 PRINT H,Q,D,N,P
0120 NEXT H
0130 NEXT Q
0140 NEXT D
0150 NEXT N
0160 NEXT P
0170 PRINT 'THE TOTAL NUMBER OF COMBINATIONS IS:'
0180 PRINT C
0190 STOP
```

Because of the way the loops are nested in this program, it is time-consuming to execute. A better and faster method is shown in the following program.

```
0010 PRINT ' H       Q       D       N       P'
0020 FOR H=0 TO 2
0030 FOR Q=0 TO 4-2*H
0040 FOR D=0 TO 10-5*H-2.5*Q
0050 FOR N=0 TO 20-10*H-5*Q-2*D
0060 P=100-50*H-25*Q-10*D-5*N
0070 C=C+1
0080 PRINT FLP,H;Q;D;N;P
0090 NEXT N
0100 NEXT D
0110 NEXT Q
0120 NEXT H
0130 PRINT FLP,'TOTAL WAYS ='C
```

Exercises for Chapter 5. Other Ways to Put Values into Programs

Problem 1

Make a general program for figuring out batting averages and slugging averages. When you run the program, you will have to enter the ballplayer's name (N$), the number of times he was up to bat (B), the number of home runs (R), triples (T), doubles (D), and singles (S). The program will compute the averages and will print the results.

The Program

```
0010 IPUT N$
0020 IPUT B,R,T,D,S
0030 H=R+T+D+S
0040 A=H/B
0050 PRINT 'THESE ARE THE STATISTICS FOR:'
0060 PRINT N$
0070 PRINT 'TOTAL BATTING AVERAGE IS:'
0080 PRINT A
0090 A2=(4*R+3*T+2*D+S)/B
0100 PRINT 'SLUGGING AVERAGE IS:'
0110 PRINT A2
0120 STOP
```

Problem 2

You are giving typing tests to applicants for typing jobs. The test is five minutes long. After counting the number of words the applicant typed in five minutes, you subtract 10 words for each error, and then you divide the result by 5. The minimum passing score is 50 words per minute, but if the score is between 45 and 50, you will let the applicant take the test one more time.

Write a general program that will let you type in the applicant's name, the total words, and the number of errors. The program should then compute the score and tell you if the applicant passed or not, or whether the applicant should be allowed to take the test again.

The Program

```
0010 PRINT 'ENTER NAME, NUMBER OF WORDS, NUMBER OF MISTAKES'
0020 INPUT N$,W,M
0030 S=(W-10*M)/5
0040 PRINT 'TEST RESULTS FOR:'
0050 PRINT N$
0060 PRINT 'WORDS PER MINUTE TYPED'
0070 PRINT S
0080 IF S<45 GOTO 0120
0090 IF S<50 GOTO 0140
0100 PRINT 'APPLICANT PASSED TEST'
0110 GOTO 0210
0120 PRINT 'APPLICANT DID NOT PASS TEST'
0130 GOTO 0210
0140 PRINT 'HAS APPLICANT ALREADY TAKEN TEST? ENTER YES OR NO'
0150 INPUT A$
0160 IF A$='YES' GOTO 0190
0170 PRINT 'LET APPLICANT TRY AGAIN'
0180 GOTO 0210
0190 PRINT 'APPLICANT HAS HAD TWO(2) TRIES'
0200 GOTO 0120
0210 STOP
```

Notes:

1. In line 150, we asked for a word instead of a number as input. In line 160, we tested the word. Character variables are just as valid as numeric variables in INPUT statements and IF statements. Just remember to name a character variable with a letter followed by a dollar sign, and to use quotes when you are referring to the *value* of the character variable.
2. Notice the way the testing in lines 80 and 90 is done. If a score is below 45, we get rid of it right away (line 80). The next test (line 90) only has to test if the score is below 50 to find scores between 45 and 50, since everything below 45 has already been eliminated. After the two tests, we know that any scores surviving to line 100 are above 50, so we don't have to test them again.

Exercise for Chapter 8. Setting Up Your Own Format–PRINT USING and Image Statements

You are a teacher and you want to write a program that will figure out and print a report of grades for each student at the end of the term. There have been three exams during the term, and a final exam. The grade for the term will be a letter grade based on an average of the exam scores according to this table:

90 and over	A	
80–below 90	B	(passing)
70–below 80	C	
below 70	D	(not passing)

In computing the average, the final exam has twice as much weight as any of the other three exams. The printed report is meant for the individual student, not you or the office, so it should be intelligible to him, and it should be personalized.

The Program

