Cover Copy: The place to start if you are just beginning in
               the world of computer programming.

(title page)

Basic Computer 99/2

Book 2: BASIC for Beginners

Book 2: BASIC for Beginners--Contents

You and Computer Programming

What is computer programming? Nothing mysterious! Programming is simply
communicating with a computer--telling it what to do and when to do it.
To program your computer you only need to learn two things: the language
your computer understands, and the way you talk to it. No lengthy
training periods or super-sophisticated skills are required.

The Language--BASIC

To communicate with any computer, you need to learn its language. The
language we'll be exploring here is a form of BASIC (short for Beginners
All-purpose Symbolic Instruction Code). BASIC was developed by John
Kemeny and Thomas Kurtz at Dartmouth College during the middle 1960's.
Although BASIC is only one of many computer languages, it is one of the
most popular in use today. It's easy to learn and simple to use, yet it
is powerful enough to do almost anything you want to do with your
computer.

As you work through this book, you'll notice one striking fact about
BASIC: it's very much like English! You'll see words like PRINT, NEW,
RUN, and LIST. The meanings of these words in BASIC are almost identical
to the definitions you already know and understand. This is what makes
BASIC so easy to learn and fun to use.

The Way You Talk to the Computer--the Keyboard

Now, how do you talk to the computer? Everything you need to use to
communicate with your computer is right there on the keyboard. You type
your instructions, and the computer "hears" them.

About this Book

This book guides you step by step through the process of learning BASIC. The
material included here gives you a good foundation for the continued
development of your programming skills. Throughout the book, each explanation
of a statement or command is followed by one or more examples for you to try
out. Also, you can (and should) experiment with other examples of your own to
help you become thoroughly acquainted with the capabilitites of your computer.

Remember: the computer is a tool for your use and enjoyment, not something to
be feared. It has no intelligence, only some extraordinary capabilities. It
does exactly and only what you tell it to do; it can't do anything by itself.

TO PROGRAM THE COMPUTER:

1. Learn the language (TI-99/2 BASIC).

2. Learn the means of talking to the computer (keyboard).

How to PRINT Messages in the Immediate Mode

In the Immediate Mode, your computer "immediately" performs each BASIC
statement you've typed in as soon as you press ENTER. Because you can see an
instant response on the screen, the Immediate Mode is a good way to introduce
and explore certain TI-99/2 BASIC language statements.

Turn on your television, and then turn on the console. When the title screen
appears, press any key to begin. When you see the next screen, press 1 for
TI-99/2 BASIC.

```
-----------------------------------
        TEXAS INSTRUMENTS
             TI-99/2
        BASIC COMPUTER

READY-PRESS ANY KEY TO BEGIN
-----------------------------------

     1983  TEXAS INSTRUMENTS
```

```
        TI-99/2
     BASIC COMPUTER

PRESS
1 FOR TI-99/2 BASIC
```

The flashing underline you see on the screen is called the cursor. It tells
you that the computer is ready for you to use.

    >_ ("prompting character" and flashing cursor)

The PRINT statement tells the computer to display something on the screen.
You type the word PRINT, followed by a message enclosed in quotation marks,
and the computer prints the message when you press ENTER.

Note: Type quotation marks by holding down either the SHIFT or FCTN key while
pressing the P key.

Type the PRINT statement below:

    PRINT "THIS IS A MESSAGE"   (your statement to the computer)
    THIS IS A MESSAGE   (the computer's response)

Remember to press the ENTER key after the ending quotation marks. This is the computer's cue to perform what you have requested.

NOTE: If you make a mistake, don't worry about it. Just press ENTER and start over for right now. Correcting errors are covered a little later.

Now type this PRINT statement:

    PRINT "HI THERE"
    HI THERE

Try another example. Type these words, then press ENTER.

    PRINT "I SPEAK BASIC. DO YOU?"

(When you run out of room on a line, just keep typing--the computer automatically returns the cursor to the beginning of the next line.)

Experiment by entering more PRINT statements with your own messages. As long as you begin and end the message with quotation marks, the computer displays it.

(Notice that the lines move up on the screen when you press ENTER and again when the computer finishes printing its line. This procedure is called scrolling. The cursor shows you where the next line begins.)

Performing Calculations with the PRINT Statement

You can use PRINT to display numbers. With numbers, you do not need quotation marks. To try this, type the word PRINT, follow it by a number, and press ENTER.

    PRINT 4
     4

You can also have the computer perform calculations with the PRINT statement. Study the information below for typing arithmetic symbols on the keyboard. The computer uses the asterisk (*) for multiplication and the slash (/) for division.

plus sign	+	Use the SHIFT key and the = key.
minus sign	-	Use the SHIFT key and the / key.
times sign	*	Use the SHIFT key and the 8 key.
divide sign	/	Use the / key.
decimal point	.	Use the Period (.) key.

Try the examples on the right; then make up your own numbers. Try several kinds of calculations.

```
PRINT 3+4
 7

PRINT 86+59
 145

PRINT 1.2+46+3.1+98.6
 148.9

PRINT 6.4-3.5
 2.9

PRINT 99-18
 81

PRINT 45*9
 405

PRINT 7.98*56.07
 447.4386

PRINT 67/56
 1.196428571

PRINT 42/6
 7
```

Error Messages and Ways to Correct Errors in the Immediate Mode

Every computer programmer makes mistakes, so don't hesitate to try experiments of your own as you go through the examples in this book. Errors do not hurt the computer. It quickly recognizes things it cannot do and gives you an error message such as INCORRECT STATEMENT or CAN'T DO THAT to tell you to try again. When this happens, you can simply identify the error, retype your line correctly, and press ENTER again.

Some of the most common errors are typing a wrong letter and omitting a necessary part of a statement. For example, here are some things that your computer does not accept in a PRINT statement:
    1. A misspelling in the word PRINT.
    2. A missing or extra quotation mark.
    3. Spaces within the word PRINT.

Experiment with some intentional errors to become more comfortable with error messages.

(1) Misspelling in the word PRINT

    PIRNT "THIS IS A MESSAGE"

    ** INCORRECT STATEMENT **

(2) Missing or extra quotation marks

    PRINT "THIS IS A MESSAGE

    ** INCORRECT STATEMENT **

(3) Spaces within the word PRINT

    P RINT "THIS IS A MESSAGE"

    ** INCORRECT STATEMENT **

Try a few more messages with the PRINT statement, introducing intentional errors so that you become familiar with the error messages.

If, however, you see an error before you press ENTER, you can correct it. The following shows two ways to correct errors in the Immediate Mode (if you have not yet pressed ENTER):

1. While holding down either the SHIFT or the FCTN key, use LEFT ARROW (S key) to backspace to the error and correct the error by typing over it.

    If, while backspacing, you move the cursor past the error, move the cursor to the right with RIGHT ARROW (SHIFT D or FCTN D) until the cursor is positioned over the error. (Note that characters are not erased as you move the cursor over them. If you need to erase a character or word, use the SPACE BAR to advance the cursor over the character.)

2. You can press ERASE (FCTN 3) to erase the entire line and start over. (Hold down the FCTN key and press 3).

Now look back at the PRINT statements on the previous page. Type the commands incorrectly (exactly as written), but instead of pressing ENTER and getting an error message, correct the errors using one of the above methods.

Experiment with your own intentional errors and practice correcting them before you press ENTER. As you continue your work with the computer, make use of these methods to correct errors. (We'll discuss other ways to correct errors later in this book.)

Using Numeric Variables and the LET Statement

A variable is a "name" given to a number or a group of characters. Although there are two types of variables, in this section we'll consider only those variables that give names to numbers. These are called numeric variables. A numeric variable is just a name given to a numeric value.

As the word _variable_ implies, the value of a variable can be made to change or vary. A number is assigned to a numeric variable with the LET statement. Variables can be up to 15 characters long, but they are generally kept fairly short for convenience.

In the LET statement the word LET is followed by one space, then the variable (the name), then an equals sign, and finally the numeric value you are assigning to the variable.

Try a few examples. Type in the following lines, pressing ENTER at the end of each line.

```
LET A=5
LET A2=8
LET ALPHA=10
```

You can think of variables as labeled boxes that hold assigned values. Only one value at a time may be assigned to a given variable, but you can change a value easily. Type this LET statement, pressing ENTER at the end of the line.

```
LET A=8
```

The value of A is no longer 5. The 5 has been replaced by 8.

Now use PRINT statements to check the values you have entered. Type PRINT A and press ENTER.

```
PRINT A
8
```

Did you notice that this PRINT statement is different from the PRINT
statements we explored earlier? We didn't put quotation marks around the A
even though it is a letter. That's because we didn't want to print the letter
A; we wanted to see the numeric value assigned to A.

Now, check for the values of A2 and ALPHA. (Remember to press the ENTER key
at the end of each line, even though it isn't shown.)

        PRINT A2
         8

        PRINT ALPHA
         10

The value of every numeric variable is zero if it has not yet been assigned a
value. Try the following PRINT statement.

