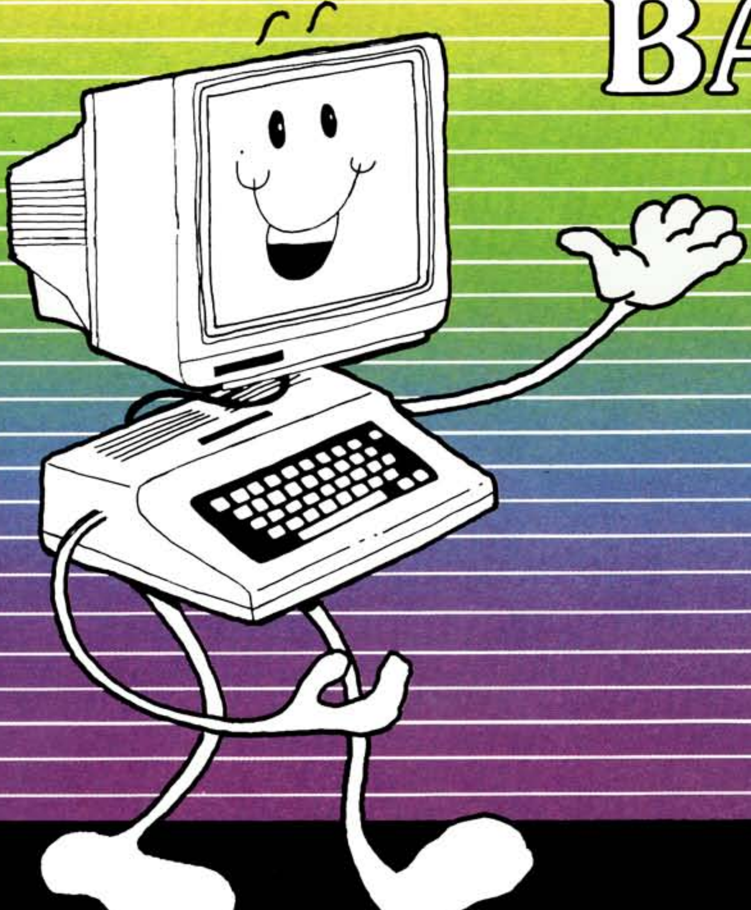
TANDY®

COLOR COMPUTER 3

EXTENDED BASIC



COLOR COMPUTER 3
EXTENDED
BASIC

10 9 8 7 6 5 4 3 2 1

Hello Newcomers . . .

If you don't know a thing about computers, relax—this manual is for you. Using it, you "program" your computer with its own language—Extended BASIC. You'll start by:

- Composing music
- Creating light shows
- Playing games
- Painting pictures

If you're eager to get down to business, be patient. Having fun is the fastest way to learn.

So, spend some time with your computer. Type whatever you want. Play with it. Feel at ease! You have an amazing tool to command.

. . . And Welcome Back Oldtimers

Welcome back to the Color BASIC family! You might already know the original Color BASIC language. You might also know the expanded Extended Color BASIC language.

The language described in this book—Extended Color BASIC Version 2—is a greater expansion of Color BASIC and Extended Color BASIC. Using Extended Color BASIC Version 2, you can:

- Draw pictures with nearly perfect detail, using hundreds and thousands of screen positions.
- Color pictures with turquoise, indigo, violet, and dozens of other exciting colors.
- Create special effects, such as mixing text with high resolution graphics.
- View text, and program in 32, 40 or 80 columns.

The table below lists the new commands provided by Extended Color BASIC Version 2.

Chapter	Command
1	PALETTE RGB
7	WIDTH, LOCATE, ATTR, SHIFT 0 (for true lowercase)
8	PALETTE, PALETTE CMP, PALETTE RGB
19	BUTTON
24	PALETTE
30	HSCREEN, HCIRCLE, HCLS, HCOLOR, HDRAW, HLINE, HPAINT, HRESET, HSET, HPOINT, HPRINT, PALETTE
31	HBUFF, HPUT, HGET
38	HSTAT
39	ERLIN, ERNO, ONERR, ONBRK

CONTENTS

PART 1 / THE BASICS

PART 2 / HAVING FUN

PART 3 / DRAMATIC IMAGES

PART 4 / THE BIG PICTURE

PART 5 / GETTING DOWN TO BUSINESS

PART 6 / BACK TO BASICS

PART 7 / ODDS AND ENDS

PART 1 / THE BASICS

In this part, you'll learn how to program. But before you start, put yourself in the right frame of mind.

- Feel comfortable. You don't have to understand **everything** at one time.
- Try out your own ideas. You don't have to do everything **our** way.
- Have fun and enjoy your color computer!

Ready? Turn the page and begin.

1 / MEET YOUR COMPUTER

This chapter and the next introduce you to your computer—the way it works, some of its talents, and even a couple of its quirks. By the time you reach Chapter 3, you'll be ready to program.

This is How to Start (Power Up)

Connect your computer to a television set or to a video monitor. Instructions on how to do this are in the introduction manual, *Introducing Your Color Computer 3*, that comes with your color computer.

Then:

1. Turn on the television set or video monitor.
2. If you use a television set, select Channel 3 or 4 and set the antenna switch to COMPUTER.
3. Turn on your computer. The POWER button is on the left rear side of your keyboard.

The BASIC startup message appears on your screen:

```
EXTENDED COLOR BASIC v.r
COPR. 1982, 1986 BY TANDY
UNDER LICENSE FROM MICROSOFT
AND MICROWARE SYSTEMS CORP.
```

(*v.r* is the number specifying which version and release of BASIC you have.)

If the BASIC startup message does **not** appear on your screen:

- Turn off your computer. Wait 30 seconds, and turn on your computer again.
- Adjust the brightness and contrast on your television set or video monitor.
- Check all the connections.

If the screen still does not show the BASIC startup message, refer to "Troubleshooting and Maintenance" in your introduction manual.

When the screen shows the BASIC startup message, you're ready to begin.

Using the Keyboard (The SHIFT 0 Keys)

Experiment with the keyboard, and type whatever you want. You are now in the *uppercase-only mode*. This means that all the characters you type appear on your screen as uppercase (capital) letters. The letters show as dark characters on a light colored screen.

Hold down the SHIFT key and press 0. Then, release both keys, and type some more characters. Now, the characters appear in reversed colors (light characters on a dark background). If the characters do **not** appear in reversed colors, press SHIFT 0 again. Be sure to hold down SHIFT before pressing 0.

By pressing (SHIFT)(@), you enter the *upper/lowercase mode*. The lowercase letters you type appear in reversed colors, and the uppercase letters you type appear in normal colors.

To type an uppercase letter, use the (SHIFT) key, just as you would on a typewriter. Hold down (SHIFT), and type the letter you want in uppercase letters.

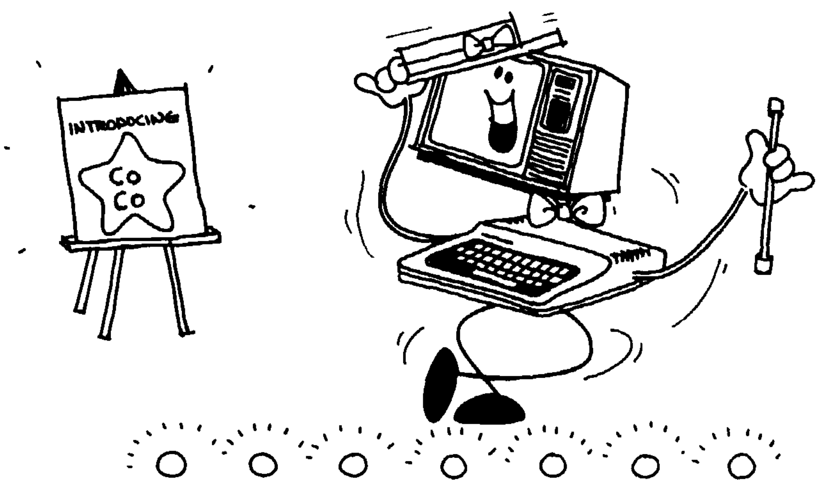
Now, return to the uppercase-only mode by pressing (SHIFT)(@) again. You will find it easier to be in the uppercase-only mode when you enter commands.

Entering a Command (The PRINT Command)

Press the (ENTER) key. Don't worry about anything but the last line of type on your screen. It says:

```
OK
```

OK is the computer's prompt. It's telling you, "OK, I'm ready when you are."



Give the computer your first command. Type this **exactly** as it is below:

```
PRINT "HI, I'M YOUR COLOR COMPUTER"
```

When you reach the right side of your screen, keep typing. The last part of the message appears on the next line.