```
0010 PRINT 'HAVE ALL STUDENTS BEEN PROCESSED? ENTER YES OR NO'
0020 INPUT A$
0030 IF A$='YES' GOTO 310
0040 PRINT 'ENTER NAME OF STUDENT'
0050 INPUT N$
0060 PRINT 'ENTER GRADES FOR EXAMS'
0070 INPUT E1,E2,E3,F
0080 A=(E1+E2+E3+2*F)/5
0090 IF A<70 GOTO 180
0100 IF A<80 GOTO 160
0110 IF A<90 GOTO 140
0120 G$='A'
0130 GOTO 190
0140 G$='B'
0150 GOTO 190
0160 G$='C'
0170 GOTO 190
0180 G$='D'
0190 PRINT FLP,'FINAL GRADES FOR :',N$
0200 PRINT FLP,'YOUR GRADE FOR THE FIRST EXAM WAS: ',E1
0210 PRINT FLP,'YOUR GRADE FOR THE SECOND EXAM WAS: ',E2
0220 PRINT FLP,'YOUR GRADE FOR THE THIRD EXAM WAS: ',E3
0230 PRINT FLP,'YOUR GRADE FOR THE FINAL WAS: ',F
0240 PRINT USING 250,FLP,A,G$
0250 :YOUR FINAL AVERAGE IS ###.## AND YOUR FINAL GRADE IS #
0260 IF G$='D' GOTO 290
0270 PRINT FLP,'CONGRATULATIONS ON PASSING THIS COURSE!'
0280 GOTO 300
0290 PRINT FLP,'YOU DID NOT PASS. MAKE AN APPOINTMENT TO SEE ME'
0300 GOTO 10
0310 STOP
```

Exercise for Chapter 10. Arrays

Here is the 1972 Internal Revenue schedule for single taxpayers. Write a program using this schedule so that a clerk can enter any amount of taxable income and have the tax figured automatically.

Schedule X–Single Taxpayers Not Qualifying for Rates in Schedule Y or Z

If the amount on line 55 of form 1040 is over:	*But not over:*	*pay:*	*plus:*	*of the amount over:*
$ -----	$ 500	$ 00	14%	$ 000
$ 500	$ 1000	$ 70	15%	$ 500
$ 1000	$ 1500	$ 145	16%	$ 1000
$ 1500	$ 2000	$ 225	17%	$ 1500
$ 2000	$ 4000	$ 310	19%	$ 2000
$ 4000	$ 6000	$ 690	21%	$ 4000
$ 6000	$ 8000	$ 1110	24%	$ 6000
$ 8000	$ 10,000	$ 1590	25%	$ 8000
$ 10,000	$ 12,000	$ 2090	27%	$ 10,000
$ 12,000	$ 14,000	$ 2630	29%	$ 12,000
$ 14,000	$ 16,000	$ 3210	31%	$ 14,000
$ 16,000	$ 18,000	$ 3830	34%	$ 16,000
$ 18,000	$ 20,000	$ 4510	36%	$ 18,000
$ 20,000	$ 22,000	$ 5230	38%	$ 20,000
$ 22,000	$ 26,000	$ 5990	40%	$ 22,000
$ 26,000	$ 32,000	$ 7590	45%	$ 26,000
$ 32,000	$ 38,000	$ 10,290	50%	$ 32,000
$ 38,000	$ 44,000	$ 13,290	55%	$ 38,000
$ 44,000	$ 50,000	$ 16,590	60%	$ 44,000
$ 50,000	$ 60,000	$ 20,190	62%	$ 50,000
$ 60,000	$ 70,000	$ 26,390	64%	$ 60,000
$ 70,000	$ 80,000	$ 32,790	66%	$ 70,000
$ 80,000	$ 90,000	$ 39,390	68%	$ 80,000
$ 90,000	$100,000	$ 46,190	69%	$ 90,000
$100,000	--------	$ 53,090	70%	$ 100,000
	A	B	C	D

To write this program, we are going to use four arrays:

A: Maximum amount of income in each category
B: Minimum tax for that bracket
C: Percent at which excess is taxed
D: *Amount over* column

We don't need to make an array for the minimum amount of income in each income bracket, because we will not use it; instead, we will test to see what bracket the income belongs to in the same way we tested the typing scores in problem 2 from Chapter 5. Also, column D is exactly the same as column A except each element is *bumped* by one place.

The columns of the schedule are labeled to show which column corresponds to which array.

The Program