        PRINT F
         0

Note: In BASIC, the LET statement is not the only way to assign a numeric
value to a variable. Your computer also accepts the assignment without the
word LET.

        JACK=3

        JILL=5

        PRINT JACK*JILL
         15

In other words, the word LET is optional in BASIC; your computer accepts the
assignment with or without it.

Assigning Numeric Expressions to a Numeric Variable

You have just seen how a numeric variable can be assigned a single-number value. An arithmetic expression (such as 4*5 or A+1) can also be assigned to a variable.

(If you just completed the previous section, the computer still holds the following values in memory. If you are just now beginning a new session with the computer, enter these statements again.).

```
LET A2=8
LET ALPHA=10
```

You can assign new values to the variable A and check the new value with a PRINT statement after each one. Enter the following lines. Note that the variable always appears before the equal sign and the expression always appears following the equal sign.

```
LET A=5*3
PRINT A
 15


LET A=A2+ALPHA
PRINT A
 18


LET A=A-5
PRINT A
 13


LET A=A/2
PRINT A
 6.5
```

Using the CALL CLEAR Statement

As you type instructions to the computer in the Immediate Mode, the screen
eventually fills with instructions and the resulting displays. The lines
scroll up the screen as you enter additional instructions. There is an easy
way to clear the screen of previous lines so that you can concentrate on the
line you are currently typing. This is done with the CALL CLEAR statement.

When you enter the words CALL CLEAR, the screen is cleared except for the
prompting symbol and the cursor. You can use the CALL CLEAR statement anytime
you like, whether the screen is filled or not.

```
A=36
CALL CLEAR
```

   (screen is cleared and the cursor returns to the lower-left-hand corner of
   the screen)

The CALL CLEAR statement clears the screen, but not the computer's memory. If
you assign a value to a variable and then clear the screen, the computer still
holds that value in memory. To verify this, tell the computer to PRINT the
value of A.

```
PRINT A
 36
```

Note: As you work through this book, you'll see several BASIC statements that
begin with the word CALL. Your computer has certain built-in subprograms for
special purposes (such as clearing the screen), and a CALL statement tells the
computer to "call" the subprogram named in the statement.

Using String Variables and the LET Statement

The words that you told the computer to PRINT a few pages back are called
character strings. A "string" of characters is anything that you enclose in
quotation marks. Usually this means alpha characters (letters of the
alphabet), but a string can include any character the keyboard can type,
including numbers, letters, punctuation, spaces, and symbols.

A string variable is a name given to a "string" of characters. A string
variable name always has a dollar sign ($) at the end of it. Because of this,
you (and the computer) can always tell string variables from numeric variables.

You already know what numeric variables are: numeric values assigned to names
(variables), like "K=50". String variables differ from numeric variables in
the following ways.
1. The variable name must end with a $.
2. The alphanumeric characters in the string must be enclosed in
   quotation marks.
3. Strings of numbers cannot have arithmetic operations performed with or
   upon them.

Try a couple of examples. Clear the screen (with CALL CLEAR). Using SHIFT 4
for the dollar sign ($), enter this.

    LET W$="HASTE MAKES WASTE"

    PRINT W$
    HASTE MAKES WASTE

Notice that the character string is displayed, rather than the string
variable. As with numeric variables, do not put quotation marks around the
variable in the PRINT statement because you do not want to print the
characters "W$"; you want to see the value assigned to W$.

Correcting Errors with DELete and INSert

Earlier you practiced correcting errors by using the ARROW keys (S or D) in
conjunction with either the FCTN or SHIFT keys. You also used ERASE (FCTN 3)
to erase the entire line and start over. There are two other key combinations
that you can use to correct errors. These are:

   DELete (FCTN 1)--deletes character(s)

   INSert (FCTN 2)--inserts character(s)

Type the following statement exactly as written, but don't press ENTER yet.

   PRINT I M LENING TO CORRRRRRECT ERROORS"

There are obviously several errors in this statement. First, you need to
insert quotation marks before the character I. Backspace with LEFT ARROW to
the I. Press INSert (FCTN 2); then type quotation marks (").

   PRINT "I M LENING TO CORRRRRRECT ERROORS"

Next, insert an A before the M. To do this, advance the cursor (with RIGHT
ARROW) to the M. Press INSert and then type an A.

   PRINT "I AM LENING TO CORRRRRRECT ERROORS"

Notice that to insert a character, you must determine where you want it
inserted and then position the cursor over the character immediately following
that point. Now position the cursor over the first N in LENING. Then insert
an A and an R in LENING so that it reads LEARNING.

   PRINT "I AM LEARNING TO CORRRRRRECT ERROORS"

You can insert or delete single characters or as many characters as you want. To delete characters, position the cursor over the first character you want to delete. Now let's delete four of the Rs in CORRRRRRECT. Position the cursor over the first R; then press DELete four times (while holding down the ECTN key, press 1 four times). If you press ECTN 1 down for more than one second, the automatic repeat feature deletes characters more quickly. However, you must be careful not to delete too many characters, or you will have to insert them again.

    PRINT "I AM LEARNING TO CORRECT ERROORS"

Now delete one of the Os in ERROORS using what you have learned. Look over the statement and correct any other errors (if you have made any). Then press ENTER.

    PRINT "I AM LEARNING TO CORRECT ERRORS" (press ENTER)
    I AM LEARNING TO CORRECT ERRORS

Using the Comma (,), Semicolon (;), and Colon (:) as PRINT separators

A single PRINT statement can be used to print two or more items. By using the comma, semicolon, or colon, you can control the way the computer displays these items. Try these examples.

```
CALL CLEAR
A2=8
ALPHA=10
PRINT A2,ALPHA
 8              10
```

Now, try these:

```
AL=6
ALBERT=8
PRINT AL;ALBERT
 6  8
```

The computer divides the display screen into two horizontal zones. When you use a comma (,) between two (or more) variables in a print statement, you are telling the computer to print the values in different zones. On the other hand, the semicolon (;) instructs the computer to print the numbers close together.

Now try this example, using the colon (:).

```
PRINT AL:ALBERT
 6
 8
```

The colon has the computer print each item on separate lines.

Now try an example that uses character strings.

```
N$="JACK SPRAT"

PRINT N$
JACK SPRAT
```

```
W$=" ATE NO FAT."          (Note the one space at the beginning of
PRINT W$                    this string to prevent its running on with
 ATE NO FAT.                the previous word.)
```

```
PRINT N$;W$                (Computer prints the two strings close
JACK SPRAT ATE NO FAT.     together on one line.)
```

Now print the two strings again using first the comma (,) and then the colon
(:) as print separators to see the differences:

```
PRINT N$,W$
JACK SPRAT     ATE NO FAT.
```

```
PRINT N$:W$
JACK SPRAT
 ATE NO FAT
```

Using the NEW Command

The words "command" and "statement" are sometimes used interchangeably.
Generally, commands are used in the Immediate Mode (sometimes called the
Command Mode, without line numbers) and statements are used in programs (with
line numbers; this is covered a bit later).

The NEW command produces visible results similar to those of the CALL CLEAR
statement in that both clear the screen. An important difference, however, is
that the NEW command also clears the computer's memory. When you use the NEW
command, any information you have entered is erased. This should be done when
you begin a new activity (in the Immediate Mode) or a new program, so that old
information that the computer may be storing (such as the value assigned to a
variable) does not interfere with what you are about to do.

On the previous page you assigned and printed several numeric and string
variables. These values are still stored in the computer's memory. (If you
are beginning a new session with the computer, enter these statements again.)
They are:

```
A2=8
ALPHA=10
AL=6
ALBERT=8
N$="JACK SPRAT"
W$=" ATE NO FAT."
```

If you clear the screen with a CALL CLEAR command, you can still print the
values of these variables, as you did in the section on the CALL CLEAR
command. Try it:

```
CALL CLEAR
PRINT ALPHA;A2;N$
 10  8 JACK SPRAT
```

```
CALL CLEAR
PRINT ALBERT:AL:W$
  8
  6
  ATE NO FAT.
```

If you enter the NEW command, however, these values are erased from memory and cannot be printed. The NEW command has the same effect on the computer's memory as turning the computer off and then back on again. Enter the following:

```
NEW
PRINT ALPHA
0

PRINT N$
```

If you print the value of a numeric variable, a zero is displayed. If you print the value of a string variable, a blank line is displayed, because the string is empty.

Computer Graphics--Positioning Characters with CALL VCHAR and CALL HCHAR

One of the most exciting things you can do with your computer is to create graphic designs right on the screen. With your computer's graphic capability, you can make a design, draw a picture, create a gameboard, and so on.

This section introduces you to two simple yet powerful graphics statements. CALL VCHAR and CALL HCHAR are used to position a character or draw a line of characters on the screen. Later in this book, we'll show you how to use graphics statements in programs.

The Basic Computer 99/2 uses 28 printing positions on each line. For graphics, however, the computer allows 32 character positions on each line. Think of the screen as a "grid" of square blocks made of 32 columns and 24 rows.

