SEIKO

WRIST INFORMATION SYSTEM
UC-2000 SERIES

Controller
UC-2200

BASIC MANUAL

CONTENTS

▶ PART I BASIC Operating Manual

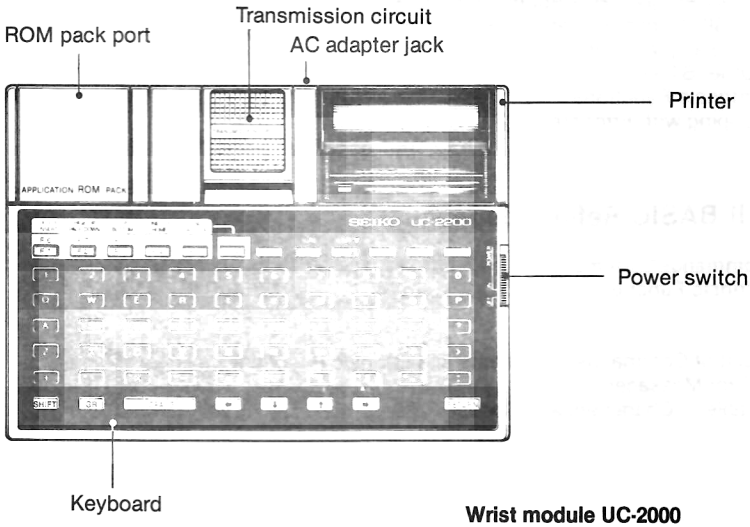
▶ PART II BASIC Reference Manual

PART 1 BASIC Operating Manual

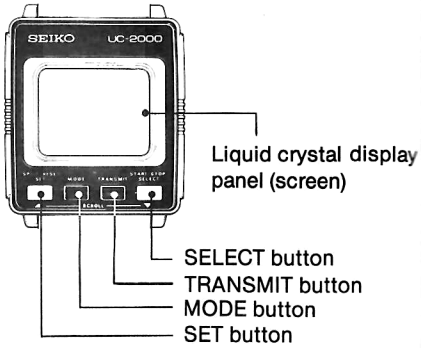
Chapter 1 Getting Started in BASIC

This manual describes UC-2200 BASIC, the programming language built into the Controller UC-2200. Before learning to use the proramming language, however, you need to know the procedure for starting BASIC up on the combination of the Wrist Module UC-2000 and Controller UC-2200.

Controller UC-2200

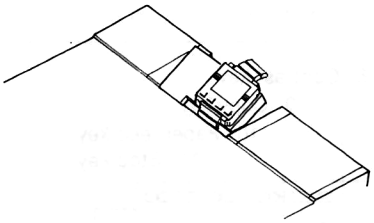


Wrist module UC-2000



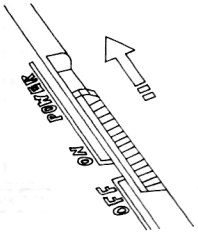
Press the TRANSMIT button ⇨

1. Press the TRANSMIT button on the watch. The words "TRANSMIT" STAND-BY will be displayed on the screen. Now the watch is ready to communicate with the controller.



2. Flick up the transmission circuit on the controller.

3. Place the watch on it as illustrated.



4. Turn the controller power switch ON.

5. Press the BASIC key on the controller

6. The following message will appear on the screen to tell you that the computer is ready to run under BASIC.

Now you can enter a BASIC program into the controller.

Memo 1

UC-2200 BASIC

The version of BASIC built into the SEIKO UC-2200 is an 8K BASIC written by Microsoft. It is an expanded version of BASIC 80, with some commands exclusive to UC-2200 BASIC. Most personal computers run versions of Microsoft BASIC, including BASIC 80.

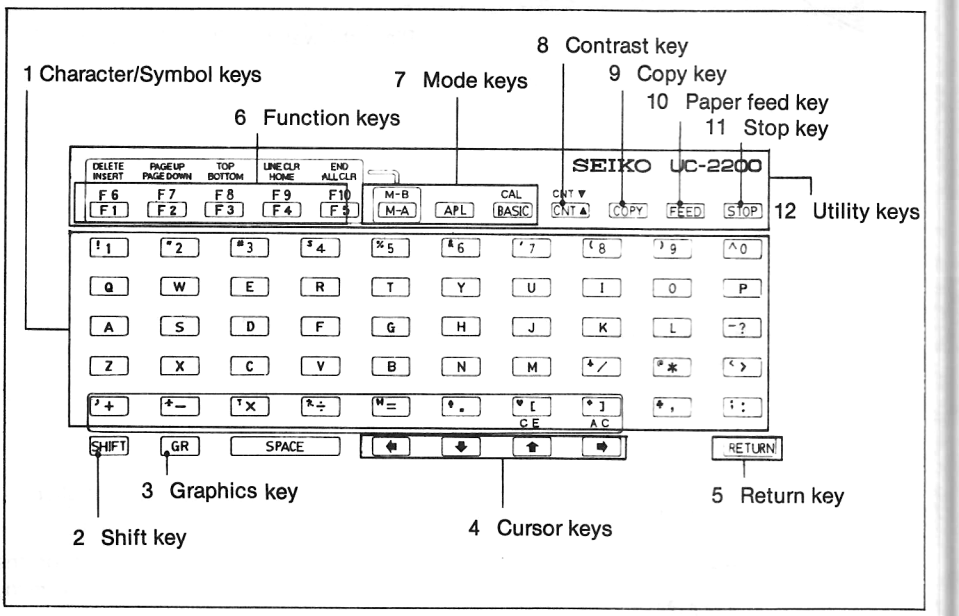
Since UC-2200 BASIC has a full set of essential commands and functions for ordinary programming applications, learning it is a good introduction to BASIC programming. And mastering UC-2200 BASIC also gives you mastery of BASIC 80, for they are nearly the same language. Learning UC-2200 BASIC will be an extremely important asset in advancing your personal computer career.

Chapter 2 The Keyboard

To operate the UC-2200, you need to have its keyboard functions at your fingertips. Thorough familiarity with the keyboard will help you get the most out of your computer.

2-1 The Keyboard Layout

Take a good look at your keyboard.



F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 key

On the top row of the keyboard is a set of keys labelled F1 to F10 . These are the function keys. Each function key has a variety of uses. These keys will help you to enter, correct, and run your programs efficiently. Chapter 3, "The Function Keys," explains their uses in detail. At this point, just learn where they are located on the keyboard.