This should be on your screen:

```
OK
PRINT "HI, I'M YOUR COLOR COMPUT
ER"
```

Now check your line. Did you put the quotation marks in correctly? Did you type the word PRINT in uppercase letters? (The computer does not understand commands typed in lowercase.)

If you made a mistake, simply press the ⊖ key, and the last character you typed disappears. Press it again, and the next to the last character disappears (and so on). Now, type the correct characters.

Ready? Press the (ENTER) key and watch. Your screen looks like this:

```
OK
PRINT "HI, I'M YOUR COLOR COMPUT
ER"
HI, I'M YOUR COLOR COMPUTER
OK
```

Your computer followed your instructions by printing the message you have in quotes on your screen.

Entering Numbers (Numbers *v* Strings)

Have the computer print another message. Type:

```
PRINT "2"
```

Press (ENTER). The computer prints your message.

Try another one:

```
PRINT "2 + 2" (ENTER)
```

The computer prints:

```
2 + 2
```

You probably expect much more than an electronic mimic . . . maybe some answers! Give your computer some numbers without the quotation marks. Type:

```
PRINT 2 + 2 (ENTER)
```

This time the computer prints the answer:

```
4
```

The quotation marks obviously have a meaning. Experiment with them some more. Type each of these lines:

```
PRINT 5+4 (ENTER)
PRINT "5+4" (ENTER)
PRINT "5+4 EQUALS" 5+4 (ENTER)
PRINT 6/2 "IS 6/2" (ENTER)
PRINT "8/2" (ENTER)
PRINT 8/2 (ENTER)
```

Any conclusions?

RULES ON STRINGS *v* NUMBERS

The computer sees everything you type as *strings* or *numbers*. If it's in quotes, it's a string. The computer sees it **exactly** as it is. If it's not in quotes, it's a number. The computer calculates it as a numerical problem.

A Color Calculator (+, −, /, *, and (↑))

Any arithmetic problem is a snap for the computer. Do some long division. Type:

```
PRINT "3862 DIVIDED BY 13.2 IS" 3862/13.2 (ENTER)
```

Do a multiplication problem:

```
PRINT 1589 * 23 (ENTER)
```

Notice that the computer uses an asterisk (*) for multiplication.

You can raise a number to a power by using the (↑) key. To print 3 to the power of 2, type:

```
PRINT 3 (↑) 2 (ENTER)
```

Try a few more problems:

```
PRINT "15 * 2 = " 15*2 (ENTER)
PRINT 18 * 18 "IS THE SQUARE OF 18" (ENTER)
PRINT 33.33/22.82 (ENTER)
```

Now it's your turn. Write two commands that print these two problems and their answers:

157 / 13.2 =
95 * 43 =

DO-IT-YOURSELF COMMANDS

____

____

____

____

____

If you use the correct commands, this is what the computer prints on your screen:

```
157 / 13.2 = 11.8939394
95 * 43 = 4085
```

Ready for the answers? The correct commands are:

```
PRINT "157 / 13.2 = " 157/13.2
PRINT "95 * 43 =" 95*43
```

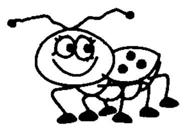
Making Mistakes (Error Messages)

Type this line, deliberately misspelling the word PRINT as "PRIINT":

```
PRIINT "HI" (ENTER)
```

The screen shows:

```
?SN ERROR
```



?SN ERROR stands for *syntax* error. This is the computer's way of saying, "PRIINT is not in my vocabulary. I have no idea what you want me to do." Any time you get the ?SN error, you probably made a typing error.

The computer also gives you error messages when it **does** understand what you want it to do, but it feels you're asking it to do something that is illogical or impossible. For instance, try this:

```
PRINT 5/Ø (ENTER)
```

The screen shows:

```
?/Ø ERROR
```

which means, "I can't divide by 0—that's impossible!"

If you get an error message you don't understand, flip to the Appendix. We've listed all the error messages there and what probably caused them.

A Screen of a Different Color (The CLS Command)

So far, all you've seen your computer do is display characters on a green screen. But your color computer has other colors too. Type:

```
CLS8 (ENTER)
```

Now, your screen is orange with a green stripe at the top. Your command told the computer to clear the screen and display Color 8—orange.

But why the green stripe? The computer must use the current background color whenever it displays characters. Later, you'll learn how to change the background color, but for now, the background color is green. Type some more characters. The computer uses a green background for them also.

Press (ENTER) to get the OK prompt. Now type:

```
CLS4 (ENTER)
```

You see a green stripe at the top, as before, and the rest of your screen is one of two colors.

- If you have a television set or a composite monitor, the rest of your screen is red. (Throughout this book, we refer to these displays as CMP monitors.)
- If you have an *RGB* monitor, the rest of your screen is black.

Some colors, such as Color 4, look different on a CMP monitor than they look on an RGB monitor.

You can produce nine colors with the CLS command. To see them, enter CLS with any number in the range 0 to 8. (If you enter a number outside the range 0 to 8, the screen shows the error message **MICROSOFT**.)

Now, try CLS without a number:

```
CLS (ENTER)
```

When you don't use a number, the computer assumes you want to display the current background color which, at this point, is green.

The Standard Colors (The PALETTE Command)

The nine colors that you can produce with the CLS command on a CMP monitor are the color computer's *standard colors*. In most of this book, we use the standard colors.

The standard colors are:

CLS Number	Standard Color
0	Black
1	Green
2	Yellow
3	Blue
4	Red
5	Buff
6	Cyan
7	Magenta
8	Orange

If you have an RGB monitor, you can produce the standard colors by entering the PALETTE RGB command. Type:

```
PALETTE RGB (ENTER)
```

Now, you can use the CLS command to produce the standard colors on your RGB monitor. For example, type:

```
CLS4 (ENTER)
```

Before you entered the PALETTE RGB command, CLS4 made your screen black. Now, CLS4 makes your screen red.

From this point on, remember: If you have an RGB monitor and want to produce the standard colors, you must enter the PALETTE RGB command **each** time you turn on your computer.

Computer Sound Off—One, Two... (The SOUND Command)

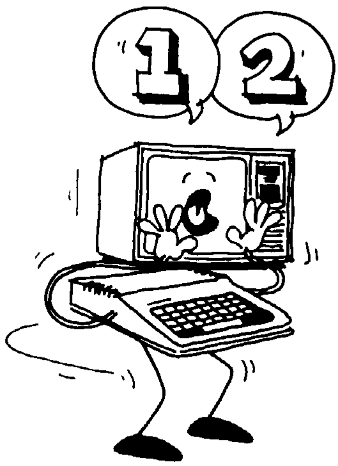
Type this:

```
SOUND 1,1ØØ (ENTER)
```

If you don't hear anything, turn up the volume and try again.

What you're hearing is six seconds of the lowest tone the computer can hum. How about the highest tone? Type:

```
SOUND 255,1ØØ (ENTER)
```



The second number tells the computer how long to hum the tone. You can use any number in the range 1 to 255. Try 1:

```
SOUND 128,1 (ENTER)
```

The computer hums the tone for about 6/100ths of a second. Try 10:

```
SOUND 128,1Ø (ENTER)
```

The computer sounds the tone for 6/10ths of a second.

Try different number combinations, but keep each number in the range 1 to 255. (If you enter a number outside the range 1 to 255, the screen shows error message ?FC ERROR.)

Say It in Its Own Words (Command Syntaxes)

In this chapter, you have learned about four commands—PRINT, SOUND, CLS, and PALETTE. In learning about these commands, you have learned the formats or the *syntaxes* that you need to use when you enter them.

Later, you might forget the syntax for a certain command. For this reason, each time we introduce a new command, we include the command's syntax. The syntax is in a box so it is easy to find.

In many of the syntaxes, we use italicized words or abbreviations to represent information that you need to provide. For example, in the PRINT syntax, *message* represents an actual message (such as "HI, I'M YOUR COLOR COMPUTER") that you need to provide.

The syntax for PRINT is:

> **PRINT** *message* Prints *message* on the display. The message can be a string (in which case BASIC prints the string exactly as it is), or a number (in which case BASIC prints the number's value).

The syntax for CLS is:

> **CLS** *c* Clears your text screen and displays Color *c* on it. *c* is a number from 0 to 8 representing the color you want displayed. If you omit *c*, BASIC displays the current background color.

The syntax for SOUND is:

> **SOUND** *n1*, *n2* Sounds the specified tone (*n1*) for a specified period of time (*n2*). *n1* is a number in the range 1-255. n2 is a number in the range 1-255.

The syntax for PALETTE is:

> **PALETTE RGB** Sets the computer to display the standard colors on an RGB monitor.