ILLUSTRATION (grid with 32 columns and 24 rows)

Each square on the grid is identified by two values called coordinates--a row number and a column number. For example, the coordinates 5,7 mean the fifth row and the seventh column, and the coordinates 10,11 mean the tenth row and the eleventh column.

The first thing to try is to place a character in a particular square on the screen. For the time being, consider that a character is one of the 26 letters of the alphabet, the numbers 0 through 9, and certain other symbols, such as the asterisk (*), the plus and minus signs (+ and -), and the slash (/). (Later you will learn about other characters available for graphics.) Each character is assigned an identifying numeric value of its own. The values for the full character set are given in Appendix XX in Book 4.

By using either CALL VCHAR or CALL HCHAR, naming the two coordinates (row and column), and identifying a character by its numeric value, you can place the character in any spot you choose. Here's the form used for these two statements.

    CALL VCHAR(12,17,42)   (row 12, column 17, character number 42--the
                                asterisk)

Try this example, and you'll see an asterisk (*) appear near the center of the screen.

Try a few more examples. First, clear the screen by typing CALL CLEAR and pressing ENTER. Type the following.

    CALL VCHAR(15,10,67)   (row 15, column 10, character number 67--C)

Don't forget the parentheses in the statement--they are important! Now try the CALL HCHAR statement.

    CALL HCHAR(16,10,67)   (row 16, column 10, character number 67--C)

The order for entering the row number, the column number, and the character's numeric value is the same for both CALL VCHAR and CALL HCHAR, and they do the same thing when you are positioning a single character on the screen.

Using the Repetition Feature of CALL HCHAR and CALL VCHAR

When you try to draw a line of characters, you find that there is a distinct
difference between the functions of the statements, CALL HCHAR and CALL
VCHAR. CALL VCHAR causes a vertical column of characters to appear, whereas
CALL HCHAR displays a horizontal row of characters. To display a line with
either statement, you add a fourth numeric value to the statement: the number
of repetitions you want. This number controls the "length" of the line.

Clear the screen by typing CALL CLEAR and pressing ENTER, and try a vertical
line. Type this:

    CALL VCHAR(11,10,86,8)   (row 11, column 10, character number 86--V, 8
                                  repetitions)

Check for errors and then press ENTER. The screen looks like this:

    CALL VCHAR(11,10,86,8)

            V
            V
            V
            V
            V
            V
            V
            V

As mentioned earlier, there are 24 horizontal rows of character blocks on the
"grid" of the screen. Therefore, you can only draw a vertical line (column)
that is 24 characters long. What happens if you enter a repeat value greater
than 24?

Clear the screen and then type the following.

    CALL VCHAR(1,10,86,50)

When you press ENTER, the screen shows the following display.

            VVV
            VV
            VV
            VV
            VV
            VV
            VV
            VV
            VV
            VV
            VV
            VV
            VV
            VV
            VV
        CALL VCVVR(1,10,86,50)
            VV

(Note: You don't actually see all 50 of the V's above because there is a blank
line after the CALL VCHAR command.)

Type

```
CALL CLEAR
CALL HCHAR(17,1,72,50)
```

and you see

```
CALL HCHAR(17,1,72,50)
```

```
HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
HHHHHHHHHHHHHHHHHHH
```

So far, you have entered actual numeric values in your statements. However, you can assign numeric values to variables and then use the variables in the CALL VCHAR and CALL HCHAR statements. Try this:

```
ROW=5
COLUMN=12
CHARCODE=67
CALL CLEAR
CALL VCHAR(ROW,COLUMN,CHARCODE)
```

Where did the "C" appear on the screen?

For a big finale, fill the screen with asterisks (numeric code 42). Type these lines, pressing ENTER at the end of each line.

```
CALL CLEAR
CALL HCHAR(1,1,42,768)
```

Continue to experiment on your own, trying different characters (see Appendix XX) and positions. For example, can you fill the screen with your first-name initial?

Review

_____1. A variable is
     A. a mistake that is made repeatedly.
     B. a statement used in the Immedi e Mode.
     C. a word or letter that is assig d a particular value.
     D. a value that always remains the same.

_____2. The difference between numeric variables and string variables is that
     A. one is for serious programming and one is just for fun.
     B. string variables are always longer.
     C. numbers are assigned to numeric variables and characters are
        assigned to string variables.
     D. numerals are assigned to numeric variables and numbers are assigned
        to string variables.

_____3. The difference between NEW and CALL CLEAR is
     A. that NEW only clears the screen and CALL CLEAR clears the
        computer's memory.
     B. not important.
     C. impossible to explain.
     D. that CALL CLEAR only clears the screen and NEW both clears the
        screen and erases the computer's memory.

_____4. Which of the following are valid ways to correct errors?
     A. Backspace with LEFT ARROW and type over the error.
     B. Use DEL (FCTN 1) to delete incorrect characters.
     C. Use INS (FCTN 2) to insert correct characters.
     D. Press ERASE (FCTN 3) and type the line again.
     E. All of the above.

5. Match each type of punctuation with the resulting display (when the
   particular punctuation is used between two items in a PRINT statement).
   _____A. comma (,)   X. items printed close together
   _____B. semicolon (;)   Y. items on separate lines
   _____C. colon (:)   Z. each item is in a different print zone

_____6. The numbers within the parentheses after a CALL HCHAR or CALL VCHAR
       command specify (in the correct order):
     A. (row, column, character-code, number of repetitions)
     B. (character-code, number of repetitions, row, column)
     C. (column, row, character-code, number of repetitions)
     D. (number of repetitions, row, column, character-code)

Simple Programming--Line Numbers

So far you have been entering single instructions and the computer has performed them immediately. A computer program is simply a list of these instructions that the computer performs in a certain order. A program is different because the computer waits until you have entered all your instructions and does not perform them until you tell it to. Thus you can enter the program, correct errors, and revise or edit as much as you want until you are ready for the computer to perform the program. Then it performs the instructions in rapid succession.

How does the computer know that the instruction that you enter is not to be performed immediately? This is done by putting line numbers in front of the instructions to show that each instruction is just one in a series.

In a computer program, each statement begins with a line number, which serves two important functions:

1.   It tells the computer not to perform the statement immediately, but to store it in memory when you press ENTER.

2.   It establishes the order in which the statements are to be performed in the program.

Let's begin by using an old familiar friend, the PRINT statement, in a program. First type the word NEW and press ENTER. Now type the following program, pressing ENTER at the end of each program line:

```
10 PRINT "ARE YOU READY"        (one space after each line number
20 PRINT "TO LEARN BASIC?"      is required)
30 END
```

(As you type the program, notice the small "prompting" character that appears just to the left of the printing area. This symbol marks the beginning of each program line you type.)

In computer terminology, you have just "entered" a program. Nothing to it! Check the program now to see if there are any typing mistakes. If there are, just retype the line correctly, including the number at the beginning of the line, right there at the bottom of the screen. Then press ENTER. The computer automatically replaces the old line with the new, correct one.

Also, you may be wondering why we numbered the lines in increments of ten (10, 20, 30, etc.). Well, we could just as easily have numbered them 1, 2, 3. By using increments of ten, however, or other spreads like 100, 200, 300, etc., we can go back and insert new lines if we want to expand the existing program, and we don't have to retype the whole program! (We'll cover this clever trick when we discuss editing a program.)

When you're ready to see the program in action, type CALL CLEAR and press ENTER. The screen is cleared, but your program is not erased--it's stored in the computer's memory!

Using the RUN Command

The RUN command is the command that tells the computer to perform the list of
instructions you have given it. This is called running a program.

With the program you entered on the previous page still in the computer's
memory, we are now ready to RUN it. Type RUN and press ENTER again.

    RUN
    ARE YOU READY
    TO LEARN BASIC?

    ** DONE **

Want to "run" the program again? Type RUN again and press ENTER.

    RUN
    ARE YOU READY
    TO LEARN BASIC?

    ** DONE **

    RUN
    ARE YOU READY
    TO LEARN BASIC?

    ** DONE **

Each time you type RUN and press ENTER, the computer begins at the first
statement and follows your instructions in order until it reaches the last
statement. END means just what it says: the end, stop!

Using the LIST Command to Review Your Program

The LIST command tells the computer to display, in the correct order according to line numbers, the current program in memory.

Now that you've had a bit of programming experience, let's review some of the things you did when you entered the previous program. To refresh your memory, we'll get the program back on the screen.

First, type CALL CLEAR (without a line number) and press ENTER to clear the screen. Now type LIST and press ENTER again:

```
LIST
10 PRINT "ARE YOU READY"
20 PRINT "TO LEARN BASIC?"
30 END
```

The program above consists of three statements or "lines." As in the Immediate Mode, you pressed ENTER when you finished typing each program line. Pressing ENTER defines the end of the program line, just as the line number identifies the beginning of the line. It is also the computer's cue to store the line in it's memory. Pressing ENTER at the end of each program line is essential--without it, your line will not be correctly stored by the computer.