→ ← ↑ ↓ keys

These keys are the cursor keys. Move the cursor, that little blinking rectangle [▮] on the screen, wherever you like.

Memo 2

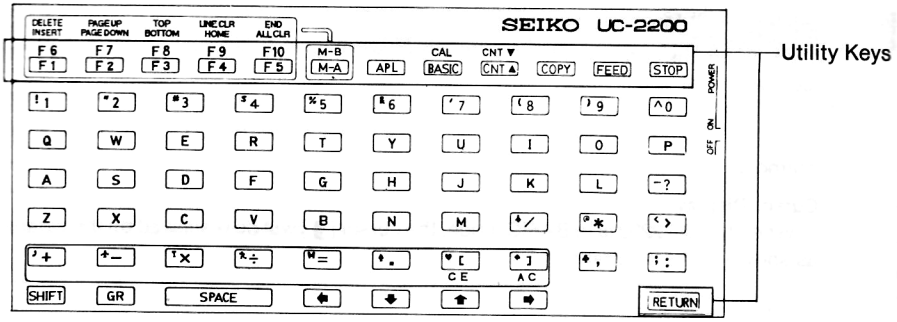
Cursor Display

When the UC-2200 is in BASIC mode, the cursor [▮] will be displayed on the screen as shown in the illustration.

When the cursor is on the screen, you can enter letters, numbers, or symbols from the keyboard. But when the cursor is not on the screen, the screen is locked; you cannot enter anything from the keyboard. When you enter a letter, number, or symbol, it will be displayed on the screen at the spot where the cursor was, and the cursor will move one space to the right.

Utility Keys

The row of keys to the right of the function keys includes the APL and BASIC keys. Also, the RETURN key is located in the lower righthand corner of the keyboard; it is essential to the preparation and execution of BASIC programs. These keys, plus the function keys, are called the utility keys. Be sure you know where they are located.



[F1] to [F10] ..... Function keys in each controller mode

[M-A] ..... Calls up Memo-A mode

[M-B] ..... Calls up Memo-B mode

[APL] ..... Calls up the Application mode

[BASIC] ..... Calls up the BASIC mode

[CNT▲] [▼CNT] ..... Adjusts the contrast on the screen, the liquid crystal display of the watch. CNT ▲ intensifies the contrast and CNT ▼ weakens it.

[COPY] ..... Turns on the printer. What is printed out depends on the mode selected.

[FEED] ..... Feeds printer paper forward

[STOP] ..... Stops computer processing

[RETURN] ..... Inputs displayed data or commands into the UC-2200 memory.

2-2 To key in characters

The keyboard is designed so that you can key in letters (upper or lower case), symbols such as + or −, and graphics characters such as " ┤ "
What is entered by pressing each key depends on whether you are using the keyboard in alphanumerics or graphics:

Alphanumeric keyboard: Upper- and lower-case letters; digits; and symbols
Graphics keyboard: Graphics characters and Greek letters

●Alphanumeric keyboard

The UC-2200 automatically starts in the alphanumeric keyboard when you enter the BASIC mode.
In the alphanumeric keyboard, you can key in upper- and lower-case letters as well as digits and symbols. Simply pressing a letter key keys in an upper-case letter. To produce a lower-case letter, press that key while holding down the SHIFT key.

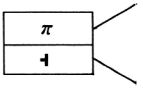
●Graphics keyboard

The GR key (graphics key) is just to the right of the SHIFT key.

(1) To key in a graphics character, press a letter, number, or symbol key while holding down the GR key.
(2) To key in Greek letters, press one of the symbol keys — +, −, =, ×or ÷— while holding down both the GR and the SHIFT keys.

Appendix 3, "Table of Characters and Codes," shows all the graphic characters that you can use with the graphics keyboard.

= key



a) Holding down both the GR and the SHIFT keys, press the = key. " π " will appear on the screen.
b) Holding down the GR key, press =. " ┤ " will appear on the screen.

Do you find the keyboard confusing? It is more complicated than an ordinary typewriter keyboard, but after all, the UC-2200 is a computer. At first it may be hard to remember where the keys are and what they do, but that is nothing to worry about. Just keep trying; you will soon be sued to it. And mastering this complicated-looking keyboard will be a key step in learning BASIC programming.

Chapter 3 The Function Keys

With the UC-2200 in BASIC mode, press any key and the corresponding character will appear on the screen. That gives you a visual check on the accuracy of your entry. The UC-2200 will also send you messages on the screen. You will be able to converse with the UC-2200 through the keyboard and screen. But to communicate with it easily, you need to learn more about the workings of the function keys in BASIC mode.

3-1 What the function keys do

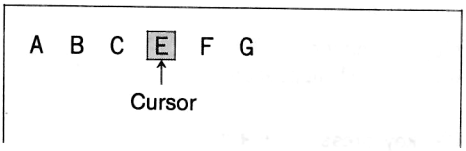
The function keys [F1] to [F10] have been assigned a variety of functions that help you enter and modify programs efficiently.

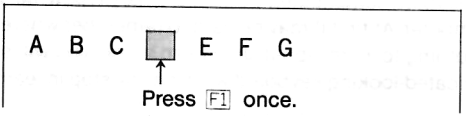
[F1] key (Function: INSERT)

Pressing the [F1] key inserts a space at the cursor position. The character which had been at the cursor position, or the string of characters of which it is part, is moved one space to the right.

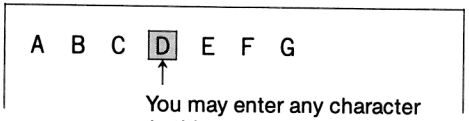
Example

Display on the screen





Press [F1] once.
A space will be inserted
between C and E.



You may enter any character
in this space.

F2 key (Function: EDIT)

Use the F2 key when editing or correcting a program. The F2 key is helpful when you want to renumber the lines of a program to fit a particular application or to correct a line after you have received an error message on the screen. With the F2 key, you can have the line you want displayed on the screen to correct syntactic errors or many any other changes you care to.

The F2 key saves you the trouble of keying in E D I T to effect an EDIT command.

Example

```
20  AB   <-- This statement should be A = B.
↑
Line number to be corrected
```

Key in F2, 2, and 0.

```
OK
EDIT  20
```

Press RETURN.

```
20  AB
```

Use the ↓ key to move the cursor to B.

```
20  AB
```

Press the F1 key to insert a space between A and B.

```
20  A B
```

Press the = key to insert an equals sign between A and B.