```
0010 REM SET UP THE ARRAYS
0020 DIM A(25),B(25),C(25),D(25)
0030 MAT READ A
0040 DATA 500,1000,1500,2000,2500,3000,3500,4000,10000,12000,14000
0050 DATA 16000,18000,20000,22000,24000,26000,32000,38000,44000,50000
0060 DATA 60000,70000,80000,90000,100000,1500000
0070 MAT READ B
0080 DATA 0,70,145,225,310,395,485,575,670,2090,2630,3210,3830
0090 DATA 4510,5230,5990,7550,10490,13390,16390,20190,26390
0100 DATA 32790,39590,46190,53090
0110 MAT READ C
0120 DATA 14,15,16,17,19,21,24,25,27,29,31,34,36,38,40
0130 DATA 45,50,55,60,62,64,66,68,69,70
0140 D(1)=0
0150 FOR I=1 TO 24
0160 D(I+1)=A(I)
0170 NEXT I
0180 REM GET AMOUNT OF TAXABLE INCOME
0190 PRINT "ENTER AMOUNT OF TAXABLE INCOME; IF FINISHED,ENTER 0"
0200 INPUT X
0210 IF X=0 GOTO 330
0220 REM TEST X TO SEE WHICH RANGE IT BELONGS TO
0230 FOR I=1 TO 25
0240 IF X<A(I) GOTO 260
0250 NEXT I
0260 REM COMPUTE TAX HERE
0270 T=B(I)+(C(I)/100)*(X-D(I))
0280 PRINT USING 290,FLR(X),T
0290 :TAX ON $$###,### IS $$###,###.##
0300 PRINT USING 310,FLR(X)
0310 :TAX ON $$###,### IS ????.??
0320 GOTO 190
0330 STOP
```

Here are two square matrices:

$$\begin{bmatrix} 14 & -2 & 7 \\ 10 & 0 & -8 \\ 13 & 27 & 4 \end{bmatrix} \quad \begin{bmatrix} -1 & -3 & -5 \\ 6 & 24 & 38 \\ 5 & 2 & 9 \end{bmatrix}$$

Write a program that prints their sum, difference, product, and inverses (if the inverses exist).

The Program

```
0010 DIM A(3,3),B(3,3),C(3,3)
0020 MAT READ A,B
0030 DATA 14,-2,7,10,0,-8,13,27,4
0040 DATA -1,-3,-5,6,24,38,5,2,9
0050 PRINT FLP,'THESE ARE THE MATRICES FOR THE COMPUTATIONS'
0060 MAT PRINT A,B
0070 MAT C=A+B
0080 PRINT FLP,'THEIR SUM IS'
0090 MAT PRINT FLP,C
0100 MAT C=A-B
0110 PRINT FLP,'THEIR DIFFERENCE IS'
0120 MAT PRINT FLP,C
0130 MAT C=A*B
0140 PRINT FLP,'THEIR MATRIX PRODUCT IS'
0150 MAT PRINT FLP,C
0160 IF DET(A)=0 GOTO 250
0170 MAT C=INV(A)
0180 PRINT FLP,'THE INVERSE OF A IS'
0190 MAT PRINT FLP,C
0200 IF DET(B)=0 GOTO 270
0210 MAT C=INV(B)
0220 PRINT FLP,'THE INVERSE OF B IS'
0230 MAT PRINT FLP,C
0240 GOTO 280
0250 PRINT FLP,'THE DETERMINANT OF A IS 0, THERE IS NO INVERSE'
0260 GOTO 200
0270 PRINT FLP,'THE DETERMINANT OF B IS 0, THERE IS NO INVERSE'
0280 STOP
```

Exercises for Chapter 11. More Things You Can Do with BASIC

Problem 1

Make a chart of the values of sine and cosine of X, as X goes from 0° through 360° in 45° steps.

The Program

```
0010 PRINT 'DEGREES', 'SIN', 'COS'
0020 FOR X=0 TO 360 STEP 45
0030 Y=RAD(X)
0040 PRINT X,SIN(Y),COS(Y)
0050 NEXT X
0060 STOP
```

Problem 2

You are going to bake a cake. The recipe calls for an 8-inch square pan. You only have an 8-inch round pan, and a 9-inch round pan. Which pan will have a surface area closest to the required one?

The Program