Now type NEW and then LIST. What happens?

```
NEW
LIST

* CAN'T DO THAT
```

You get the error message CAN'T DO THAT. You have asked the computer to do something impossible, since it can't LIST a program if it has been erased with the NEW command.

How to Edit a Program--Correcting Errors

You have already practiced correcting errors in instructions by backspacing
with LEFT ARROW or by pressing ERASE and beginning the instruction again. You
also practiced correcting errors with DELete and INSert. These methods work
(if you have not yet pressed ENTER) both in the Immediate Mode and when
entering program lines. If you have already pressed ENTER, there are several
other ways to edit a program line, one of which we mentioned when you entered
your first program a few pages back. These are:

1. Retyping the line correctly, including the line number, and pressing
   ENTER again. The computer will replace the old line with the new
   correct line in its memory.

2. Using the EDIT command followed by the line number of the line which
   you wish to edit. (Or you can simply type the line number and then
   press UP ARROW (FCTN E or SHIFT E) or DOWN ARROW (FCTN X or SHIFT X).
   The current line appears, and you simply use the ARROW keys to position
   the cursor over the error and correct the error by typing over it.
   Editing by these methods often requires less typing.

Let's practice editing by using the above methods. Below is a program which
converts pounds to kilograms. Enter the program just as it is written (it has
some intentional errors in it).

    10 K=600
    20 P=2.2*J
    30 PRINT H
    40 END

Change the number 600 to 60 in line 10. Use the first method to correct this
line. Simply type the corrected line, press ENTER, and the computer replaces
the old line in memory. Type

    10 K=60

and press ENTER.

Now to prove that the corrected line is in the computer's memory, clear the screen (CALL CLEAR) and list the program (LIST). This is what you see.

```
10 K=60
20 P=2.2*J
30 PRINT H
40 END
```

The variable J should be changed to K in line 20. To fix this, let's use the second method we discussed: using the EDIT command. Enter

```
EDIT 20
```

Line 20 appears, with the cursor flashing over the variable P. Using RIGHT ARROW (SHIFT D or FCTN D), position the cursor over the J and type a K over it. Then press ENTER. Now clear the screen and LIST the program.

As you can see, the computer has the corrected version of line 20 in memory. Now what is the error in line 30? The variable H has not been assigned a value. Actually, we want to PRINT the value of P. Type 30 and press UP ARROW (FCTN E or SHIFT E).

When line 30 appears, move the cursor to th. H at the end of the statement, type a P over the H, and press ENTER.

Now if you like, you can list the program once more to see the whole program corrected.

```
10 K=60
20 P=2.2*K
30 PRINT P
40 END
```

Let's study what this program does. We said it converts kilograms to pounds (1 kilogram = 2.2 pounds). We've used the variables K (for kilograms) and P (for pounds) to help us remember which value is which, and we began our program by assigning values to these variables.

In this case, we are trying to find out how many pounds are equal to 60 kilograms, so we have defined K as 60. Notice that we have defined P as 2.2 X K. If we stopped here and ran the program at this point, the computer would perform the conversion, but it wouldn't show us the answer! So we added the PRINT statement.

Now RUN the program. What is the answer?

Adding Program Lines

What you have just done is called "editing" a program. The ability to edit or change a program without retyping the whole thing is one you'll come to value highly as your programming skills grow. To give you an idea of the great flexibility editing adds to programming, let's experiment with a few more changes in the present program.

We mentioned earlier that the reason we number program lines in increments of 10 (instead of 1, 2, 3, etc.) is to allow program lines to be added without having to retype the whole program. Before we experiment with a few examples, let's clear the screen and recall our program. Type CALL CLEAR, then LIST

```
LIST
10 K=60
20 P=2.2*K
30 PRINT P
40 END
```

We might want to add a CALL CLEAR statement to the program, so that we won't have to keep clearing the screen from the keyboard each time we "run" the program. Type:

```
5 CALL CLEAR
```

Now LIST the program again to see the new line (type LIST and press ENTER).

```
           LIST
The old    10 K=60
program    20 P=2.2*K
           30 PRINT P
           40 END


           5 CALL CLEAR
           LIST
The new    5 CALL CLEAR
program    10 K=60
           20 P=2.2*K
           30 PRINT P
           40 END
```

Compare the two programs on the screen, and notice that the computer has automatically placed the new line in its proper order. Run the program again to see the effect of the added line.

Now let's add another line that helps to point out our answer. Type

```
           27 PRINT "THE ANSWER IS:"
```

and press ENTER. When you run the program again, you'll see this:

```
           THE ANSWER IS:
            132

           ** DONE **
```

Removing Program Lines

Quite often it's necessary to remove a line or lines from a program. Deleting a program line is a very simple procedure.

The program we have stored right now doesn't really have any lines we want to delete. Just for practice, however, let's remove line 5.

First, clear the screen and list the program as it is now. Line 5 is the first line of the program, a CALL CLEAR statement. To remove it, simply type 5 and press ENTER. Then LIST the program again. Presto! Line 5 is gone!

```
                LIST
Old             5 CALL CLEAR
program         10 K=60
                20 P=2.2*K
                27 PRINT "THE ANSWER IS:"
                30 PRINT P
                40 END


                5               (Here's where we deleted line 5.)
                LIST
New             10 K=60
program         20 P=2.2*K
                27 PRINT "THE ANSWER IS:"
                30 PRINT P
                40 END
```

That's all there is to it. To remove a line, type the line number and press ENTER. The computer then deletes the line from program memory.

Since we really need line 5 in this program, let's reenter it. Type

```
        5 CALL CLEAR
```

and press ENTER.

Using the INPUT Statement with a Numeric Variable

The INPUT statement tells the computer to stop the program in progress and wait for input from the keyboard. The value you enter is then assigned to the variable contained in the INPUT statement. Thus, the INPUT statement, like the LET statement, is a way of assigning values to variables.

If you want a value for a variable to be different each time a program is run, the INPUT statement is better than the LET statement because the program itself does not have to be changed.

In the conversion program we have been working with, you can easily change the value of K simply by retyping line 10 to assign a new value. (Remember that the word LET is not necessary when assigning a value to a variable.) Try it by typing

        10 K=40

Then run the program. The answer you get is the number of pounds equivalent to 40 kilograms. But suppose you had many values for K, and you wanted to find the equivalent value of P for each one. It would become rather tiresome to retype line 10 each time.

An INPUT statement causes the computer to display a question mark and stop, waiting for you to type in a value and press ENTER. The value you enter is then assigned to the variable contained in the INPUT statement. For example, type
        10 INPUT K

and press ENTER. Now run the program again.

The question mark and cursor show you that the computer is waiting for you to "input" a value for K. This time we'll let K=70, so type 70 and press ENTER. The computer prints your answer:

        ? 70
        THE ANSWER IS:
         154

        ** DONE **

Now you can run the program as many times as you like, changing the value of K each time the computer prints a question mark and stops. Try the program several times with different values for K.

The INPUT statement can also be used to print a "prompting" message (instead of simply a question mark) that helps you remember what value the computer is asking for. Change line 10 again by typing

    10 INPUT "KILOGRAMS?":K

and pressing ENTER. Now run the program again. First the program asks:

    KILOGRAMS?

Let's let K=50 this time. Type 50 and press ENTER.

    KILOGRAMS?50
    THE ANSWER IS:
     110

    ** DONE **

By now, your program looks like this:

    5 CALL CLEAR
    10 INPUT "KILOGRAMS?":K
    20 P=2.2*K
    27 PRINT "THE ANSWER IS:"
    30 PRINT P
    40 END

If you'd like, you can list it on the screen at this time and review the changes you've made so far. When you're ready, we'll go on to look at one more change.

Using the INPUT Statement with a String Variable

Let's make your conversion program a little more personal by using a string variable. Type these two lines:

    8 INPUT "NAME, PLEASE?":B$

    26 PRINT "OK, ";B$

(Clear the screen and list the program again so you can see how the new lines fit in.)

When you run the program this time, the two INPUT statements will stop the program twice:

The computer asks:	You type in:
NAME, PLEASE?	Your name and then press ENTER.
KILOGRAMS?	The number of kilograms and then press ENTER.

Let's try it. Type RUN and press ENTER.

    NAME, PLEASE?

We'll type in HARRY (that's a nice name) and press ENTER. Then we'll see

    NAME, PLEASE?HARRY
    KILOGRAMS?

Again let's type 70 for the number of kilograms. Press ENTER again and you'll see:

    NAME, PLEASE?HARRY
    KILOGRAMS?70
    OK, HARRY
    THE ANSWER IS:
     154

    ** DONE **

Using the GO TO Statement--Endless Loops--The BREAK Key

So far in your programming studies, you have seen that the computer performs
the instructions in a program in the exact order that they are listed
(according to line numbers). When the computer finishes the last instruction
in a program, it stops. There are ways to change this order or make the
computer repeat a series of lines over and over. One of the statements that
allows you to do this is the GO TO statement. (GO TO can be typed as GOTO in
a program. The computer accepts it either way).

The GO TO statement tells the computer to do exactly what it says: go to a
different line than the one the computer would normally perform next (the next
line in succession).