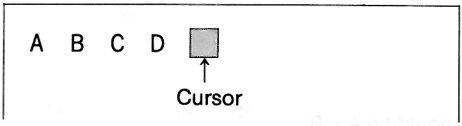
```
20  A= B
```

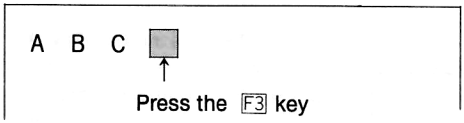
Press RETURN to make the UC-2200 store the revised line. That completes the correction procedure.

F3 key (Function: BACKSPACE)

The F3 key makes the cursor move to the left one space. Anything that had been entered in that space will be erased.

Example





Press the F3 key

The cursor moves one space to the left, erasing the D.

F4 key (Function: PAUSE)

When the UC-2200 is listing out a program on the screen line by line, the line you need to see may scroll past too quickly for you to check it. Pressing F4 will freeze the display so that you can study it at your leisure. When you are ready to go on, press any key. The PAUSE function can be used during a program list or other sequential displays.

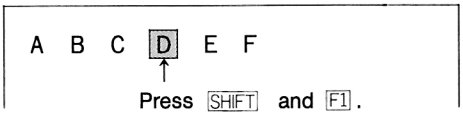
F5 key (Function: RUN + RETURN)

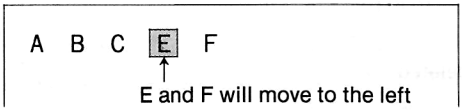
RUN is the command that executes a BASIC program. You can enter the RUN command by pressing the keys R, U, N, and RETURN . Or you can simply press F5, which enters the RUN command to execute your program.

F6 key (Function: DELETE)

Holding down SHIFT , press F1 to delete the character at the cursor location. The characters in that line to the right of the cursor will be moved a space left to close up the space.

Example

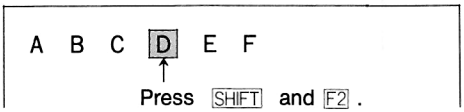


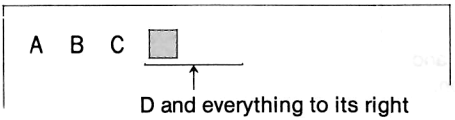


F7 key (Function: DELETE TO THE END OF THE LINE)

The F7 key is also used to delete characters. Holding down SHIFT , press F2 to erase everything from the cursor location to the right end of the logical line.

Example

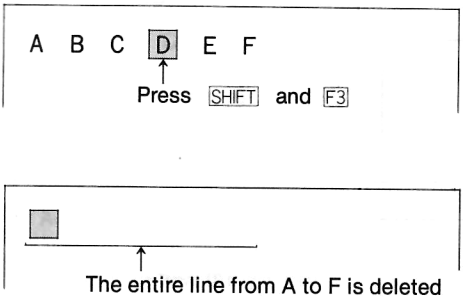




F8 key (Function: LINE DELETE)

The F4 key deletes whole logical lines. Holding down SHIFT, press F3, and the entire line in which the cursor is located will be erased.

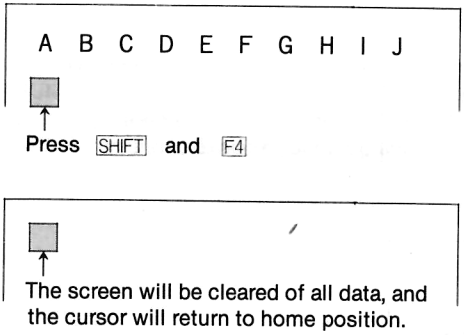
Example



F9 key (Function: CLEAR SCREEN)

Holding down SHIFT, press F4, and the screen will be cleared. The cursor will return to home position (the upper lefthand corner of the screen).

Example



F10 key (Function: END)

When you are finished using the UC-2200 in BASIC mode, hold down SHIFT and press F5. The computer will leave BASIC mode and can be shifted to memo or application mode.

3-2 Summary of Function Key Operation

KEY	KEY OPERATION	WHAT IT DOES
F1		INSERT: Inserts a blank space left of the cursor.
F2		EDIT: Displays the line you specify from the program in memory, for editing.
F3		BACKSPACE: Moves the cursor one space left, deleting anything entered there.
F4		PAUSE: Temporarily stops a program listing or other sequential display.
F5		RUN + RETURN : Executes a program just as if you had entered RUN and RETURN .
F6	Hold down SHIFT and press F1	DELETE: Deletes the character at the cursor location.
F7	Hold down SHIFT and press F2	DELETE TO THE END OF THE LINE: Deletes the character at the cursor position and everything to its right in the same logical line.
F8	Hold down SHIFT and press F3	DELETE LINE: Deletes the entire logical line marked by the cursor.
F9	Hold down SHIFT and press F4	CLEAR SCREEN: Clears all characters and symbols from the screen. The program in memory is not affected.
F10	Hold down SHIFT and press F5	END: To leave BASIC mode, press the STOP key, then hold down the SHIFT key and press F5 . The computer will leave BASIC mode.

Chapter 4 Introduction to Programming — First Steps

4-1 Programming skills come only with practice.

A computer can do only what it is told. When you want your UC-2200 to work for you, you have to give it an ordered set of instructions written in a language it can understand. That set of instructions is a program. The UC-2200 understands the programming language called BASIC. It can read, analyze, and carry out instructions in BASIC programs.

Computer programs are written by people. To write a program for your UC-2200, you need to learn a new language, BASIC. You must also learn to organize your instructions logically. Computers are literal-minded devices; they cannot infer what you really meant from what you said. Your UC-2200 will do what you tell it and nothing more.

Moreover, a human being can come up with several ways to carry out a given order. The computer cannot. You have to tell it every step in the procedure; those steps will be the lines of your program.

Elegant programming requires logic and experience, but do not be afraid to write something because it might be less than elegant. The computer does not care, after all. And each program you write will teach you more. Your first programs may be clumsy with superfluous parts compared with what an experienced programmer would do. But practice and only practice will make you an experienced programmer, too, with the knack of writing elegant programs.

Memo 3

MODES OF OPERATION

When you put your UC-2200 in BASIC mode, the screen displays the prompt OK. OK means the computer is ready to accept BASIC commands. At this point, BASIC may be used in either of two modes: direct or indirect. In the direct mode, BASIC statements and commands are not preceded by line numbers. They are executed as they are entered and the results are displayed immediately. This mode is convenient for using BASIC as a calculator for quick computations that do not require a complete program.