Learned in Chapter 1

COMMANDS	KEYBOARD CHARACTERS	CONCEPTS
PRINT	←	string *v* numbers
SOUND	ENTER	error messages
CLS	SHIFT 0	syntax
PALETTE		

A refresher like this is at the end of each chapter. It helps you make sure you didn't miss anything.

2 / YOUR COMPUTER NEVER FORGETS (... unless you turn it off ...)

One skill that makes your computer so powerful is its *memory*. In this chapter, you learn how to get the computer to remember any information you want.

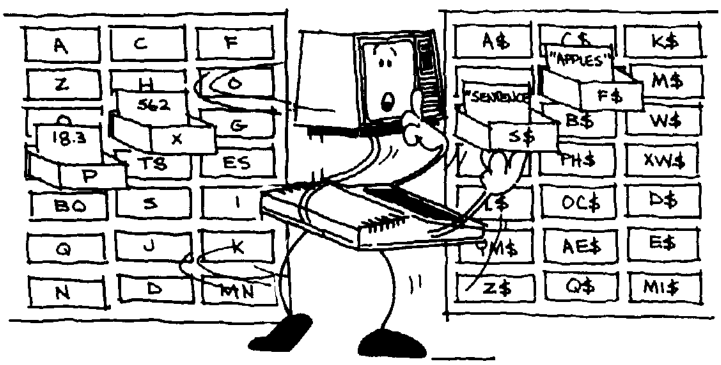
This Is How It Works... (The LET Command)

Have the computer "remember" the number 13. Type:

```
A = 13 (ENTER)
```

Now, type whatever you want. When you're done, press (ENTER). See if the computer remembers what A means by typing:

```
PRINT A (ENTER)
```



Your computer remembers that A is 13 as long as you have it on . . . or until you do this. Type:

```
A = 17.2 (ENTER)
```

Now, if you ask it to **PRINT A**, it prints 17.2.

You don't have to use the letter A. You can use any letter from A to Z. In fact, you can use any two letters from A to Z. Type:

```
B = 15 (ENTER)
C = 20 (ENTER)
BC = 25 (ENTER)
```

Now, have the computer print all the numbers you asked it to remember. Type:

```
PRINT A, B, C, BC (ENTER)
```

If you want the computer to remember a "string" of letters or numbers, use a letter with a dollar sign ($). Type:

```
A$ = "TRY TO"
B$ = "REMEMBER"
C$ = "THIS, YOU"
BC$ = "GREAT COMPUTER"
```

Then type:

```
PRINT A$, B$, C$, BC$ (ENTER)
```

Computer programmers have a name for all the letters you used: *variables*. So far, you used these variables:

YOUR COMPUTER'S MEMORY

NUMBERS	CHARACTERS
A = 17.2	A$ = "TRY TO"
B = 15	B$ = "REMEMBER"
C = 20	C$ = "THIS, YOU"
BC = 25	BC$ = "GREAT COMPUTER"

Spot-check the above variables to see if the computer remembers the right information. For instance, to see if BC still contains 25, type:

```
PRINT BC (ENTER)
```

Think of variables as little boxes in which you can store information. One set of boxes is for strings; the other set is for numbers. Each box has a label.

You can store information in a variable or change the information that is already stored in an existing variable by entering a simple command such as A=5. This simple command is actually called the LET command, but the color computer lets you omit the word LET.

The syntax for the LET command is:

LET *variable = value* Assigns a value to a variable. You can omit the word LET and simply type *variable = value*.

Note: Some versions of BASIC require that you include the word LET. With the color computer, the word LET is optional.

Rules, Rules, Rules . . . (The TM Error)

Whenever you store data into variables, you need to make sure you follow these rules.

RULES ON STORING DATA INTO VARIABLES

- Data in quotes is *string data*. You can store string data only in *string variables* (variables with a $ sign).
- Data not in quotes is *numeric data*. You can store numeric data only in *numeric variables* (variables without a $ sign).

As an example of what happens when you disobey these rules, type these four commands:

```
D = "6" (ENTER)
Z = "THIS IS STRING DATA" (ENTER)
D$ = 6 (ENTER)
Z$ = 12 (ENTER)
```

The computer responds to each of these commands with `?TM ERROR` (Type-Mismatch Error). This is because each of them attempts to store data into the wrong kind of variable.

- The first two commands attempt to store **string** data into **numeric** variables.
- The second two commands attempt to store **numeric** data into **string** variables.

Type these commands, which the computer accepts:

```
D$ = "6" (ENTER)
Z$ = "THIS IS STRING DATA" (ENTER)
D = 6 (ENTER)
Z = 12 (ENTER)
```

You've now added this to your computer's memory.

YOUR COMPUTER'S MEMORY

NUMBERS	STRINGS
D→6	D$→"6"
Z→12	Z$→"THIS IS STRING DATA"

A New Kind of Arithmetic (Using Numeric Variables)

Now, do something interesting with what you told the computer to remember. Type:

```
PRINT D * 2 (ENTER)
```

The computer prints the product of D times 2. (The computer remembers that D equals 6.)

Try this line:

```
PRINT Z/D
```

The computer prints the quotient of Z divided by D.

Would this work?

```
PRINT D$ * 2 (ENTER)
```

Did you try it? You see `?TM ERROR`. The computer cannot multiply string data.

Cross out the commands below that the computer rejects:

EXERCISE WITH VARIABLES

F = 22.9999999
M = "19.2"
DZ$ = "REMEMBER THIS FOR ME"
M$ = 15
Z = F + F

Finished? These are the commands the computer **accepts**.

F = 22.9999999
DZ$ = "REMEMBER THIS FOR ME"
Z = F + F

RULES ON VARIABLES

You can use any two characters from A to Z for a variable. The first character must be a letter from A to Z. The second can be a letter or a number. If you want to assign string data to the variable, put a dollar sign after the variable. Otherwise, the variable can hold only numeric data. String variables can store up to 249 characters.

Learned in Chapter 2

CONCEPTS

Variables
String *v* Numeric Variables

3 / A SIMPLE PROGRAM

You've learned some commands. Now, all you need to do is combine them into a program.

A Simple 1-Line Program (The NEW and RUN Commands)

Type:

```
NEW (ENTER)
```

This command erases whatever might be in the computer's memory.

Now type this line. Be sure you type the number 10 first—that's important.

```
10  PRINT "HI, I'M YOUR COLOR COMPUTER" (ENTER)
```

Did you press (ENTER)? Nothing happened, did it? Nothing you can see, that is. You just wrote your first program. Type:

```
RUN (ENTER)
```

The computer runs your program. Type RUN again and again to your heart's content. The computer runs your program any time you wish, as many times as you wish.

In your first program, you used two new commands: NEW and RUN. Their syntaxes are:

NEW Clears memory.

RUN *line numbers* Runs the specified line numbers. *line numbers* is optional; if omitted, the entire program runs.

And Now, A 2-Line Program (The LIST Command)

Your first program works well; so add another line to it. Type:

```
20 PRINT "WHAT IS YOUR NAME?" (ENTER)
```

If you make a mistake in typing this line, or any other line, simply type the line over again.

Now, type:

```
LIST (ENTER)
```

The computer displays the entire program. Your screen shows:

```
10 PRINT "HI, I'M YOUR COLOR COMPUTER"
20 PRINT "WHAT IS YOUR NAME?"
```

The command that you used to display the program is the LIST command. Its syntax is:

LIST *line numbers* Displays the specified *line numbers*. *Line numbers* is optional; if omitted, the entire program displays.

Entering Your Name (The INPUT Command)

Run the program. Type:

```
RUN (ENTER)
```

The computer displays:

```
HI, I'M YOUR COLOR COMPUTER
WHAT IS YOUR NAME?
```

What do you suppose would happen if you answer the computer's question? Try it.

When you simply type your name, the computer doesn't understand what you mean. In fact, the computer can only understand what you mean when you talk to it in its own language.

Use a word the computer understands: INPUT. The syntax for INPUT is:

INPUT "*message*"; *variable* Prints your message; then, waits for you to input information and labels that information as *variable*. *Message* is optional. If you use "*message*", remember to use a semicolon after the "*message*".

Change Line 20 so it uses the word INPUT rather than PRINT. How do you change a program line? Simply type it again using the same line number. Type:

```
20 INPUT "WHAT IS YOUR NAME"; A$ (ENTER)
```

This tells the computer to:

- PRINT "WHAT IS YOUR NAME".
- Wait for you to type some characters and press (ENTER).
- Label the characters you type as A$.