So far, you've been developing programs that operate from beginning to end in
a straight sequential order. There are many situations, however, in which you
want to interrupt this orderly flow of operation. Look at the following
program, but don't enter it yet:

```
10 CALL CLEAR
20 INPUT K
30 PRINT K
40 PRINT:::::::::::
50 K=K+1
60 GO TO 30
```

Here we "send" the program back to line 30 by using a GO TO statement in line
60. The GO TO statement causes the actions performed by lines 30, 40, and 50
to be repeated over and over again, setting up what's called a loop.

Notice that we don't use an END statement. That's because the program never
gets beyond line 60! (The END statement isn't necessary in BASIC anyway.)
The computer won't stop until you tell it to by pressing BREAK (the key at the
lower-left-hand corner of the keyboard. You can get the same result with
CLEAR [ECIN 4]). This is called an "endless loop."

Let's enter the program now. First, type NEW and press <u>ENTER</u> to erase the computer's memory, and then type these lines:

```
10 CALL CLEAR
20 INPUT K
30 PRINT K
40 PRINT::::::::::
50 K=K+1
60 GO TO 30
```

Before you run the program, we'll examine a diagram called a flowchart, explaining how the program works.

Program Line	Operation
10 CALL CLEAR	Clears the screen
20 INPUT K	Stops and waits for initial value of K
30 PRINT K	Prints the current value of K
40 PRINT::::::::::	Prints nothing; just gives you 10 blank lines
50 K=K+1	Reassigns a new value to K (the old value +1)
60 GO TO 30	Transfers the program back to line 30

Now run the program, putting in 1 for the beginning value of K. Watch how quickly the computer counts--almost too fast to follow! That's why we added line 40--to display some blank lines. This line puts ten blank lines in between the numbers (with ten colons) so that you can see the numbers better.

Let the computer count as long as you want to. When you are ready to stop the program, press <u>BREAK</u>. You'll see *BREAKPOINT AT <u>line-number</u> on the screen, indicating where the program stopped. Run the program as many times as you want, using whatever number you wish as the initial value for K (50, 100, 500, etc.).

More Practice with the GOTO Statement

If you try to send the program to a non-existent line number, however, you'll get an error message.

(From here on we'll use GOTO instead of GO TO, since the computer accepts it either way.) Suppose, for example, we type in

    60 GOTO 25

and press ENTER. Try it, run the program, and see what happens! You'll see this error message:

    * BAD LINE NUMBER IN 60

So correct the line by typing and entering

    60 GOTO 30

and run the program again.

Can we change the program to make it count by 2's, or 5's? You bet we can! By making one program change, let's make the computer count by 2's: Type

    50 K=K+2

and press ENTER. Now run the program, typing in 2 when the computer asks for the starting value of K.

Experiment with the program for a while, making it count by 3's, 5's, 10's, etc.

Using the FOR-NEXT Loop

The FOR-NEXT loop is a way to make the computer repeat a series of program
lines a specified number of times and then continue with the rest of the
program.

Earlier we presented several examples of the GOTO loop, which repeats a set of
statements indefinitely--or until you press BREAK to stop the program. The
FOR and NEXT statements also create a loop, but they are different from GOTO
in two important ways:

1. The FOR and NEXT statements are two lines in the program, the FOR line and
    the NEXT line, each with its own line number.

2. You control the number of times the loop is performed. After the loop has
    been "executed" the number of times you specify, the program moves on to
    the line that follows the NEXT line.

The FOR line has the form

    30 FOR A=1 TO 3

The NEXT line could be

    80 NEXT A

These two lines cause the portion of the program between the FOR and NEXT
lines to be performed three times. In this example, the starting value of A
is 1; after each pass through the loop, A is increased by 1. Its value is
then tested against the upper limit (3, in this example). After the third
pass through the loop, A is equal to 4, so the program "exits" (or leaves) the
loop to the line following the NEXT line, which is line 80.

To help you see the differences between GOTO and FOR NEXT more clearly, let's compare two similar programs, one with a GOTO loop and one with a FOR-NEXT loop.

A GOTO Loop

Type NEW, press ENTER, and then enter this program:

```
10 CALL CLEAR
20 A=1
30 PRINT "A=";A
40 A=A+1
50 GOTO 30
```

Before you run the program, think for a few minutes about what it will do. First, the initial value of the variable A will be set to 1. Then the computer will print out the current value of A. Finally, the value of A will be increased by 1, and the program will loop back to line 30. It will go on with this procedure until you press BREAK.

Ready to run the program? Type RUN and press ENTER to see it in action. When you're ready to stop it, press BREAK.

A FOR-NEXT Loop

Now let's examine a similar "counting" program with a FOR-NEXT loop. Type NEW and press ENTER to erase the first program. Then type these lines:

```
10 CALL CLEAR
20 FOR A=1 TO 5
30 PRINT "A=";A
40 NEXT A
50 PRINT "OUT OF LOOP"
60 PRINT "A=";A
```

Think about the way this program will be performed. The value of A will start
at 1 and will be increased by 1 each time the program completes line 40. As
soon as the value of A is greater than 5, the program will exit the loop and
continue with line 50. If we listed the lines in their order of performance,
along with the increasing values of A this is what we would have:

Line Number	Value of A
10	0
20	1
30	1
40	2
30	2
40	3
30	3
40	4
30	4
40	5
30	5
40	6
50	6
60	6

Run the program, and the screen should look like this:

```
A= 1
A= 2
A= 3
A= 4
A= 5
OUT OF LOOP
A= 6

** DONE **
```

The following flowcharts illustrate the differences in the two programs.

GOTO Program	FOR-NEXT Program
Clear screen.	Clear screen.
Set initial value of A.	Set the "parameters" for A: beginning and ending values.
Print "A=" and current value of A.	Print "A=" and current value of A.
Increase A by 1.	
Loop back to line 30.	Increase A by 1; check to see if the new value for A exceeds the upper limit set by line 20. If the answer is "no," repeat lines 30 and 40. If "yes," break out of loop.
(Loop continues until you stop the program by pressing BREAK.)	
	Print "Out of Loop."
	Print "A=" and current value of A.
	Stop program run.

We can use the FOR and NEXT statements to build a controlled time delay into a program. Consider this example:

```
20 FOR A=1 TO 1000
30 NEXT A
```

Better still, let's try it! Type NEW, press ENTER, and then type in the following program:

```
10 CALL CLEAR
20 FOR A=1 TO 1000
30 NEXT A
```

Now run the program. What happens on the screen? Not much, really. The cursor disappears. After a short time delay (while the computer "counts" from 1 to 1000), the cursor reappears and the program ends:

** DONE **

Although no other lines are being executed between the FOR and NEXT lines, time passes while the computer counts the number of loops, in this example from 1 to 1000.

Nested FOR-NEXT loops

It is possible for us to use more than one FOR-NEXT loop--one inside
another--in a program. We call these nested loops.

Now let's examine a program with nested FOR-NEXT loops. The following program
displays sixty-four of the alphanumeric characters, codes 32 through 95. (See
Appendix XX for a list of the character codes.) Enter these lines:

```
NEW
10 CALL CLEAR
20 CHAR=32
30 FOR ROW=7 TO 14
40 FOR COLUMN=13 TO 20
50 CALL HCHAR(ROW,COLUMN,CHA
R)
60 CHAR=CHAR+1
70 NEXT COLUMN
80 NEXT ROW
```

There are several things we'd like to point out about this program. First,
FOR-NEXT loops do not have to start counting at 1. They can begin with
whatever numeric value you need to use. Second, the nested loop (FOR
COLUMN-NEXT COLUMN) is not just a time-delay loop. It actually controls a
part of the program repetition.

Finally, line 50 is called a wrap-around line. It has more than 28
characters, so part of it prints on another line on the screen. This is an
important point: program lines can be more than one screen-line long. In
fact, a program line, in general, can be up to four screen lines (112
characters) in length. Notice that wrap-around lines (that is, the second,
third, or fourth screen lines of a program line) are not preceded by the small
prompting symbol.

Run the program, and the sixty-four characters are printed in nice, neat rows on the screen:

```
 !"#$%&'
()*+,-./
01234567
89:;<=>?
@ABCDEFG
HIJKLMNO
PQRSTUVW
XYZ[ ]^_
```

        ** DONE **
Hi there Gary, you really shouldn't go off and leave me. I get lonely and start talking to myself.

Hold on! There are only sixty-three characters on the screen! What happened to the other one? Well, there are actually sixty-four. Look at the top line, and notice that it appears to be indented one space. That's because character 32 is a space. Even though a space doesn't print anything on the screen, it does occupy room on a line, and it is a character as far as the computer is concerned.

Error Conditions with FOR-NEXT Loops

We mentioned earlier that a nested loop must be completely contained within another loop. Were your program to include lines like these,

```
20 FOR A=1 TO 6
30 FOR X=5 TO 10
...
80 NEXT A
90 NEXT X
```

the computer would stop the program and give you this error message:

```
*CAN'T DO THAT IN 90
```

The computer can't go back inside the completed "A" loop to pick up the beginning of the "X" loop.