The indirect mode is the mode for entering programs. Program lines are preceded by line numbers, are stored in memory, and then are executed when you enter a RUN command.

4-2 Tips on programming

The suggestions given here will help you improve your programming skills. You have another valuable source of advice — the computer enthusiasts among your friends. They will be happy to give you the benefit of their greater experience. Each programmer has his own approach; your friends may suggest better solutions than we do in this manual.

● First define the purpose of your program.

What do you want the computer to do? That is the first question to ask yourself. Once you know what your goal is, start writing. But if your idea is still nebulous, wait until it has taken shape in your mind. If you start programming before you know where you are going, solving one programming problem after another will lead nowhere — except to giving up using the computer in frustration.

To get off to a good start, write down your idea — the task to be assigned to your computer. And concentrate on writing instructions for that task. Do not let clever ideas lead you astray.

●Take notes on the necessary procedures.

Suppose you want to write a program for determining the area of a rectangle. The following steps are necessary·

1 First, enter the length and width of the rectangle into the UC-2200·
2 Then calculate the area with the formula area equals width multiplied by length.
3 Finally display the result on the screen.

These steps are summarized in Figure A.

Figure A

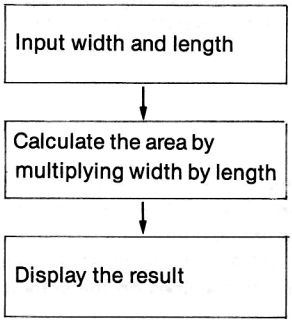
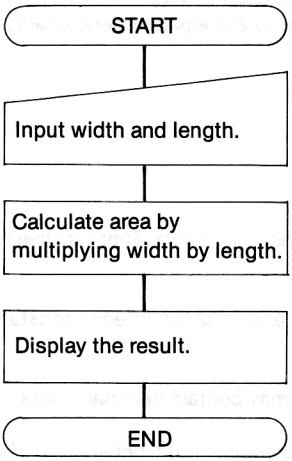


Figure A is fine as a reminder for yourself, but programmers often use a more convenient and systematic way of representing their ideas, called a flowchart. An example is shown in Figure B.

Figure B



Now that the procedure has been divided into elementary steps, let's see how to write it in BASIC. If you come to a step which you simply do not see how to write in BASIC, try reducing that step to simpler operations and writing them in BASIC.

Memo 4

To input a program

To input your program, first enter a line number. The UC-2200 runs each program in the order of the line numbers. Line numbers may be any positive integer from 0 to 65,529. Limited memory capacity makes 65,529 the maximum line number. It is usually convenient to number the lines at intervals of 10 (10, 20, 30, and so on). When you have input a line, for instance 10 INPUT T, be sure to press the RETURN key. Unless you press RETURN ., your instruction 10 INPUT T will not be stored in the UC-2200 memory.

A program line can be a maximum of 39 characters long. The program line is not one line on the screen. It is a logical line, defined as everything from the line number to the RETURN.

4-3 Programming

Now let's write a program for finding the area of a rectangle. After you enter each line of your program, be sure to press the RETURN key. The RETURN key marks the end of the line, telling the computer to store it and wait for the next line.

A line of program here means of logical line of BASIC code from the line number to the press of the RETURN key. It has nothing to do with the lines on the screen. A logical line is a maximum of 39 characters long.

Memo 5

CONSTANTS

Constants are the actual values BASIC uses during execution. There are two types of constants: string constants and numeric constants.

A string constant is a sequence of up to 255 alphanumeric characters or symbols enclosed in double quotation marks.

Examples:
"HELLO"
"$25,000.00"
"Seiko"

Numeric constants are positive or negative numbers. There are five types of numeric constants:

1. Integers
   Whole numbers between −32768 and +32767. Integer constants do not have decimal points.
2. Fixed point constants
   Positive or negative real numbers; may contain decimal points.
3. Floating point constants
   Positive or negative numbers in exponential form, between 10^{-38} and 10^{38}. A floating point constant consists of an optionally signed integer or fixed point number, the mantissa, followed by the letter E and an optionally signed integer, the exponent. Double precision floating point constants use the letter D instead of E.

Examples:
235.988E−7 = 0.0000235988
2359E6 = 2359000000

●Step 1: Input width and length

In BASIC variables can be thought of as boxes which store numbers and characters of strings of numbers and characters. Each variable (box) is given a name.

Here we shall define a variable, W, to store the width, and another variable, L, to store the length.

BASIC has command called INPUT that is used to ask you to enter a number or a character to be put in a variable's box. It can be used in the following way:

Memo 6

VARIABLE NAMES AND DECLARATION CHARACTERS

Variable names are formed of letters and numbers. A variable name must start with a letter. Though your variable names may be any length, the computer looks at only the first two letters or letter and number. Therefore, if variables named CONSTANT and COUNTER are used in the same program, the computer will treat them as the same variable.

A variable name may not be a reserved word. (Reserved words are all BASIC commands, statements, function names, and operator names.)

Variables may represent either a numeric value or a string. If the variable is a string, its name must end with a dollar sign. For example: A$ ="SEIKO".The dollar sign is a variable type declaration character; that is, it "declares" that the variable will be a string.

N$ . . . . . a string variable

ABC . . . . . a single precision numeric variable

Note: If a variable name begins with FN, it is assumed to be a call to a user-defined function.

INPUT <variable>

When the computer executes the command INPUT A, a question mark will appear on the screen, indicating that the computer is ready to accept input from the keyboard. That question mark is called a prompt.

If you then key in the number 3 and press RETURN , the number 3 will be assigned as the value for the variable A (or placed in the A box, if you like).

Let's use the INPUT command in the program to give the computer values for width and length.

```
10 INPUT W
20 INPUT L
```

●Step 2: Calculate area by multiplying width by length.

Now we must calculate the area from the values of width and length input. The area of a rectangle equals its width multiplied by its length.

BASIC includes mathematical functions, including arithmetic functions. The commands for the arithmetic functions are their symbols: addition +; subtraction −; multiplication *; and division /). It also can perform the logarithmetic function (LOG) and the exponential function (EXP). To calculate the product of width and length, we want to multiply. BASIC uses an asterisk (*) as the multiplication sign. A slash (/) is the division sign.

In BASIC, the area of a rectangle is calculated as follows:

```
W*L
```

But that expression does not tell us the result. For that we need another variable, which we shall call A for area. It is used as a box to store the result of the calculation.

```
30 A=W*L
```

This expression tells the computer to put the result of the calculation into the box for variable A.