Add one more line to the program:

```
30 PRINT "HI," A$ (ENTER)
```

Now, list the program again to see if yours looks like ours. Type:

```
LIST (ENTER)
```

The program looks like this:

```
10 PRINT "HI, I'M YOUR COLOR COMPUTER"
20 INPUT "WHAT IS YOUR NAME?"; A$
30 PRINT "HI," A$
```

Can you guess what will happen when you run it? Try it:

```
RUN (ENTER)
```

That worked well, didn't it? This is probably what happened when you ran the program (depending on what you typed as your name):

```
HI, I'M YOUR COLOR COMPUTER
WHAT IS YOUR NAME? JANE
HI, JANE
```

RUN the program again using different names. For example:

```
HI, I'M YOUR COLOR COMPUTER
WHAT IS YOUR NAME? HUGO
HI, HUGO
```

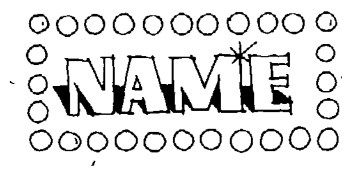
Again and Again (The GOTO Command)

By using another new command—called GOTO—you can have the computer run the same commands over and over. GOTO's syntax is:

`GOTO` *line number* Goes to *line number*.

Type this line:

```
40 GOTO 30
```



Now, run the program. The computer prints your name again and again without stopping. GOTO tells the computer to go back to Line 30:

```
10 PRINT "HI, I'M YOUR COLOR COMPUTER"
20 INPUT "WHAT IS YOUR NAME?"; A$
30 PRINT "HI," A$
40 GOTO 30
```

Your program now runs perpetually. Each time it gets to Line 40, it goes back to Line 30. We call this a *loop*. You can stop this endless loop in two ways:

- Hold down (SHIFT) and press (@). This pauses the program. Press any key to continue the program.
- Press (BREAK). This ends the program.

Print Spacing (The PRINT, and PRINT; Commands)

Press (BREAK) to end the program.

You can make a big change simply by adding a comma or a semicolon to the PRINT command. Try the comma first. Type Line 30 again, but with a comma at the end:

```
30  PRINT "HI," A$,
```

Run the program. The computer displays everything in two columns.

Press (BREAK) and try the semicolon. Type:

```
30  PRINT "HI," A$;
```

Now, run the program. You probably won't be able to tell what the program's doing until you press (BREAK). See how the semicolon crams everything together?

RULES ON PRINT PUNCTUATION

This is what punctuation at the end of a PRINT line does:

- A comma makes the computer go to the next column. Use it to print in columns.
- A semicolon makes the computer stay where it is. Use it to pack what you print together.
- No punctuation makes the computer go to the next line. Use it to print in rows.

A Compact Program (The Colon (:))

By now, you might assume that each new command you add to a program must begin on a new line. Actually, you can combine several commands into one line using a colon (:) to separate them.

For example, you could combine all four commands in the above program into one line. First, change Line 10 so that it includes all four commands. Type:

```
10  PRINT "HI, I'M YOUR COLOR COMPUTER":INPUT "WHAT IS
    YOUR NAME?"; A$:PRINT "HI," A$; (ENTER)
```

Then, delete Lines 20, 30, and 40, by typing:

```
2Ø (ENTER)
3Ø (ENTER)
4Ø (ENTER)
```

Now, run the program. It should work the same way it did when it was three lines.

Combining commands conserves memory and is useful when you write a long program that requires a large amount of memory. The problem with combining commands is that it makes a program more difficult to read and understand. We want all the programs in this manual to be easy to understand; so in most of this manual, we won't combine commands.

RULES ON ENTERING PROGRAM LINES

- A program line consists of one or more commands, separated by colons (:).
- A program line can contain as many commands as you want to include, providing that the entire line has 249 characters or less.

Changing The Program (Inserting, Deleting, and Changing Program Lines)

You may not realize it, but you now know three ways to change a program. Here's a summary:

- You can **insert** a program line by entering the line. BASIC will automatically insert the line at the correct place in the program. All line numbers go in ascending numeric sequence. For example, to insert Line 20 between lines 15 and 22, type:

```
2Ø PRINT "IT'S EASY TO CHANGE A PROGRAM"
(ENTER)
```

- You can **change** a program line by entering the line over again. For example, to change Line 20, type:

```
2Ø PRINT "SIMPLY TYPE THE LINE OVER AGAIN"
(ENTER)
```

- You can **delete** a program line by entering only the line number. For example, to delete Line 20, type:

```
2Ø (ENTER)
```

Color/Sound Demonstration (An Example of a Program)

Want to play with color and sound some more? First, erase memory. Type:

```
NEW (ENTER)
```

Then, enter this program:

```
10  PRINT "TO MAKE ME CHANGE MY TONE"
20  INPUT "ENTER A NUMBER FROM 1 TO 255"; T
30  SOUND T, 50
40  GOTO 10
```

Run the program to get a sample of the computer's tones.

What happens if you change Line 30 to:

```
30  SOUND 50, T
```

Hint: Look back in Chapter 1 where we talk about SOUND.

. . . .

Know the answer? If you make the above change, the computer hums the same tone each time, but for a different length of time.

DO-IT-YOURSELF PROGRAM

First, press (BREAK), then erase this program by typing NEW. Now see if you can write a program, similar to the one above, to make the computer show a certain color. Remember, you can display nine colors with the CLS command, 0 through 8.

This is our program:

```
10  PRINT "TO MAKE ME CHANGE MY COLOR"
20  INPUT "TYPE A NUMBER BETWEEN 0 AND 8"; T
30  CLST
40  GOTO 10
```

Add Polish to the Program (The IF/THEN Command)

Pressing the (BREAK) key is an abrupt way to stop the program. Why not have the computer politely ask if you're ready to end?

Press (BREAK) to stop the program. Then change Line 40 to:

```
40  INPUT "DO YOU WANT TO SEE ANOTHER COLOR?"; R$
```

Then, add this line:

```
50  IF R$ = "YES" THEN 20
```

Run the program. Type YES and the program keeps running. Type anything else and the program ends.

This is what the program looks like now:

```
1Ø  PRINT "TO MAKE ME CHANGE COLORS"
2Ø  INPUT "TYPE A NUMBER BETWEEN Ø AND 8"; T
3Ø  CLST
4Ø  INPUT "DO YOU WANT TO SEE ANOTHER COLOR"; R$
5Ø  IF R$ = "YES" THEN 2Ø
```

This is what the new lines do:

- Line 40 prints a question and tells the computer to stop and wait for an answer: R$.
- Line 50 tells the computer to go back to Line 20 **If** (and only if) your answer (R$) is "yes." If not, the program ends, because it has no more lines.

Learned in Chapter 3

COMMANDS	CONCEPT	KEYBOARD
NEW	Change lines	(BREAK)
RUN	Insert lines	(SHIFT)(@)
LIST	Delete lines	(:)
INPUT	Combine commands	
GOTO	Create loops	
PRINT,		
PRINT;		
IF/THEN		

4 / A LOOP

In this chapter you experiment with computer sound effects. First, you need to use two new commands, FOR and NEXT, to teach the computer to count.

How to Count (The FOR and NEXT Commands)

FOR and NEXT are two commands, but they are always used together. Their syntaxes are:

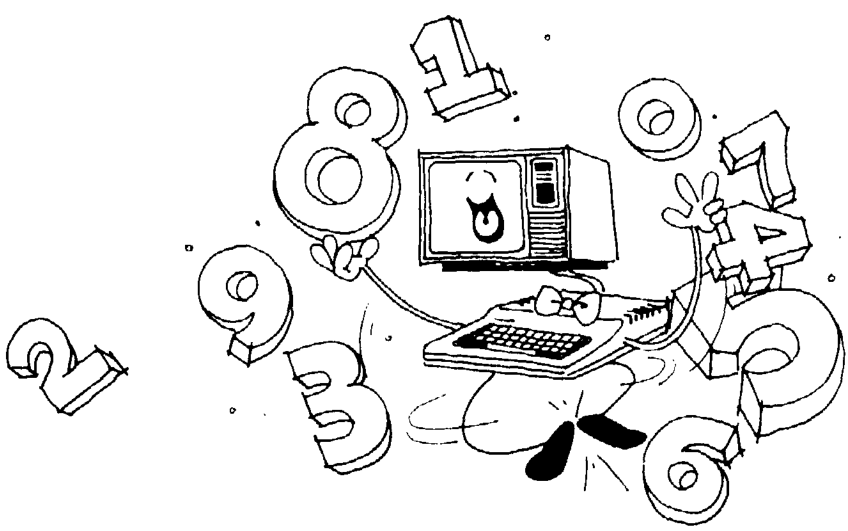
FOR *variable*=*n1* TO *n2* STEP *n3* Stores *n1* in *variable* and, each time the computer loops back to FOR, adds *n3* to *variable*. STEP *n3* is optional; if omitted, the computer uses STEP 1.