Another possible error condition with FOR and NEXT statements is the omission of either the FOR line or the NEXT line. For example, if you attempted to run this program,

```
10 FOR A=1 TO 5
20 PRINT A
30 END
```

the computer would respond with

```
*FOR-NEXT ERROR
```

If you encounter an error message, just list the program (type LIST and press ENTER), identify the error, and correct the problem line or lines.

Review

1. What in a program line tells the computer not to perform the line immediately?

2. What are two ways to display an existing program line for editing?

3. What punctuation is missing from the following statement?

    10 INPUT "LENGTH?" L

4. A GOTO statement often causes an "endless loop" whereby a program will not stop by itself. What are two ways to stop a program in progress from the keyboard?

5. What is the term used to describe a FOR-NEXT loop within another FOR-NEXT loop?

(Answers are on page XX. If you miss a question, go back to the appropriate section and review the information before you proceed to the next section.)

More on PRINT Separators (, ; :)

While using the PRINT statement in the Immediate Mode, we saw that a difference in spacing occurred when we used a comma, semicolon, or colon to separate numeric values in a PRINT statement. Let's take another look at this.

Spacing with Commas

Try each of the following examples. (In each, we'll assume that the screen has been cleared by typing CALL CLEAR and pressing ENTER.)

```
PRINT 1,2
 1              2

PRINT 1,2,3,4,5,6
 1              2
 3              4
 5              6
```

So far we have used only small positive integers. Let's try some simple negative numbers.

```
PRINT -1,-2
-1             -2
```

Now let's try a combination of positive and negative numbers.

```
PRINT 1,2,-3,-4
 1              2
-3             -4
```

Note that the computer always leaves a space preceding the number for the sign of the number. For positive numbers, the plus sign (+) is assumed and is not printed on the screen. For negative numbers, the computer prints a minus sign (-) before the number.

We mentioned earlier in this book that there are two print zones on the screen line. Each print zone has room for fourteen characters per line.

```
Print Zone 1  Print Zone 2
(spaces 1-14) (spaces 15-28)
```

When you use a comma to separate numeric values of variables in a PRINT statement, the computer is instructed to print only one value in each zone. Therefore, since there are only two print zones on each line, the computer can print a maximum of two values per screen line. If the PRINT statement has more than two items, the computer simply continues on the next line until all the items have been printed.

Now let's try some examples with string variables, using commas as
"separators."

```
A$="ZONE 1"

B$="ZONE 2"

PRINT A$,B$
ZONE 1          ZONE 2
```

The strings (the letters and numbers within the quotation marks) are printed
in different zones on the screen when a comma is used to separate the string
variables.

Try this example:

```
A$="ONE"

B$="TWO"

C$="THREE"

D$="FOUR"

PRINT A$,B$,C$,D$
ONE             TWO
THREE           FOUR
```

(Note that for strings, the computer does not leave a preceding space.)

Spacing with Semicolons

Now let's look at semicolon spacing. Try these examples:

```
PRINT 1;2
 1  2
```

Aha! The numbers are much closer together.

```
PRINT 1;2;3
 1  2  3
```

```
PRINT 1;2;-3;-4;5;-6;7
 1  2 -3 -4  5 -6  7
```

The semicolon instructs the computer not to leave any spaces between the
values or variables in the PRINT statement. Then why do we see spaces between
the numbers on the screen? Two reasons! First, remember that each number is
preceded by a space for its sign. Second, every number is followed by a
trailing space. The trailing space is there to guarantee a space between all
numbers, even negative ones.

If the semicolon tells the computer to leave no spaces between variables in a
PRINT statement, what happens when we use string variables rather than
numeric? Let's try some examples.

```
A$="HI THERE!"

B$="HOW ARE YOU?"

PRINT A$;B$
HI THERE!HOW ARE YOU?
```

The two strings are run together. If we want a space to appear between them,
then, we must include the space inside one of the sets of quotation marks!
For example, let's change A$. Type

```
A$="HI THERE! "
PRINT A$;B$
```

```
HI THERE! HOW ARE YOU?
```

Spacing with Colons

There is a third "separator" that can be used: the colon. The colon instructs
the computer to print the next item at the beginning of the next line. It
works the same way with both numeric and string variables. Enter these lines
as an example:

```
A=-5
B$="HELLO"
C$="MY NAME IS ALPHA"
PRINT A:B$:C$
-5
HELLO
MY NAME IS ALPHA
```

To review for a moment, then, these are the three print separators we have
used:

Punctuation mark	Operation
Comma	Prints values in different zones; maximum of two items per line.
Semicolon	Leaves no spaces between items. (The spaces that appear between numbers are results of the built-in display format for numeric quantities.)
Colon	Prints next item on following line.

Understanding the Order of Arithmetic Operations

You've been introduced before to the arithmetic powers of your computer, but it's time now take a more detailed look of some of its mathematical capabilities. For example, what is the answer to this problem:

    4+6*5=?   (Remember, * means "multiply" to the computer.)

Let's say, for example, that the answer represents an amount of money you owe a friend. Your friend argues that you owe him $50, because

    4+6=10, and
    10x5=50

You, however, don't agree. You say you only owe $34, because

    6x5=30
    4+30=34

Who is right? Why not ask your computer?

    Type PRINT 4+6*5
    and press ENTER.

The answer is 34. You win!

Order of Operations

There is a commonly accepted order in which arithmetic operations are performed, and your computer performs calculations in that order. In any problem involving addition, subtraction, multiplication, and division, the arithmetic operations are completed in this way:

    Multiplications and divisions are performed
    before additions and subtractions.

This is the method your computer used to solve the previous example. It first multiplied 6*5 and then added the result to 4, giving you a final answer of 34. Now try this example:

    PRINT 6+15/3*2-4

Before you press ENTER, let's think about the way the computer evaluates this problem. Scanning the problem from left to right, the computer solves it in this order:

    15/3=5
    5*2=10
    6+10=16
    16-4=12

Your answer, then, should be 12. Press ENTER now, and see the result:

    PRINT 6+15/3*2-4
     12

Using Parentheses to Alter the Order of Operations

Suppose, however, that we want the computer to solve the last problem like this:

    (1) Add 6 and 15.
    (2) Divide the result by 3.
    (3) Multiply that result by 2.
    (4) Subtract 4, giving a final result of 10.

We can change the built-in computational order by using parentheses. Try this:

    PRINT (6+15)/3*2-4   Press ENTER.

The answer, 10, is displayed on the screen, because the computer has completed the computation inside the parentheses first. So our new order of operations becomes:

    (1) Complete everything inside parentheses, innermost first.
    (2) Complete multiplication and division, in order from left to right.
    (3) Complete addition and subtraction, in order from left to right.

Now try this example:

    PRINT 8/2*4/2

The answer is 8, because
            8/2=4
            4*4=16
            16/2=8

But suppose we entered the problem with parentheses, like this:

    PRINT 8/(2*4)/2

This time, we get a result of .5, because the expression within the parentheses has been solved first:

            2*4=8
            8/8=1
            1/2=.5

Here's a slightly harder problem to try:

    PRINT 274+10/2*100-30

If we enter the problem just like this, we obtain an answer of 744 because

            10/2=5
            5x100=500
            274+500=774
            774-30=744

But by adding parentheses in different places we can get a variety of answers:

```
PRINT (274+10)/2*(100-30)
 9940

PRINT (274+10)/(2*100)-30
 -28.58

PRINT (274+10/2)*100-30
 27870
```

Try the following for practice:

```
PRINT 38+6-4
PRINT 38+6-4*2
PRINT (38+6-4)*2
PRINT ((38+6-4)*2)/(6+2)
```

Rearrange the parentheses in the last problem. How is the answer affected?

Understanding Scientific Notation

So far, all the examples we've tried have given results in a normal decimal display form. However, the computer displays very long numbers (more than ten digits) in a special way. Try this program:

```
NEW
10 CALL CLEAR
20 A=1000
30 FOR X=1 TO 5
40 PRINT A
50 A=A*100
60 NEXT X
```

When you run the program, the first four results are printed out in the normal form. The last result, however, looks like this:

$$1E{\dagger}11$$

We call this special form scientific notation. It's just the computer's way of handling numbers that won't fit into the normal ten-digit space allotted for numbers.

$1E{\dagger}11$ means 1×10^{11} or 100,000,000,000

As you can see, $1E{\dagger}11$ represents a very large number!

Using the INT Function

The INT function gets its name from the word integer, meaning whole number,
one that has no fractional part. Integers include zero and all of the
positive and negative numbers that have no digits after the decimal point.

The best way to learn how the INT function works is by trying it. First,
let's work a division problem that doesn't result in a whole number answer.
Type

    PRINT 16/3

and press ENTER. The answer is 5.333333333.