Note: We often write mathematical expressions like this: B × C = D, with the formula first and then answer. In BASIC, however, always write expressions in this order: Answer = Formula. D = B*C. Here D is the variable (box) in which to store the result obtained with the formula.

Memo 7

OPERATORS

Operators perform mathematical or logical operations on values. The operators provided by UC-2200 BASIC may be divided into four categories:

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Functional operators

The arithmetic operators are given in the order in which the computer will carry them out:

Operator	Operation	Example
^	Exponentiation	x^y
-	Negation	-x
*, /	Multiplication, Floating point division	x * y x/y
+, −	Addition, Subtraction	x + y, x − y

Since the computer multiplies before it adds, its answer to 6*3 + 2 is 20.

To change the order in which the operations will be performed, use parentheses. Operations within parentheses are performed first. Inside the parentheses, the usual order of operations is maintained. Thus, 6*(3 + 2) = 30.

Here are some sample algebraic expressions and their BASIC counterparts.

Algebraic Expression	BASIC Expression
$X + 2Y$	X + Y*2
$X - \frac{Y}{2}$	X − Y/2
$\frac{XY}{Z}$	X * Y/Z
$\frac{X + Y}{Z}$	(X + Y)/Z
$(X^2)^Y$	(X ∧ 2) ∧Y
X^{YZ}	X∧ (Y * Z)
X (−Y)	X*(−Y)

●Step 3: Display the result.

In this step, the computer displays the area it has calculated on the screen. Since the result was stored inn variable A back in Step 2, all you have to do is ask the computer to display A. BASIC has another command, PRINT, which is used to display characters and variables on the screen.

The print command is used as follows:

PRINT <variable>
<number>
<character string or number>

Use the PRINT command to write the next line of our program.

```
40 PRINT A
```

●Summary

We have programmed steps one, two, and three in BASIC. Putting them together, we have the following program to determine the area of a rectangle:

```
10 INPUT W
20 INPUT L
30 A=W*L
40 PRINT A
50 END
```

The END statement in line 50 is the command which notifies the UC-2200 that the program has run to its end. The END statement may be omitted if it would be placed at the last line of the program anyway.

4-4 Running a Program

Now you know how to write a program for determining the area of a rectangle. Let's see if it really works.

Key in R U N RETURN **or simply press** F5 **.**

Do you understand what is happening? What's that question mark on the screen? That is the prompt; the UC-2200 is asking you what the length is. If you want to find the area of a rectangle 4cm. wide by 5cm. long, key in 5 and press RETURN . Here is another question mark. Now what does it want? The computer is asking you to input the width of the rectangle. Key in 4 and press RETURN . And the screen instantly displays the answer, 20. Now you know: the area of that rectangle is 20 square centimeters.

Did your program work this way? If your UC-2200 could not run your program, it gave you an error message such as ?SN ERROR on the screen, to help you locate your bug. BASIC regards as an error anything that makes the computer fail to complete running the program. The error messages let you know that there is a problem. In addition, BASIC tells you the number of the line containing the error. For details, see Appendix 2, Error Messages.

Chapter 5 Introduction to Programming — Striding Ahead

5-1 Every program has room for improvement

```
10 INPUT W
20 INPUT L
30 A=W*L
40 PRINT A
50 END
```

This is the program we wrote in Chapter 4 to determine the area of a rectangle. It is not a bad program, but it is too primitive to use easily. As it is, if you enter a negative figure for width or length, the computer will of course give a negative area as the result. But there are no negative areas in the real world; we should provide against such results. Worse, the program does not tell us what it is asking for when it displays those question marks as prompts on the screen. It would be a good idea to have the UC-2200 show what it is asking to be input, along with the prompt.

As you work with programs yourself, you will develop a sense of what a satisfactory program needs to do. BASIC will answer most, if not all, of your needs. As you become more familiar with BASIC, your programs will become more refined and easier to use.

For example, the program for calculating the area of a rectangle could be rewritten as follows:

```
100 REM DET. OF AREA
110 INPUT "WIDTH=";W
120 INPUT "LENGTH=";
L
130 AREA=W*L
140 AREA=ABS(AREA)
150 PRINT "AREA=";AR
EA
160 END
```

The INPUT statement and PRINT statement have been modified, and new commands and symbols (ABS, REM, AND;) are used. Actually the INPUT and PRINT commands are used in a different way from the earlier version.
A solitary question mark on the screen is not an adequate prompt since you cannot tell what the computer wants. To improve the situation, put a message — the question being asked — between quotation marks right after the INPUT command. The UC-2200 will display the message before the question mark when asking for input from you. The format for this INPUT statement is

INPUT "Message"; <variable>

The semicolon in this line means "followed by on the same line." The semicolon is used in the same way in line 150, the PRINT command.
The REM command on line 100 tells the computer to ignore everything in the rest of that line. This command allows you to insert a comment in the program, for instance to remind yourself what the next step does. When the UC-2200 comes to the line with the REM command, it skips it and proceeds to the next line. Explaining your program with messages on the screen and memos in REM statements will make it easier to understand and use.
The ABS command in line 140 tells the computer to find the absolute value of the variable AREA. Use ABS in this way:

Variable = ABS <variable>

Using this ABS command means the computer will never give you a negative result for area, even if you enter a negative value for width or length.
These new commands make your program more useful and go a long way towards harnessing what the UC-2200 can really do. Writing such programs will enrich your work and leisure, but only if you master BASIC. Don't worry, though — BASIC is the simplest computer language, and anyone can use it easily.

5-2 BASIC Commands at Your Fingertips

This section presents the most important BASIC commands with a generous selection of sample programs. Before long you will know how simply yet powerful BASIC commands are.

●Sample Program 1

The first sample program is given below.

```
10 CLS
20 CLEAR
30 FOR I=1 TO 100
40 A=A+I
50 NEXT I
60 PRINT A
```

This program calculates the sum of the integers from 1 to 100. That is, it adds up 1+2+3.....+100.

Line 10 contains CLS, the command to clear the screen.

CLEAR in line 20 is the command to clear all variables, setting all numerical variables to zero and all string variables to null. The UC-2200 has each variable permanently store the value given it until you give it a new value or deliberately clear the variable. If you leave the variables set at their previous values, those values may be used in your program and may make your program run incorrectly. Therefore, before running a program, you should empty all the variables with the CLEAR statement.

The FOR-TO statement in line 30 is always used in combination with the NEXT statement in line 50. We will call this pair the FOR-NEXT statement.