NEXT *variable* If *variable* is less than or equal to *n2*, loops back to the corresponding FOR command. Otherwise, BASIC proceeds to the next command.

These syntaxes might make FOR and NEXT sound complicated, but they are actually simple to use. Type in NEW to erase memory, then type:

```
10  FOR X = 1 TO 10
20  PRINT "X = " X
30  NEXT X
40  PRINT "I HAVE FINISHED COUNTING"
```

Run the program.



Before trying to figure out what FOR and NEXT do, replace Line 10 with each of the lines below and run the program four more times.

```
10  FOR X = 1 TO 100
10  FOR X = 5 TO 15
10  FOR X = -2 TO 2
10  FOR X = 20 TO 24
```

FOR and NEXT make the computer count. Look at the last version of the program:

```
10  FOR X = 20 TO 24
20  PRINT "X = " X
30  NEXT X
40  PRINT "I HAVE FINISHED COUNTING"
```

Line 10 tells the computer the first number is 20 and the last number is 24. It uses X to label all these numbers.

Line 30 tells the computer to keep going back to Line 10 for the next number (the NEXT X) until it reaches the last number (number 24).

Look at Line 20. Since Line 20 is between the FOR and NEXT lines, the computer prints the value of X each time it counts:

```
X = 20
X = 21
X = 22
X = 23
X = 24
```

Add another line between FOR and NEXT:

```
15 PRINT "... COUNTING ...''
```

Run the program. Your computer executes any lines you choose to insert between FOR and NEXT.

DO-IT-YOURSELF PROGRAM 4-1

Write a program that makes the computer print your name 10 times.

Hint: The program must count to 10.

DO-IT-YOURSELF PROGRAM 4-2

Write a program to print the multiplication tables for 9 (9*1 through 9*10).

Hint: PRINT 9*X is a perfectly legitimate command.

DO-IT-YOURSELF PROGRAM 4-3

Write a program that prints the multiplication tables for 9*1 through 9*25.

Hint: By adding a comma in the PRINT line, you can get all the problems and results on your screen at once.

Finished? These are our programs:

Program 4-1

```
10  FOR X = 1 TO 10
20  PRINT "THOMAS"
30  NEXT X
```

Program 4-2

```
10  FOR X = 1 TO 10
20  PRINT "9*"X"="9*X
30  NEXT X
```

Program 4-3

```
10  FOR X = 1 TO 25
20  PRINT "9*"X"="9*X,
30  NEXT X
```

Counting by Two's (The STEP Option)

Now, have the computer count in a slightly different way. Erase your program by typing **NEW** (ENTER). Then, type this new program:

```
10  FOR X = 2 TO 10 STEP 2
20  PRINT "X= " X
30  NEXT X
40  PRINT "I HAVE FINISHED COUNTING"
```

Run the program. Do you see what STEP 2 does? It makes the computer count by 2's. Line 10 tells the computer that:

- The first X is 2
- The last X is 10

  ... AND STEP 2 ...

- All the X's between 2 and 10 are 2 apart (2, 4, 6, 8, 10). STEP 2 tells the computer to add two to get each NEXT X.

To make the computer count by 3's, make all the X's 3 apart. Try this for Line 10:

```
10  FOR X = 3 TO 10 STEP 3
```

Run the program. Your screen shows:

```
X = 3
X = 6
X = 9
```

Here are more FOR ... STEP lines if you want some more practice:

```
10  FOR X = 5 TO 50 STEP 5
10  FOR X = 10 TO 1 STEP -1
10  FOR X = 1 TO 20 STEP 4
```

You may be wondering about the programs you ran at the first of this chapter without using STEP. If you omit STEP, the computer assumes you mean STEP 1.

Counting the Sounds (An Example of FOR/NEXT)

Now that you taught the computer to count, you can add some sound. Erase your old program, and type this:

```
10 FOR X = 1 TO 255
20 PRINT "TONE " X
30 SOUND X, 1
40 NEXT X
```

This program makes the computer count from 1 to 255. Each time it counts a new number, it does what Lines 20 and 30 tell it to do:

- Line 20 prints X, the current count.
- Line 30 sounds X's tone.

For example:

- The first time the computer gets to FOR, in Line 10, it makes X equal to 1.
- Then, it goes to Line 20 and prints 1, the value of X.
- Line 30 has it sound tone #1.
- Then, it goes back to Line 10 and makes X equal to 2.
- It repeats this process until X=255 or you press (BREAK).

What do you think the computer will do if you make this change to Line 10:

```
10  FOR X = 255 TO 1 STEP -1
```

PROGRAMMING EXERCISE

Using STEP, change Line 10 so the computer sounds tones from:

1 The bottom of its range to the top, humming every tenth note.

2 The top of its range to the bottom, humming every tenth note.

3 The middle of its range to the top, humming every fifth note.

10 ______________________________

10 ______________________________

10 ______________________________

Ready for the answers?

```
10  FOR X = 1 TO 255 STEP 10
10  FOR X = 255 TO 1 STEP -10
10  FOR X = 128 TO 255 STEP 5
```

DO-IT-YOURSELF PROGRAM 4-4

Now, see if you can write a program that makes the computer hum:

1 from the bottom of its range to the top, and then
2 from the top of its range back to the bottom

The answer to this, and the remaining "Do-It-Yourself" programs are in the back of this book.

But Can It Sing?

Yes. In Chapter 11, you'll learn how to have the computer play your favorite songs.

Learned in Chapter 4

COMMANDS

FOR...TO...STEP
NEXT

5 / A LOOP WITHIN A LOOP

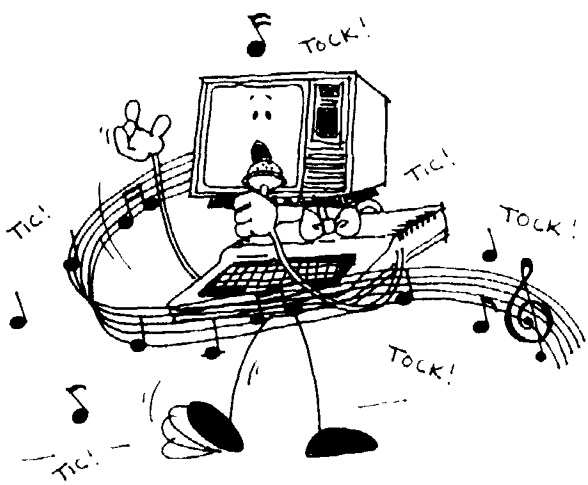
Now that you know how to use FOR/NEXT, you can have the computer keep time. First, use FOR/NEXT to set a *timer pause*. Then, use a *nested* FOR/NEXT to build a clock.

Setting a Timer Pause (A Practical Use of FOR/NEXT)

Type:

```
10  FOR Z = 1 TO 460 * 2
20  NEXT Z
30  PRINT "I COUNTED TO 920"
```

Run the program. Be patient and wait a few seconds. Two seconds, to be precise. It takes the computer two seconds to count to 920.



Lines 10 and 20 set a *timer pause*. By making the computer count to 920, you keep the computer busy for two seconds.

This is groundwork for a stopwatch. Erase the program, and type:

```
10  PRINT "HOW MANY SECONDS?"
20  INPUT S
30  FOR Z = 1 TO 460*S
40  NEXT Z
50  PRINT S " SECONDS ARE UP!!!"
```

Run the program. Enter the number of seconds you want timed.

DO-IT-YOURSELF PROGRAM 5-1

It would help if the stopwatch could sound some kind of alarm. Add lines to the end of the program to give it an alarm.

A Loop Within a Loop (Nested FOR/NEXT Commands)

Before making an actual clock, you need to know how to use a FOR/NEXT loop **within** a FOR/NEXT loop.

Type this new program:

```
1Ø  FOR X = 1 TO 3
2Ø  PRINT "X = " X
3Ø  FOR Y = 1 TO 2
4Ø  PRINT, "Y=" Y
5Ø  NEXT Y
6Ø  NEXT X
```

Run it. This should be on your screen:

```
X = 1
                    Y = 1
                    Y = 2
X = 2
                    Y = 1
                    Y = 2
X = 3
                    Y = 1
                    Y = 2
```

Programmers call this a *nested loop*. This is what the program does:

I. It counts X from 1 to 3. Each time it counts X:

   A. It prints the value of X

   B. It counts Y from 1 to 2. Each time it counts Y, it prints the value of Y.

   C. When it finishes counting the Y's, it goes back to count the next X (Line 10).

II. When it finishes counting the X's, the program ends.

As an alternate way of writing this program, you can combine Lines 50 and 60 into one line.

```
10  FOR X = 1 TO 3
20  PRINT "X = " X
30  FOR Y = 1 TO 2
40  PRINT, "Y =" Y
50  NEXT Y, X
```

Line 50 tells the computer to go back for the next Y, then, when it finishes counting all the Y's, go back for the next X.