Now try this example:

    PRINT INT(16/3)
     5

INT kept the whole number part of the answer and threw away the digits after
the decimal point. Notice that the number or expression that the INT function
works on must be enclosed in parentheses. Try another example:

    PRINT INT(7/6)   (7/6=1.166666666)
     1   (INTeger of 7/6=1)

The answer is 1; all of the fractional part has been discarded.

How about a real-life problem? Let's say a salesclerk is giving $1.37 in
change to a customer. The customer wants as many quarters as possible. How
many quarters can be given?

```
     PRINT INT(1.37/.25)
```

The answer is 5. Five quarters can be given.

More than one INT function can be used in a PRINT statement. Here's an example:

```
     PRINT INT(1/3);INT(20/9)
      0  2
```

What would happen if you entered these values with the INT function: 8, 8.99, 8.34? Try them and see.

```
     PRINT INT(8)
      8
```

```
     PRINT INT(8.99);INT(8.34)
      8  8
```

If you use INT with a whole number (integer), you just get the same number back. In the other two examples, no matter what digits are to the right of the decimal point, the INT function "truncates" or cuts off those digits--that is, it works this way for positive numbers. What happens with negative numbers?

We'll use a program to explore INT and negative numbers. Enter these lines:

```
     NEW
     10 CALL CLEAR
     20 FOR A=1 TO 7
     30 PRINT -A/3,INT(-A/3)
     40 NEXT A
```

Now RUN the program. The screen shows these results:

```
-.3333333333            -1
-.6666666666            -1
-1                      -1
-1.333333333            -2
-1.666666666            -2
-2                      -2
-2.333333333            -3
```

So INT(X)--where X represents a number or a mathematical expression--computes the nearest integer that is less than or equal to X. Perhaps looking at a number line will help to explain.

    (number line graphic)

As you see from the number line, when X has the value -0.3, the nearest integer that is less than or equal to X is -1.

One last feature associated with INT is very useful to know. It can appear on the right side of an equals sign in an assignment statement. For example, try the next series of lines.

```
A=INT(4/3)+2
PRINT A
 3
```

In the assignment statement, INT(4/3) produces the integer result of 1. This result is added to the constant 2, yielding 3 as a final result. A is then assigned the value of 3 and printed.

Try some other experiments with INT so that you become even more familiar with how it works.

Using the RND Function and the RANDOMIZE Statement

The letters in the name RND are taken from the word RaNDom. To find out what RND does, let's try a few examples in the Immediate Mode.

Enter the NEW command, and then enter this line:

    PRINT RND

Now try entering the line again. Here's an interesting situation! Every time we use RND, we get a different number. That's exactly what RND does--it generates random numbers.

Now let's try a program that produces ten random numbers. Enter these lines:

    20 FOR LOOP=1 TO 10
    30 PRINT RND
    40 NEXT LOOP

When you've checked your program for errors, run it. A list of ten random numbers is printed on the screen. Look at the numbers closely. Are any two of the numbers identical?

You may have noticed that all the numbers generated by RND are less than one (1.0) in value. Also, there are no negative numbers. RND is preset to produce only numbers that are greater than or equal to zero and less than one ($0 \leq n < 1$).

Write down the numbers this program produced, and then run the program a second time. Check your written list against the numbers on the screen this time. Very strange! The list of numbers is the same!

This feature of the RND function is important to remember and can be very useful in certain applications. Within a program, RND produces the same sequence of random numbers each time the program is run.

UNLESS...!! Unless the BASIC statement RANDOMIZE is used in your program. Add the RANDOMIZE statement shown below to the program that is still in your computer.

```
10 RANDOMIZE
```

Clear the screen now (type CALL CLEAR; press ENTER), and list the changed program on the screen:

```
LIST
10 RANDOMIZE
20 FOR LOOP=1 TO 10
30 PRINT RND
40 NEXT LOOP
```

Run the program again, and compare the new set of numbers with your written list from the first program run. Are they different this time? They should be! Continue to experiment with the program until you feel comfortable with RND and RANDOMIZE. For example, try changing line 30 of the previous program to:

```
30 PRINT RND;RND
```

What result does this change have on the program?

If you want the program to generate more or fewer than ten random numbers, just change line 20.

Other Random Number Ranges

The program you just completed generates random numbers between 0 and 1 ($0 \leq n < 1$). Now let's examine ways to increase the range of the numbers we generate.

The RND function can be used as part of any legitimate computation. For example, 10*RND and (10*RND)+7 are both valid uses of RND in BASIC. To show what is produced when RND is used in this way, enter the following statement.

    PRINT 10*RND

What number appears on the screen? Try the same statement again. What number did you get this time?

In both these examples, you should see a decimal point followed by ten digits, or one digit to the left of the decimal point, followed by nine digits to the right of the decimal point. That's because 10*RND produces random numbers in the range of 0 to (but not including) 10. Try this:

    PRINT 100*RND

and see what is produced. This time you could get one or two digits to the left of the decimal point, in the range from 0 through 99.999....

Let's use a program to generate some random numbers in the ranges 0 to 10 and 0 to 100. Enter these lines:

    NEW
    10 RANDOMIZE
    20 FOR LOOP=1 TO 5
    30 PRINT 10*RND,100*RND
    40 NEXT LOOP

Now clear the screen and run the program. Although the numbers you generate on your screen are different, they look something like this:

```
RUN
  3.196128739        11.32761568
  6.233532821        9.502421843
  7.030941884        33.17351797
  .6689170795        86.40802154
  9.388957913        .7565322811
** DONE **
```

Study the differences between the numbers in the left print zone on the screen and those in the right print zone. Can you see that the range is greater in those on the right? Run the program again to produce other numbers.

Suppose we'd like to eliminate all digits to the right of the decimal point and produce positive random whole numbers (integers). Remember the INT function we discussed earlier? This is a job for INT!

Change the program by typing and entering this new line:

```
30 PRINT INT(10*RND),INT(100*RND)
```

When you list the program now, it looks like this:

```
LIST
10 RANDOMIZE
20 FOR LOOP=1 TO 5
30 PRINT INT(10*RND),INT(100
*RND)
40 NEXT LOOP
```

When you run the program, the screen shows two series of random whole numbers
(the numbers you generate on your screen are different):

```
RUN
  9          51
  0          14
  6          77
  5           9
  1          21
```

   ** DONE **

All the numbers on the left side of the screen have values from 0 through 9,
whereas the numbers on the right have values from 0 through 99. The INT
function throws away the digits to the right of the decimal point. The
following table summarizes what we have covered so far.

Program Instruction	Range
RND	0 through .9999...
10*RND	0 through 9.9999...
INT(10*RND)	0 through 9 (integers only)
100*RND	0 through 99.9999...
INT(100*RND)	0 through 99 (integers only)

Notice that all these ranges begin with the value of zero. In many games and
simulations, however, we need random numbers that start at some other value.
For example, to simulate the throw of one die you need a random number
generator that produces values from 1 to 6. You have seen that INT(10*RND)
gives values from 0 to 9. What does INT(6*RND) produce? Change line 30 in
the program to PRINT INT(6*RND) and run the new program.

Type:

```
30 PRINT INT(6*RND)
CALL CLEAR
RUN
 4
 1
 5
 2
 3
 .
** DONE **
```

Your screen shows a list of five random numbers ranging from 0 to 5. What
happens if we added the value 1 to each item in this list? The resulting
numbers range from 1 to 6. That's just what we need to simulate the throw of
a single die. Again, alter the program as shown below and run it.

Type:

```
30 PRINT INT(6*RND)+1
CALL CLEAR
RUN

 3
 4
 1
 6
 2

** DONE **
```

That does it! The program now in your computer is a simulation (imitation) of
throwing a single die five times.

A Two-Dice Simulation

At this point we can easily design a program to simulate the throws of two six-sided dice. Before you start, erase the old program by typing NEW. Then enter the following program:

```
5 CALL CLEAR
10 RANDOMIZE
20 INPUT "NUMBER OF ROLLS?":N
30 FOR ROLL=1 TO N
40 DIE1=INT(6*RND)+1
50 DIE2=INT(6*RND)+1
60 PRINT DIE1;DIE2,DIE1+DIE2
70 NEXT ROLL
80 PRINT
90 GOTO 20
```

This program prints out the number of "spots" on each die and the sum of the spots on both dice faces. You are asked how many rolls you wish to make at the start of the program. Run the program now and watch what happens.

First, the program prints a request for the number of rolls to make. Enter a number (5, for example) and press the ENTER key.

```
NUMBER OF ROLLS?5
 2   5        7
 6   6       12
 3   1        4
 2   3        5
 1   4        5

    NUMBER OF ROLLS?
```

The program keeps looping back to the INPUT request line. (If you want to stop the program, just press BREAK.)

Try entering different values for the number rolls. What happens if you try 30 rolls? Then make some changes to the program, if you'd like to experiment. For example, how would you alter the program to simulate the throwing of three dice? Two eight-sided dice?