This program is intended to calculate the sum of the integers from 1 to 100. You could write out each addition as a separate step, but that would be tedious. The FOR-NEXT statement provides a very helpful shortcut for such tasks.

The statement FOR I = 1 TO 100 in line 30 tells the computer to increase the value of I by increments of one from 1 to 100. First the variable I is given the value 1 in line 30. Then comes the statement A = A + I in line 40. The variable A has an initial value of zero. When lines 30 and 40 have been executed once, A is set equal to 1 (that is, 0 + 1). Then the NEXT statement in line 50 sends the computer back to line 30 to get the next value of I. Since I was equal to one, adding one to it here (increasing by an increment of one) gives the new value of 2 for I this time. Then the program proceeds to line 40, where the new value of I is added to the old value of A to give a new value of 3 for A (A = 1 + 2). Next the computer executes line 50, which sends it back to line 30; the UC-2200 toils along this loop without complaint until I reaches 100. Then it leaves the loop and executes line 60.

The FOR-NEXT statement

30	FOR I = 1 TO 100	Starts with I = 1.	I = 2	I = 100
40	First operation	Carries out first operation once	Carries out first operation a second time	Carries out first operation the hundredth time
50	NEXT I	Returns to line 30	Returns to line 30	Leaves loop, goes to line 60
60	Second operation			At last, carries out second operation

The FOR-NEXT statement is used, as in this example, to make the computer repeat an operation. Every time I is incremented by one, the statement A = A + I is executed again. In the example, this process is repeated 100 times to calculate the sum of integers from 1 to 100. There are unexpected uses for FOR-NEXT statements. For instance, the statement can be used as a timer.

```
FOR J = I to 200
NEXT J
```

This loop puts the computer on a treadmill, repeating a series of dummy operations, to gain you some time. For example, it will take the computer about a second to work through a loop with I = 1 to 67. The dummy loop technique is useful when you need to keep something displayed on the screen briefly, then clear it off. Have the clear screen command come after the computer exits from the dummy loop.

FOR-NEXT statement can be combined with the STEP statement. Without the STEP statement, the increment is always 1. With the STEP statement, you can set the increment as you like.

In line 30 the statement FOR I = 1 to 100 had I increase in increments of one. But the statement FOR I = 1 TO 100 STEP 5 has I increase by increments of five up to 100. STEP 10 would make I increase at increments of 10. STEP 5 would make I decrease at increments of 5.

To use a negative increment such as STEP −5, however, you must write the FOR-TO statement so that the variable ranges from a high to a low value:

```
FOR I = 100 TO 0 STEP −5
```

If a NEXT statement refers to the FOR statement nearest it, the variable name can be omitted from the NEXT statement. In the example, 50 NEXT would run correctly.

●Sample Program 2

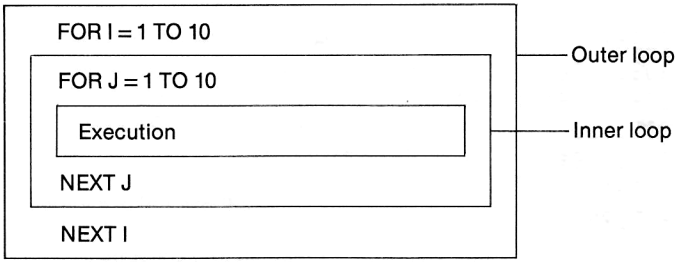
```
10 CLEAR
20 FOR I=1 TO 10
30 FOR J=1 TO 10
40 PRINT I;J
50 NEXT J
60 NEXT I
```

This sample program uses nested FOR-NEXT loops. That is, the program uses two FOR-NEXT statements, one occurring within the other's loop. This technique is called nesting loops. There is a rule governing nested loops. Each FOR-NEXT loop is a complete sequential cycle of operations. When two loops are used together, one loop must enfold the other completely. The paths of the two loops must not cross.

In the sample program, the computer executes the loop for variable J ten times before it executes the loop for variable I again. Line 40 will be executed 100 times in all. Reversing the order of lines 50 and 60 would violate the rule for nested loops, since the instructions for the J loop would not be contained within the I loop.

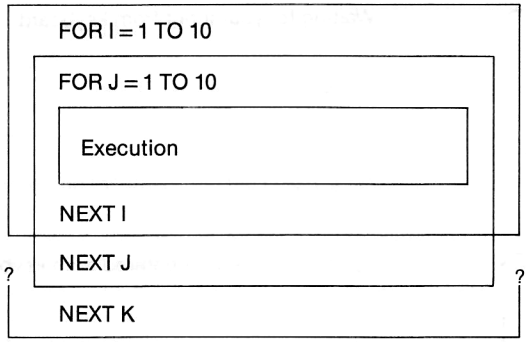
If two loops share the same end point, they could both end with the same NEXT statement.

CORRECTLY nested loops



*To nest two loops, one loop must be completely contained within the other.

INCORRECTLY nested loops



*The inner loop is not contained within the outer loop.
*If a NEXT statement has no corresponding FOR statement, the screen will display an NF ERROR message, and the program will come to a standstill.

●Sample Program 3

```
10 PRINT "A";
20 GOTO 10
```

Run this program and you will find the screen covered with As. The GOTO command consists of GOTO and a line number.

GOTO <line number>

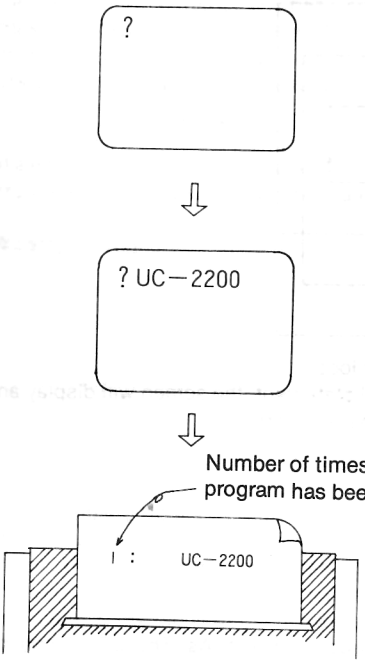
When the computer comes to a GOTO statement, it will unconditionally jump to the line named in it. In Sample Program 3 the computer is sent directly to line 10 as soon as it executes line 10 and goes on to line 20. Thus, the statement on line 10, which means, "Display an A and do not move to a new line," is executed over and over again.