Regardless of which way you write the program, whenever you use nested loops, be sure to close the inner loop before closing the outer loop.

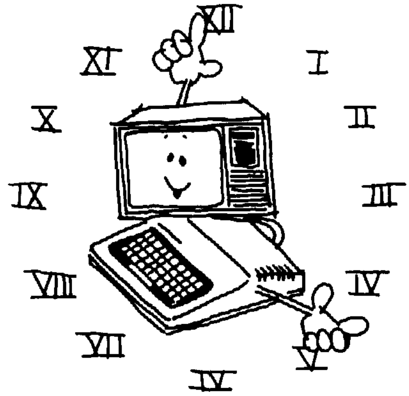
Right	Wrong
10 FOR X = 1 TO 3	10 FOR X = 1 TO 3
20 FOR Y = 1 TO 2	20 FOR Y = 1 TO 2
30 NEXT Y	30 NEXT X
40 NEXT X	40 NEXT Y

Right	Wrong
10 FOR X = 1 TO 3	10 FOR X = 1 TO 3
20 FOR Y = 1 TO 2	20 FOR Y = 1 TO 2
30 NEXT Y, X	30 NEXT X, Y

A Computer Clock (Example of Nested FOR/NEXT Commands)

This example shows how to use nested FOR/NEXT commands to make a computer clock. Type:

```
10  FOR S = 0 TO 59
15  CLS
20  PRINT S
30  SOUND 150, 2
40  FOR T = 1 TO 390
50  NEXT T
60  NEXT S
70  PRINT "1 MINUTE IS UP"
```



Run the program. This is what the program does:

I. It counts the seconds from 0 to 59 (Lines 10 and 60). Each time it counts one second.

   A. It clears the screen (Line 15).

   B. It prints the second (Line 20).

   C. It sounds a tone (Line 30).

   D. It pauses long enough for one second to pass (Lines 40 and 50).

II. When it finishes counting all the seconds from 0 to 59, it prints a message that one minute has passed (Line 70).

This is a full-fledged clock:

```
10  FOR H = 0 TO 23
20  FOR M = 0 TO 59
30  FOR S = 0 TO 59
40  CLS
50  PRINT H":"M":"S
60  SOUND 150, 2
70  FOR T = 1 TO 375
80  NEXT T
90  NEXT S
100  NEXT M
110  NEXT H
```

When you run this program, the computer does this:

I. It counts the hours from 0 to 23 (Line 10). Each time it counts a new hour:

   A. It counts the minutes from 0 to 59 (Line 20). Each time it counts a new minute:

      1. It counts the seconds from 0 to 59 (Lines 30 and 90). Each time it counts a new second:

         a. It clears the screen (Line 40).

         b. It prints the hour, minute, and second (Line 50).

         c. It sounds a tone (Line 60).

         d. It pauses long enough for one second to pass (Lines 70 and 80).

      2. When it finishes counting all the 59 seconds, it goes back to Line 20 for the next minute (Line 100).

   B. When it finishes counting all the 59 minutes, it goes back to Line 10 for the next hour (Line 110).

II. When it finishes counting all the hours (0-23), the program ends. (By adding another line, 120 GOTO 10, the clock runs perpetually.)

DO-IT-YOURSELF PROGRAM 5-2

Between Lines 90 and 100 you can add some tones to sound each minute. Write a program with the added tones.

DO-IT-YOURSELF PROGRAM 5-3

Write a program that makes your computer show each of its nine colors for one second each.

Listing Long Programs
(The LIST Command and the (SHIFT)(@) Keys)

Your programs are now getting so long that you need a better way of listing them than simply typing LIST (ENTER). Try these two methods:

- Specify only those lines that you want to see with the LIST command. For example, to list only Lines 50-100, type:

  ```
  LIST 50-100 (ENTER)
  ```

- Type LIST (ENTER). Then, when the line that you want to see appears on the screen, hold down (SHIFT) and press (@). This pauses the listing. You can press any key to continue.

Learned in Chapter 5

COMMAND	KEYBOARD
FOR/NEXT	(SHIFT)(@)
STEP	

6 / DECISIONS, DECISIONS...

Here's an easy decision for the computer:

- **If** you type ORANGE . . . **then** make the screen orange.

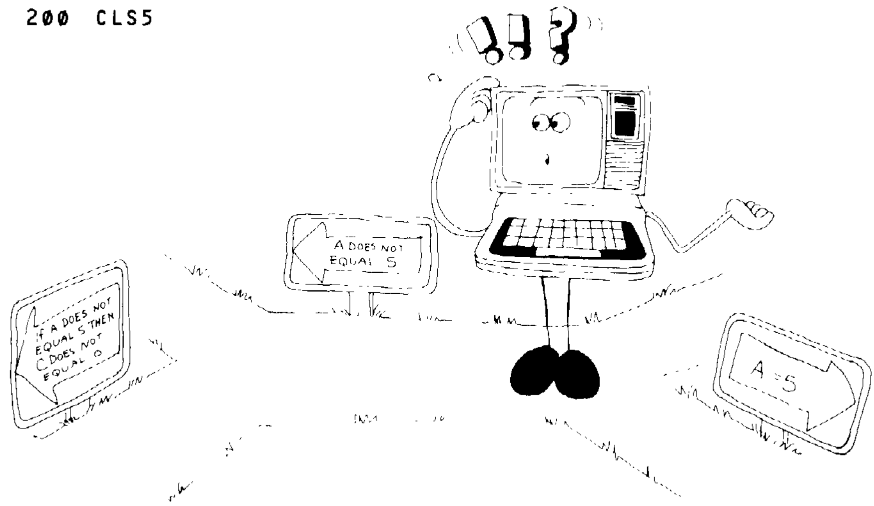
  or

- **If** you type BUFF . . . **then** make the screen buff.

Easy enough? Then, have the computer do it.

Type this program:

```
10  PRINT "DO YOU WANT THE SCREEN ORANGE OR BUFF?"
20  INPUT C$
30  IF C$ = "ORANGE" THEN 100
40  IF C$ = "BUFF" THEN 200
100  CLS8
110  END
200  CLS5
```



Run the program a few times. Try both ORANGE and BUFF as answers.

If you answer ORANGE . . . **then** . . .

1. Line 30 sends the computer to Line 100.
2. Line 100 turns your screen orange.
3. Line 110 ends the program. (If the computer gets to Line 110, it never makes it to Line 200.)

   On the other hand . . .

If you answer BUFF . . . **then** . . .

1. Line 40 sends the computer to Line 200.
2. Line 200 turns your screen buff.
3. Line 200 is the last line in the program, so the program ends.

What happens if you answer with something besides ORANGE or BUFF? Run the program again. This time, answer GREEN.

This makes the screen orange. Do you know why?

If the condition is false, the computer ignores the THEN part of the line and proceeds to the next program line.

The command you used to have the computer make a decision is the IF/THEN command. Here's the syntax for IF/THEN, but don't expect to understand it all until you reach the end of this chapter.

IF *condition* THEN *commands* ELSE *commands* Tests the *condition*. If it's true, BASIC executes the *commands* following THEN. If it's false, BASIC executes the *commands* following ELSE or, if ELSE is omitted, does nothing.

You also used the END command. Its syntax is:

END Ends program execution.

Adding Another Level (The ELSE Option and Nested IF/THEN Commands)

Take another look at the program below:

```
10  PRINT "DO YOU WANT THE SCREEN ORANGE OR BUFF?"
20  INPUT C$
30  IF C$ = "ORANGE" THEN 100
40  IF C$ = "BUFF" THEN 200
100  CLS8
110  END
200  CLS5
```

By using ELSE, you can rephrase the decision in this way:

If you type ORANGE . . .

. . . **then** . . .

Make the screen orange.

. . . or **else** . . .

If you type BUFF. . .

. . . **then** . . .

Make the screen buff.

You can say all of this to the computer in one line. Erase Lines 30-200, and type this as a new Line 30:

```
30  IF C$ = "ORANGE" THEN CLS8 ELSE IF C$="BUFF" THEN
    CLS5
```

Run the program, and it works exactly as it did before.

- If C$ equals "ORANGE," the computer executes the command following THEN, which is the CLS8 command.
- If C$ does not equal "ORANGE," the computer executes the command following ELSE, which is another IF/THEN command.