```
0010 S8=8*8
0020 R8=&PI*(4↑2)
0030 R9=&PI*(4.5↑2)
0040 D1=S8-R8
0050 PRINT 'THE 8-INCH PAN IS',D1,'SQUARE INCHES SMALLER'
0060 D2=S8-R9
0070 IF D2<0 GOTO 100
0080 PRINT 'THE 9-INCH PAN IS','SQUARE INCHES SMALLER'
0090 GOTO 110
0100 PRINT 'THE 9-INCH PAN IS',-D2,'SQUARE INCHES BIGGER'
0110 D2=ABS(D2)
0120 IF D1<D2 GOTO 150
0130 PRINT 'THE 9-INCH PAN IS CLOSER TO THE REQUIRED SIZE'
0140 GOTO 160
0150 PRINT 'THE 8-INCH PAN IS CLOSER TO THE REQUIRED SIZE'
0160 STOP
```

Problem 3

Write a program where the system *thinks* of a number and you have to guess the number.

The Program

```
0010 PRINT 'ENTER ANY NUMBER'
0020 INPUT A
0030 PRINT 'GUESS THE NUMBER I THOUGHT OF IT,IS BETWEEN 1 & 100'
0040 PRINT 'YOU HAVE 8 TRIES'
0050 N=INT(RND(A)*100+1)
0060 IF T≥8 GOTO 0190
0070 T=T+1
0080 PRINT 'MAKE A GUESS'
0090 INPUT G
0100 IF G=N GOTO 0160
0110 IF G>N GOTO 0140
0120 PRINT 'MY NUMBER IS HIGHER'
0130 GOTO 0060
0140 PRINT 'MY NUMBER IS LOWER'
0150 GOTO 0060
0160 PRINT 'CONGRATULATIONS! YOU GUESSED IT'
0170 PRINT 'IT TOOK YOU ',T,' GUESSES'
0180 GOTO 0200
0190 PRINT 'SORRY, YOU HAD 8 GUESSES, MY NUMBER WAS',N
0200 PRINT 'DO YOU WANT TO PLAY AGAIN? 1=YES, 0=NO'
0210 INPUT B
0220 IF B=0 GOTO 0260
0230 A=N
0240 T=0
0250 GOTO 0030
0260 STOP
```

Note: In line 50, RND(A) generated a random number between 0 and 1. Multiplying it by 100 scaled the number between 0 and 99. Since we wanted the number to be between 1 and 100, we added 1 to the result. Then we applied the INT function to extract just the integer part of the number.

REMEMBER, BASIC IS A GOOD LANGUAGE TO EXPERIMENT WITH.

THE MORE YOU EXPERIMENT, THE MORE YOU LEARN.

Appendix A. BASIC Statements and Commands

A complete list of the statements and commands in the BASIC language that are used by your system is shown below. A brief description of each statement and command is included. Although all the statements and commands are not discussed in this manual, each is described in detail in the *IBM 5110 BASIC Reference Manual*, SA21-9308.

BASIC STATEMENTS

CHAIN	Ends a program, then loads and begins executing another program.
CLOSE	Deactivates open stream-oriented files.
CLOSE FILE	Deactivates open record-oriented files.
DATA	Creates an internal data table of values.
DEF	Defines a function to be used in the program.
DELETE FILE	Deletes a record from a key-accessed file.
DIM	Specifies the size (dimensions) of an array or character variable.
END	Ends a program.
EXIT	Transfers program control if errors occur during input/output operations.
FNEND	Ends a function defined in a DEF statement.
FOR	Begins a loop.
FORM	Specifies the format of records in a file created for access by individual record or printed output.

[MAT] GET	Assign values from a stream-oriented file to variables.
GOSUB	Transfers program control to the beginning of a subroutine.
GOTO	Transfers program control to a specific statement.
IF	Transfers program control depending on specific conditions.
Image	Specifies the format of printed or displayed data.
[MAT] INPUT	Assigns values from the keyboard to variables during program execution.
[LET]	Assigns values to variables.
MAT	Assigns values to all elements of an array.
NEXT	Ends a loop (see FOR).
ONERROR	Transfers program control when BASIC statement errors occur.
OPEN	Activates stream-oriented files for input or output.
OPEN FILE	Activates record-oriented files for input, output, or both.
PAUSE	Interrupts program execution.
[MAT] PRINT (FLP)	Displays or prints the values of specified variables, expressions, or constants.
[MAT] PRINT USING (FLP)	Displays or prints the values of specified variables, expressions, or constants in a format defined in an image statement or variable or FORM statement.

[MAT] PUT	Writes the values of specified variables into a tape or diskette stream-oriented file.
[MAT] READ	Assigns values from the internal data table (see DATA) to variables or array elements.
[MAT] READ FILE [USING]	Assigns a record from a record-oriented file to a variable.
REM	Inserts comments or remarks in a program.
RESET [FILE]	Repositions a tape or diskette file to its beginning, end, or specified record position.
RESTORE	Causes values in the internal data table (see DATA) to be assigned starting with the first value in the table.
RETURN	Ends a subroutine or user-defined function.
[MAT] REWRITE FILE [USING]	Rewrites a specified record in a record-oriented file.
STOP	Ends a program.
USE	Saves values of variables between two programs to be executed by a CHAIN statement.
[MAT] WRITE FILE [USING]	Adds a record to the end of a record-oriented file.

BASIC SYSTEM COMMANDS