●Sample Program 4

```
10 CLEAR
20 GOSUB 50
30 LPRINT A;":";B$
40 GOTO 20
50 INPUT B$
60 A=A+1
70 RETURN
```

This program has the printer print out the numbers and characters keyed in from the keyboard.

Figure A



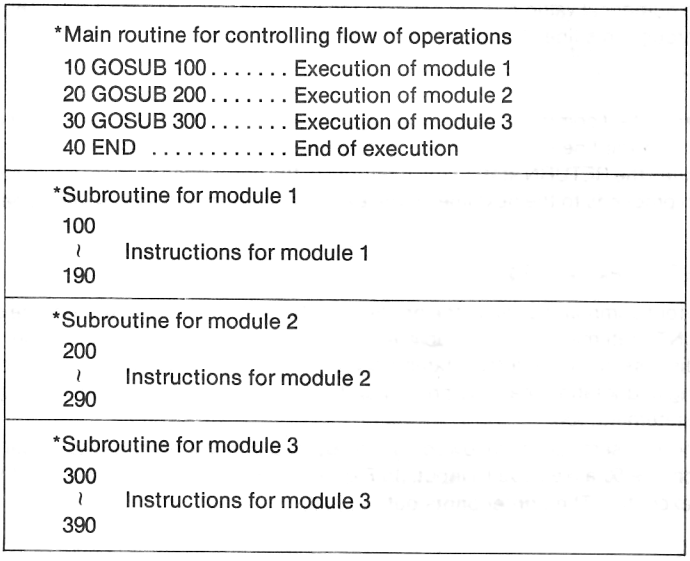
Waiting for your input from keyboard

U C − 2 2 0 0 entered from keyboard

Pressing RETURN inputs the data and has the number of times the program has been run and the data you input printed out.

The goal in programming is to write easily understandable, efficient programs. If the flow of processing steps is disorganized, your program will be a maze. Modifying it later will be time-consuming. This problem can be avoided if you analyze your ideas in the flowchart form, as was suggested earlier. But a flowchart is nothing more than a way to conceptualize what the program should do. It never directly appears in the program.

Figure B Modular Programming



One method for organizing the flow of a program makes sets of operations into independent modules, which are called by GOSUB statements. (See Figure B.) This method is quite helpful when you need to repeat a complicated set of operations. In addition, writing the program in separate modules will minimize syntax errors in your programming. The longer the program, the more the modular approach will help you.

The main program is called the main routine; the branches are the subroutines. Put this fruitful approach to work in your programs — it will help you master the world of programming.

Like the GOTO statement, the GOSUB statement used in modular programming has the computer skip irrelevant lines in running a program. But while the GOTO statement simply jumps to the designated program line, the GOSUB statement has the computer remember the number of the line from which it jumped to the subroutine. Thus, with the GOSUB command, your UC-2200 can easily return to the line from which it jumped.

Let's work through the lines of Sample Program 4 in order.

```
10 CLEAR
```

This instruction is already familiar, isn't it? CLEAR empties all variables of their contents.

```
20 GOSUB 50
```

And here comes that GOSUB statement. The computer does what it says and jumps to line 50. Of course, your UC-2200 remembers that the program was at line 20 when it jumped.

```
50 INPUT B$
```

The INPUT statement has you key in data. Key in anything you like. The variable B$ is a character string variable, which is used to store characters and numbers (but the numbers are treated not as numerical values but as characters).

```
60 A=A+1
```

The value of the numerical value A is set equal to the previous value of A plus one. Thus, in the first pass through this line, A is given the value of 1 (1 = 0 + 1).

```
70 RETURN
```

RETURN is an important command which is always used with the GOSUB statement. It is an order to go back to the line of the program containing the GOSUB statement. When the computer executes the RETURN statement, it returns to the line whose number it had stored in memory, then proceeds to the next line. In the example, it returns to line 20, then goes on to line 30.

```
30 LPRINT A;":";B$
```

LPRINT is an output command given to the printer. LPRINT statements follow the same conventions as PRINT statements. The variable A has already been given the value one. You know from the discussion of the PRINT statement that the semicolon means "to proceed to" and that anything in quotation marks is printed or displayed just as it is. Those rules apply to the LPRINT statement, too.

Then comes another semicolon, followed by B$. B$ has already been set to equal the characters which line 50 asked you to input. (In Figure A, the word UC-2200 was input.) Then line 30 is executed. The printer prints out

```
1:UC-2200
```

After printing, the program progresses to the next line.

```
40 GOTO 20
```

There's that GOTO statement we met earlier. The program jumps to line 20 unconditionally.

```
20 GOSUB 50
```

Line 20 has already been executed once. The computer is now just repeating what it has already done.

This example should suggest what a powerful tool the GOSUB statement can be. Like the GOTO statement, the GOSUB statement is frequently used in programs. It is an indispensable tool for preparing efficient programs.

●Sample Program 5

```
10 CLEAR
20 FOR I=1 TO 2000
30 IF I=1000 THEN GO
SUB 100
40 NEXT I
50 PRINT "ALL COUNTE
D"
60 END
100 PRINT "HALF COUN
TED"
110 RETURN
```

This program tells the UC-2200 to count the numbers from 1 to 2,000, report when it has counted up to 1,000, and then report when it has counted up to 2,000 and completed the job. The problem is, how should the computer decide whether it has counted up to 1,000 or 2,000. But don't underestimate it; your computer is not entirely lacking in judgment, just because it's a machine. The UC-2200 can exercise its own kind of good judgment because its BASIC includes the IF statement.

Memo 8

RELATIONAL OPERATORS

Relational operators are used to compare two values. The result of the comparison is either True (−1) or False (0). This result may then be sued to make a decision about the flow of the program.

Operator	Relation Tested	Example
=	Equality	X=Y
<>	Inequality	X<>Y
<	Less than	X<Y
>	Greater than	X>Y
<=	Less than or equal to	X<=Y
>=	Greater than or equal to	X>=Y

When arithmetic and relational operators are combined in the same expression, the arithmetic operation is always performed first. For example, the expression

X+Y<(T−1)/Z

is true if the value of X plus Y is less than the value of T minus 1 divided by Z.

Examples:

```
IF SIN(X)<0 GOTO 100
IF K<>0 THEN K=K+1
```

The format of an IF statement is as follows:

```
IF <condition> THEN <instruction>
                    <line number>
```

The IF statement is a conditional statement. The condition is a logical expression using the relational operators =, <, and >. Line 30 of the sample program states as the condition I=1000. That means that the variable I must equal 1000 for the instruction following THEN to be executed. If the condition had been I<1000, the variable I would have to be less than 1000; I>1000 would require the variable I to be more than 1000.