Note that Line 30 nests one IF/THEN command within another. You can nest as many IF/THEN commands as you want, provided the program line does not contain more than 249 characters (the maximum line length). You can also test if something is not equal to (<>), greater than (>), less than (<), greater than or equal to (>=) or less than or equal to (<=) by using one of the symbols in parenthesis instead of = in a test. For example:

```
30 IF A>B THEN 100
```

would go to 100 if A was greater than B.

. . . And Still More Levels . . . (Additional ELSE Clauses)

Suppose that in addition to what the computer did before, you want to tell the computer what ELSE it should do if C$ does NOT equal ORANGE.

In other words, you want to add another ELSE clause to your decision:

If you type ORANGE . . .

. . . **then** . . .

Make the screen orange.

. . . or **else** . . .

If you type BUFF. . .

. . . **then** . . .

Make the screen buff.

. . . or **else** . . .

```
PRINT "YOU MUST TYPE ORANGE OR BUFF"
```

To add this new ELSE clause, insert ELSE, followed by the PRINT command, at the end of Line 30:

```
30  IF C$ = "ORANGE" THEN CLS8 ELSE IF C$="BUFF"
    THEN CLS5 ELSE PRINT "YOU MUST TYPE ORANGE OR BUFF"
```

Run the program, but this time, answer the computer's question with something besides orange or buff. This response causes the computer to print the message "YOU MUST TYPE ORANGE OR BUFF."

You can add as many ELSE clauses as you want, as long as the entire line does not contain more than 249 characters.

... And Packing Even More Into a Decision (The Colon (:))

To add even more power to the IF/THEN command, you can have the computer carry out any number of commands following each THEN or ELSE clause. For example, assume you want the computer to do this:

If you type ORANGE . . .

. . . **then** . . .

Make the screen orange and go to Line 10.

. . . or **else** . . .

If you type BUFF.

. . . **then**

Make the screen buff and go to Line 10.

. . . or **else** . . .

Print a message and go to Line 20.

You can say all this in one line by using colons to separate each command. Change Line 30 to the following:

```
30  IF C$ = "ORANGE" THEN CLS8:GOTO 10 ELSE IF
    C$="BUFF" THEN CLS5:GOTO 10 ELSE PRINT "YOU MUST
    TYPE EITHER ORANGE OR BUFF":GOTO 20
```

You can add as many commands as you want to a THEN or ELSE clause, as long as the line does not contain more than 249 characters.

A More Subtle Decision (The AND and OR Conditions)

By using two more words, AND and OR, you can ask the computer to make a more subtle decision. For example, suppose you want to write a program that tests to see whether a candidate meets these job requirements:

A degree in programming
AND
Experience in programming

Erase the memory. Then type and run this program:

```
10  PRINT "DO YOU HAVE--"
20  INPUT "A DEGREE IN PROGRAMMING"; D$
30  INPUT "EXPERIENCE IN PROGRAMMING"; E$
40  IF D$="YES" AND E$="YES" THEN PRINT "YOU'RE
    HIRED!" ELSE PRINT "SORRY, WE CAN'T HIRE YOU"
50  GOTO 10
```

If you respond to both questions with YES, the computer reaches this decision:

YOU'RE HIRED!

If, on the other hand, you respond to the first question with YES, but respond to the second question with NO, the computer is forced to reach this decision:

SORRY, WE CAN'T HIRE YOU

Now, assume the requirements change so that AND becomes OR. The job requires:

A degree in programming
OR
Experience in programming

Change Line 40 so that AND becomes OR:

```
40 IF D$="YES" OR E$="YES" THEN PRINT "YOU'RE HIRED!"
   ELSE PRINT "SORRY, WE CAN'T HIRE YOU"
```

Run the program. Respond to the first question with "YES" and the second question with "NO"—just as you did before—and see the difference that one word can make:

YOU'RE HIRED!

Learned in Chapter 6

COMMANDS

IF/THEN/ELSE
END
AND
OR

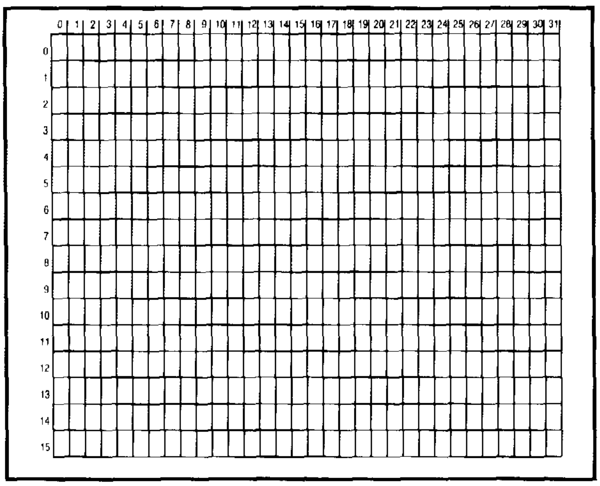
7 / THE SCREEN

So far, you've used only one screen. In this chapter, you learn to use a new screen (a big screen) and four new commands.

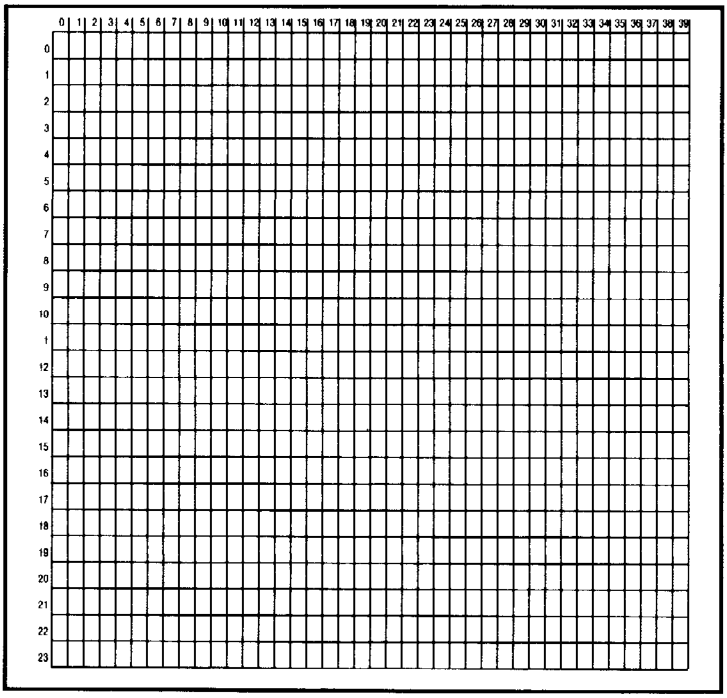
If you have an RGB monitor, type **PALETTE RGB** (ENTER) before starting this chapter. This way your colors will agree with ours.

The Big Screen (The WIDTH Command)

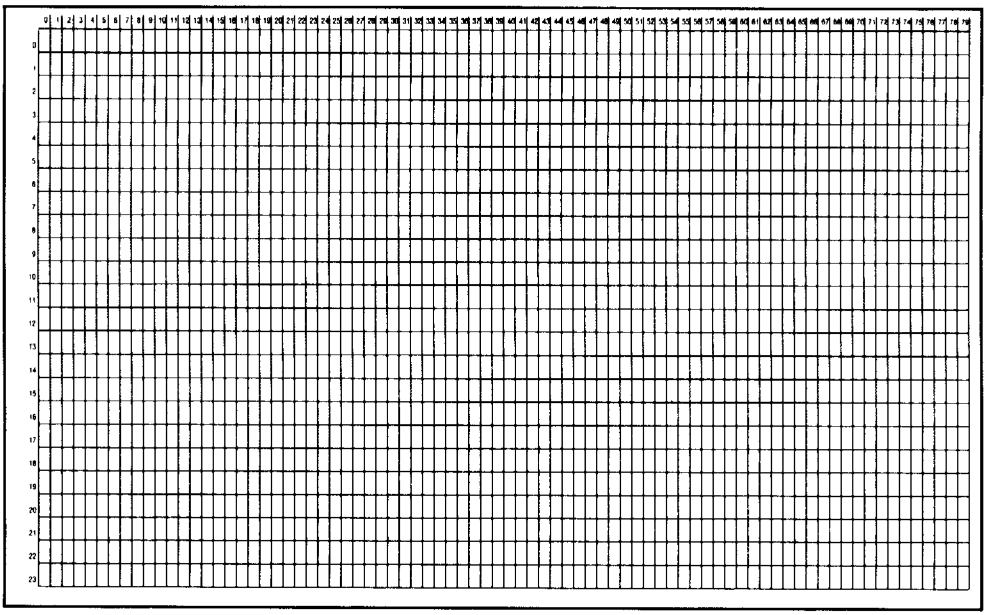
The screen you're using now is a 32 X 16 text screen. It can display *text* (alphanumeric characters) in a format that is 32 *columns* (numbered 0-31) wide by 16 *rows* (numbered 0-15) deep:



Your computer has two other text screens that you can use. One is 40 X 24:



The other is 80 X 24:



To change from screen to screen, use the WIDTH command. Its syntax is:

> **WIDTH** 40, 80, or 32 changes to the 40 x 24 screen, the 80 x 24 screen, or the 32 x 16 screen; then clears the screen.