ALERT	Alerts the operator from the procedure file (see PROC command).
AUTO	Automatically provides numbers for BASIC statements.
CSKIP	Skips within a procedure file on a specified condition.
GO	Resumes execution of a program that was halted.
LINK	Allows loading of customer support functions such as tape recovery and diskette/tape copy.
LIST	Displays or prints the BASIC program.
LOAD	Loads storage with data from tape or diskette or data from the keyboard. Also see *Function Keys* in the *IBM 5110 BASIC Reference Manual*, SA21-9308.
MARK	Prepares a tape cartridge or diskette for programs or data to be saved.
MERGE	Combines programs on tape or diskette with programs in storage or data on tape or diskette with data in storage.
PROC	Initiates the execution of a procedure file, which contains programs, commands, and/or data.
RD=	Specifies the number of digits at which rounding occurs for displayed or printed results.
RENUM	Renumbers the statements in storage.
REWIND	Rewinds the tape cartridge.
RUN	Executes a BASIC program.

SAVE	Saves the BASIC program or data on tape or diskette.
SKIP	Unconditionally skips records in a procedure file.
UTIL	Displays or prints a directory of the contents of a diskette or tape. Also allows files to be renamed, dropped, freed, and protected, and initiates the SORT function (feature).

EDITING FUNCTION

DEL	Deletes a statement or a group of statements from storage.
KEYx	Allows editing of key groups, where X = 0 to 9.

Appendix B. Customer Support Functions

The following customer support functions are available for execution on 5110 systems having storage of 16K to 64K bytes. Details of the customer support functions are available in the *IBM 5110 Customer Support Functions Reference Manual*, SA21-9311.

- Loader
- Diskette initialization
- Diskette-to-diskette copy
- Tape-to-diskette copy
- Diskette-to-tape copy
- Tape-to-tape copy
- Diskette compress
- Diskette recovery
- Tape data recovery
- Tape header recovery
- Diskette label display
- Sort diskette files into a desired sequence (optical feature)
- Compress diskette files to make unused space available

READER'S COMMENT FORM

Please use this form only to identify publication errors or request changes to publications. Technical questions about IBM systems, changes in IBM programming support, requests for additional publications, etc, should be directed to your IBM representative or to the IBM branch office nearest your location.

Error in publication (typographical, illustration, and so on). **No reply.**

Page Number *Error*

Inaccurate or misleading information in this publication. Please tell us about it by using this postage-paid form. We will correct or clarify the publication, or tell you why a change is not being made, provided you include your name and address.

Page Number *Comment*

Note: All comments and suggestions become the property of IBM.

Name _____

Address _____

_____

● No postage necessary if mailed in the U.S.A.

SA21-9306-0

Cut Along Line

Fold

Fold

FIRST CLASS
PERMIT NO. 40
ARMONK, N. Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
General Systems Division
Development Laboratory
Publications, Dept. 245
Rochester, Minnesota 55901

Fold

Fold

5110 BASIC Introduction Printed in U.S.A. SA21-9306-0

IBM®

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

READER'S COMMENT FORM

Please use this form only to identify publication errors or request changes to publications. Technical questions about IBM systems, changes in IBM programming support, requests for additional publications, etc, should be directed to your IBM representative or to the IBM branch office nearest your location.

Error in publication (typographical, illustration, and so on). **No reply.**

Page Number *Error*

Inaccurate or misleading information in this publication. Please tell us about it by using this postage-paid form. We will correct or clarify the publication, or tell you why a change is not being made, provided you include your name and address.

Page Number *Comment*

Note: All comments and suggestions become the property of IBM.

Name _____

Address _____

_____

● No postage necessary if mailed in the U.S.A.

SA21-9306-0

Cut Along Line

Fold

Fold

FIRST CLASS
PERMIT NO. 40
ARMONK, N. Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
General Systems Division
Development Laboratory
Publications, Dept. 245
Rochester, Minnesota 55901

Fold

Fold

IBM®

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

5110 BASIC Introduction Printed in U.S.

SA21-9306-0

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

IBM 5110 BASIC Introduction Printed in U.S.A. SA21-9306-0

SA21-9306-0