These relational operators are used to compare two expressions or character strings. If the condition in the IF statement is met, the instruction following THEN will be executed immediately.

The instruction following THEN can assume any form. If it is only a line number, the THEN acts like a GOTO statement.

If the condition is not met, the instruction following THEN is ignored, and the program goes to the next line number.

Memo 6

LOGICAL OPERATORS

The logical operators NOT, AND and OR can be used to refine a program's flow of control by testing combinations of relational operations by the rules of Boolean logic summarized in the following tables, where X and Y represent the relational operations being tested.

NOT:	X	NOT X
	1	0
	0	1

AND:	X	Y	X AND Y
	1	1	1
	1	0	0
	0	1	0
	0	0	0

OR:	X	Y	X OR Y
	1	1	1
	1	0	1
	0	1	1
	0	0	0

Thus, for example,

IF D<200 AND F<4 THEN 80 (program branches to line 80 if D<200 and F<4 are both ture)

IF 1>10 OR K<0 THEN 50 (program branches to line 50 if either 1>10 or K<0)

IF NOT P THEN 100 (program branches to line 100 if P is not true).

Note that logical operations are always performed after the relational operations to which they refer.

Advanced programmers may note, too, that besides the flow of control functions described above, the logical operators can also be used to perform bitwise tests and bit manipulation on integers. Thus, for example,

15 AND 14=14 (since in binary form 1111 AND 1100=1110)

4 OR 2=6 (since in binary form 0100 OR 0010=0110)

NOT 1=14 (since in binary form NOT 0001=1110).

●Sample Program 6

```
10 CLEAR
20 INPUT A
30 IF A>5 OR A<1 THE
N 20
40 ON A GOTO 100,200
,300,400,500
100 PRINT "A=1"
110 GOTO 20
200 PRINT "A=2"
210 GOTO 20
300 PRINT "A=3"
310 GOTO 20
400 PRINT "A=4"
410 GOTO 20
500 PRINT "A=5"
510 GOTO 20
```

When you run this program, the number you key in is displayed on the screen in the form "A=__". An IF statement, which you were introduced to in line 30 of Sample Program 5, is used here in line 30 to limit the range of values for A to between one and five. The noteworthy feature of this program, however, is not the IF statement but the ON-GOTO statement appearing in line 40.

```
40 ON A GOTO 100,200,300,400,500
```

The ON-GOTO statement assigns line numbers to which the computer must go depending on the value of the variable which appears between ON and GOTO (A in the sample program). If the variable equals 1, the computer must go to the first line number after the GOTO (line 100 in the sample program). If the variable equals 2, the computer goes to the second line number after the GOTO (line 200 in the sample program). If the variable equals 3, the computer goes to the third line number, and so on.

The ON-GOTO statement, therefore, divides the operating sequence into branches after line 20; where it branches depends on the value of the variable that line 20 asks you to input.

IF A>5, A<1: 20→30→20→
IF A=1: 20→30→40→100→110→20→
IF A=2: 20→30→40→200→210→20→
IF A=3: 20→30→40→300→310→20→
IF A=4: 20→30→40→400→410→20→
IF A=5: 20→30→40→500→510→20→

In an ON-GOTO statement, fractional values of variables are dropped. If the variable equaled 1.2, the decimal fraction would be ignored and the variable would be treated as equalling one.

Without the ON-GOTO statement, this program would require several IF = THEN statements. It would take the following lines of IF-THEN statements to replace the single ON-GOTO statement in line 40.

```
40 IF A=1 THEN 100
50 IF A=2 THEN 200
60 IF A=3 THEN 300
70 IF A=4 THEN 400
80 GOTO 500
```

All these IF-THEN statements are clumsy and produce an unnecessarily long, inefficient program. The ON-GOTO statement elegantly reduces them to a single line.
The ON-GOSUB statement works almost in the same way as the ON-GOTO statement. For details, review the description of the GOSUB statement.

Memo 10

FUNCTIONAL OPERATORS

Strings may be concatenated. For example,

```
10 A$="FILE" : B$="N
AME"
20 PRINT A$ + B$
30 PRINT "NEW " + A$
 + B$

   RUN
   FILENAME
   NEW FILENAME
   OK
```

Strings may be compared using the same relational operators used with numbers. String comparison can be used to test string values or alphabetize strings.

String comparisons are made by taking one character at a time from each string and comparing their ASCII codes. If all the ASCII codes differ, the lower code number precedes the higher. If, during string comparison, the end of one string is reached, the shorter string is said to be the smaller.

In string comparison, leading and trailing blanks are significant. All string constants used must be enclosed in quotation marks.

Examples:
"AA" < "AB"
"FILENAME" = "FILENAME"
"CL" < "CL"
"kg" < KG"
"SMYTH" < "SMYTHE"

●Sample Program 7

```
10 CLEAR
20 DIM A(9)
30 FOR I=0 TO 9
40 INPUT A(I)
50 NEXT I
60 FOR I=0 TO 9
70 LPRINT "A(";I;")=
";A(I)
80 NEXT I
90 END
```

When you run this program, the INPUT statement in line 40 has you key in numbers. The FOR-NEXT statement (lines 30 through 50) makes the computer repeat the input request ten times. At each request, key in any number you want — a total of ten numbers. After you have entered this data, the UC-2200 will print out the ten numbers input for A(1) through A(10) in the form "A(__)=__".

```
A( 0)= 33
A( 1)= 5
A( 2)= 108
A( 3)= 42
A( 4)= 7
A( 5)= 56
A( 6)= 900
A( 7)= 2
A( 8)= 77
A( 9)= 13
```

Line 20 of the sample program reads

```
20 DIM A(9)
```

The DIM command applies to arrays. The term "array" may be new to you. An array is a group of labeled items which have the same variable name. In the sample program, the array is the group of variables A(0) to A(9).

So far, we have used two types of variables, numbers (given names such as A, B, and C) and character strings (given names such as A$, B$, and C$). Each of these variables holds one value — character string or number — at a time. But single variables are inconvenient for handling large amounts of data. Organizing the variables into an array is much more efficient, since you can tell the computer to perform an operation on each element of the array instead of giving it separate instructions for each individual variable. Even a simple program such as Sample Program 7 would be more cumbersome without the use of an array.

If you define an array variable with the DIM statement, a group of variables is set up. Think of it as a row of boxes, as illustrated.

*DIM(9) prepares space for 10 elements in the array.