Error Conditions with RND

The error messages produced by an improper usage of RND are essentially the same as the error messages we've mentioned before. Here are some examples:

Typing Errors   Error Message

10 PRINT INT(10RND)   ** INCORRECT STATEMENT IN 10
10 PRINT INT(10*RND   ** INCORRECT STATEMENT IN 10

About the only new error condition we need to mention occurs if you try to use the letters RND as a numeric variable name in an assignment statement. For example, if you type

    RND=5

the computer responds with

    *INCORRECT STATEMENT

This occurs because RND is "reserved," to be used only as a function in BASIC. A list of all the reserved words is in Book 4.

Randomized Character Placement

The following program utilizes the INT and RND functions to generate random screen positions for a character you input. First, type NEW and press ENTER to erase your old program; then enter these lines:

```
10 RANDOMIZE
20 INPUT "CHAR CODE?":CODE
30 CALL CLEAR
40 ROW=INT(24*RND)+1
50 COLUMN=INT(32*RND)+1
60 CALL VCHAR(ROW,COLUMN,CODE)
70 GOTO 40
```

We'll use the character codes 33 through 95; because character 32 is a blank space, we want to avoid entering it when the program asks for a code number.

Before running the program, look at the line-by-line description below.

Line 10   "Randomizes" the random number
                   series each time the program is run.

Line 20   Stops and asks "CHAR CODE?".
                   Assigns number you enter to the
                   variable CODE.

Line 30   Clears prompting message and input
                   character code from the screen.

Line 40   Produces random integer in range
                   of 0 through 23; adds 1 to value
                   and assigns value to variable ROW.

Line 50   Produces random integer in range of
                   0 through 31; adds 1 to value and
                   assigns value to variable COLUMN.

Line 60   Prints input character in random
                   position designated by lines 40 and 50.

Line 70   Loops back to produce new random
                   position for character.

Now clear the screen with CALL CLEAR and run the program. For this first
example, enter 42 (the character code for the asterisk) as the input for CHAR
CODE. The screen looks something like this:

```
                                *
                                        *


        *


*                                   *
                *

*           *
                    *                           *
```

To stop the program just press BREAK. Then try running the program several
times, putting in a different character code each time. See if any unusual
designs are produced.

When you've finished experimenting with different characters, let's change the
program to generate characters at random, as well as placing them randomly on
the screen. First we'll have to decide how to set the limits we want for the
character range. Here's a general procedure for setting the limits for use
with RND:

        Subtract the lower limit from the upper limit.
        Add 1.
        Multiply that result by RND.
        Find the integer (INT) of this result.
        Add the lower limit.

Now we know that we want 63 characters, with character codes ranging from 33
through 95. So our lower limit is 33, and our upper limit is 95:

        95-33=62
        62+1 =63

The number we want to multiply by RND is 63, and we must use the INT function:

    INT(63*RND)

Now check the limits established when we add our lower limit, 33:
      0+33=33 (lowest possible character code)
      62+33=95 (highest possible character code)

INT(63*RND)+33 gives us random whole numbers in the range we need. Type the following new line:

    20 CODE=INT(63*RND)+33

and press ENTER. Now clear the screen and LIST the program to review this change.

    LIST
    10 RANDOMIZE
    20 CODE=INT(63*RND)+33
    30 CALL CLEAR
    40 ROW=INT(24*RND)+1
    50 COLUMN=INT(32*RND)+1
    60 CALL VCHAR(ROW, COLUMN, COD
    E)
    70 GOTO 40

When we run the program this time, the computer generates a random character code and then print the character in random positions on the screen. (Press BREAK when you want to stop the program.) Run the program several times to see different characters.

Experiment!--By making changes in two lines, you can cause the previous program to print different random characters each time it loops. Try it! (Hint: Think about lines 30 and 70.)

Using the IF THEN Statement

The IF THEN statement causes the computer to make a decision about whether a condition is true or false. If the condition is true, the program transfers control to a different line (as in a GOTO statement). If the condition is false, the program proceeds with the next sequential statement.

All the programs we've considered so far in this book have been constructed so that they either run straight through or loop using a GOTO or a FOR-NEXT loop. The IF THEN statement provides you with the capability of making branches or "forks" in your program. A branch or fork is a point in a program where either one of two paths can be taken, just like a fork in a road.

(ILLUSTRATION):
    TO: PRINT B

                                    TO: A=5

The general form of an IF THEN statement looks like this:

    IF condition THEN line number

The condition is a mathematical relationship between two BASIC expressions. The line number is the program line to which you want the program to branch if the condition is true. If the condition is not true, then the program line following the IF THEN statement is executed. For example,

    30 IF K<10 THEN 70

The statement says: If the value of K is less than 10, then go to line 70 of the program. If K is greater than or equal to 10, then do not branch to line 70. Instead, execute the line following line 30.

Let's try a demonstration program. (Press the comma key in conjunction with either SHIFT or FCTN to enter the < sign.) Enter these lines:

```
NEW
10 CALL CLEAR
20 K=1
30 PRINT "K=";K
40 K=K+1
50 IF K<10 THEN 30
60 PRINT "OUT OF LOOP"
```

Now run the program.

```
K= 1
K= 2
K= 3
K= 4
K= 5
K= 6
K= 7
K= 8
K= 9
OUT OF LOOP

** DONE **
```

Each time the program reaches line 50, it must make a "true or false" decision. When K is less than 10, the IF condition (K<10) is true, and the program branches to line 30. When K equals 10, however, K<10 is false. The program then executes line 60 and stops.

We mentioned earlier that the condition is a mathematical relationship between two expressions. In the example you've just seen, the mathematical relationship was <, or "less than." There are a total of six relationships that can be used in the IF THEN statement:

Relationship	Mathematical Symbol	BASIC Symbol
Equal to	=	=
Less than	<	<
Greater than	>	>
Less than or equal to	≤	<=
Greater than or equal to	≥	>=
Not equal to	≠	<>

Suppose we changed line 50 in the program to this:

    50 IF K<=10 THEN 30

How is the program's performance be affected? Try it! Enter the new line, and then run the program again.

Now the program prints the value of K all the way through 10, because the new line says, "If K is less than or equal to 10, branch to line 30."

The IF THEN statement can be a powerful tool in program development. Try this program for a graphics application:

```
NEW
10 CALL CLEAR
20 K=1
30 CALL HCHAR(K,K+1,42)
40 K=K+1
50 IF K<25 THEN 30
60 K=1
70 CALL HCHAR(K,K+3,42)
80 K=K+1
90 IF K<25 THEN 70
100 GOTO 100
```

(Press BREAK to stop the program.) Can you follow this pattern to create more than two diagonal lines?

Error Conditions with IF THEN

Like most BASIC statements, the IF THEN statement is pretty particular about
its form. The main errors that can occur in using the IF THEN statement are
shown below:

      20 IFA=B THEN 200   (No space after IF)

      20 IF A=BTHEN 200   (No space in front of THEN)

      20 IF A==B THEN 200   (Invalid relational symbol combinations)

      20 IF A= THEN 200   (No expression on one side of the relational symbol)

All of the above conditions produce an error message either when entered or
during the running of the program, along with a reference to the line number
of the statement in which the error occurs.

If the line number referenced in an IF THEN statement does not exist, the
program stops and produces a message saying that the line number referenced in
the statement is not found in the program. For example (using the line
above), if 200 is not a valid line number in your program, you see this error
message:

   * BAD LINE NUMBER IN 20

Review

1. Which arithmetic operations are performed first, multiplications and divisions or additions and subtractions?

2. How can you change the normal order of arithmetic operations in a program line?

3. What does the INT function do?

4. What is the difference between RND and RANDOMIZE?

5. If the condition in an IF THEN statement is true, what happens?

What if the condition is false?

(Answers are on page XX. If you miss a question, go back to the appropriate section and review the information before you proceed to the next section.)

Answers to Review on Page 28

1. C.   A variable is a word or letter that is assigned a particular value.

2. C.   The difference between numeric variables and string variables is that numbers are assigned to numeric avariables and characters are assigned to string variables.

3. D.   The difference between NEW and CALL CLEAR is that CALL CLEAR only clears the screen and NEW both clears the screen and erases the computer's memory.

4. E.   All of the given answers are valid ways to correct errors.

5. A matched with Z.   A comma displays each item in a different print zone.
    B matched with X.   A semicolon displays items close together.
    C matched with Y.   A colon displays items on separate lines.

6. A.   The numbers within the parentheses after a CALL HCHAR or CALL VCHAR command specify (row, column, character-code, number of repetitions).

Answers to Review on Page 51

1. Line numbers

2. EDIT line-number (press ENTER)

   line-number (press UP ARROW or DOWN ARROW)

3. Colon (:)
   10 INPUT "LENGTH?":L

4. BREAK or CLEAR

5. Nested

Answers to Review on Page 77

1. Multiplications and divisions are performed first.

2. You can insert parentheses to change the normal order of arithmetic operations.

3. The INT function truncates (deletes) any fractional part of a number; thus, INT makes the number an integer.

4. RND generates the same series of numbers. RANDOMIZE causes the RND function to generate truly random numbers.

5. If the condition is true, the computer branches to the line number specified.

If the condition is false, the computer simply proceeds to the next line in succession.

Index