Use WIDTH to change to the 40 x 24 screen. Type:

> **WIDTH 40** (ENTER)

The 40 x 24 screen is different from the 32 x 16 screen. The border is green, rather than black. The cursor is a blinking underline character, rather than a blinking block. The characters are also different in size.

Change to the 80 x 24 screen. Type:

> **WIDTH 80** (ENTER)

The 80 x 24 screen looks the same as the 40 x 24 screen, except the characters are thinner. Move back to the 32 x 16 screen. Type:

> **WIDTH 32** (ENTER)

The 40 x 24 and 80 x 24 screens are both called *high-resolution* text screens. The 32 x 16 screen is called a *low-resolution* text screen.

Screen Positions (The PRINT @ and LOCATE Commands)

You can position text on the low-resolution text screen with the PRINT @ command. You can position text on the high-resolution text screen with the LOCATE command.

The syntax for PRINT @ is:

> **PRINT @** *n*, *message* Locates the cursor at Position *n* on the low-resolution text screen; then prints *message*.

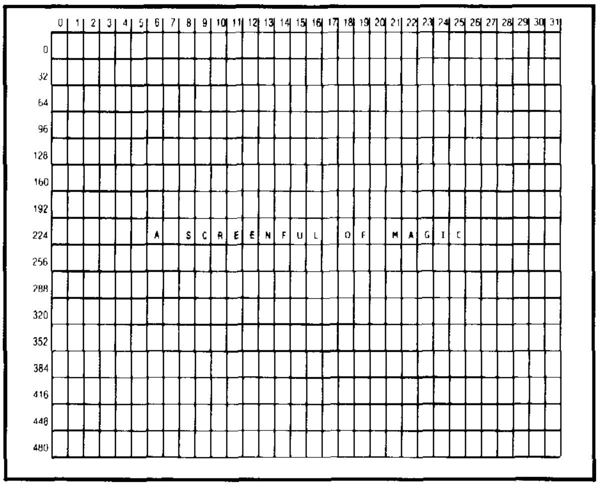
The syntax for LOCATE is:

> **LOCATE** *x*,*y* Locates the cursor at Column *x*, Row *y*. The column can be 0-39 for a 40 x 24 screen or 0-79 for an 80 x 24 screen. The row can be 0-23 for either screen.

This program, which we call "Magic," illustrates how PRINT @ works:

```
10  WIDTH 32
20  PRINT @ 230, "A SCREENFUL OF MAGIC"
```

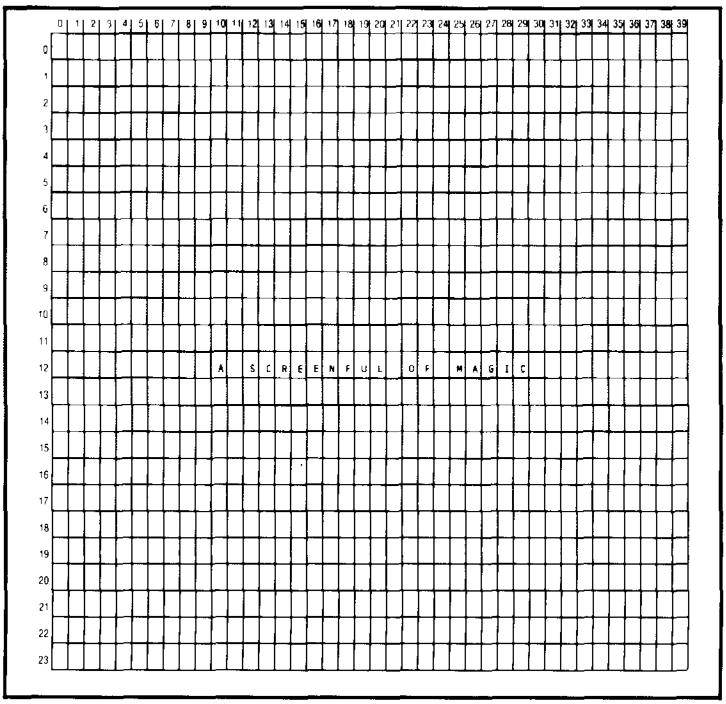
When you run this program, BASIC moves to the low-resolution text screen, locates the cursor at Position 230 (224 + 6) and prints the message **"A SCREENFUL OF MAGIC."**



This version of Magic illustrates how LOCATE works:

```
10 WIDTH 40
20 LOCATE 10,12
30 PRINT "A SCREENFUL OF MAGIC"
```

When you run this program, BASIC locates the cursor at Column 10, Row 12 and prints the message **"A SCREENFUL OF MAGIC."**



PRINT @ and LOCATE work in a similar way, but there are two differences. First, PRINT @ locates the cursor at a PRINT @ position; LOCATE locates the cursor at a row, column position. And second, PRINT @ prints a message; LOCATE does not print a message.

PRINT @ works only on the low-resolution text screen; LOCATE works only on a high-resolution text screen. If you attempt to use these words on the wrong screens, you get an ?HP ERROR.

Printing in Straight Lines The PRINT TAB Command

TAB is very handy for printing things in nice, neat columns. It's syntax is:

```
PRINT TAB(n)
```

Moves the cursor to column n on the low- and high-resolution text screens.

Try using TAB. Type:

```
PRINT TAB(22);"HELLO!" (ENTER)
```

The computer moves to column twenty-two and prints HELLO!. To print a nice neat column of numbers, enter this program:

```
10  FOR Z = 500 TO 509
20  PRINT TAB(10);Z
30  NEXT Z
```

See how easy it is to print columns with TAB? Try using it in your programs now.

Lowercase Letters (The (SHIFT)(0) Keys)

You can enter the upper/lowercase mode on both the low- and high-resolution screens. However, the way that each screen displays lowercase letters is different.

- The low-resolution text screen displays lowercase letters in reversed colors.
- The high-resolution text screen displays lowercase letters in true lowercase.

Try using the upper/lowercase mode on both screens. Hold down (SHIFT) and press (0). Then, retype Line 30 from the Magic Program. Use (SHIFT) for the uppercase letters, as you would on a normal typewriter. Type:

```
30  PRINT "A Screenful of Magic"
```

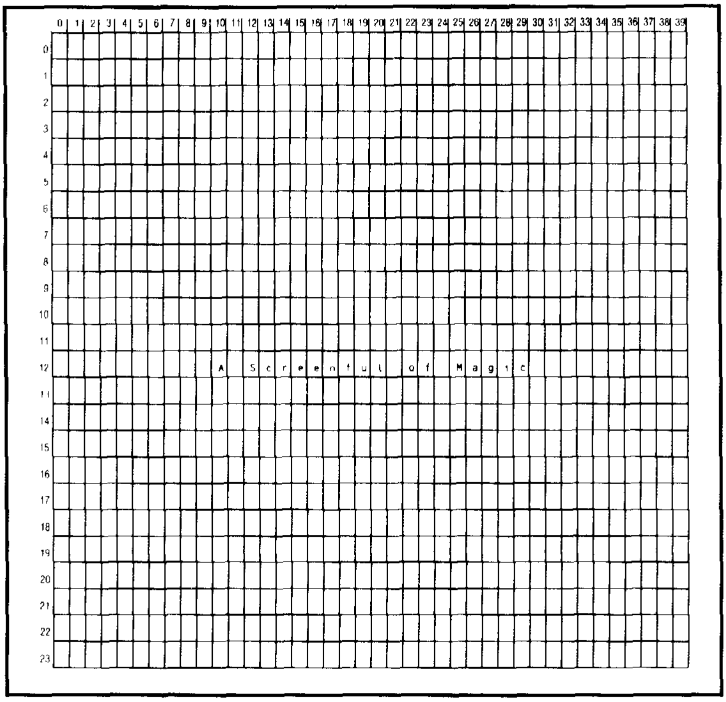
Type LIST (ENTER) to list the program. On the low-resolution text screen, Line 30 looks like this:

```
30  PRINT "A SCREENFUL OF MAGIC"
```

On the high-resolution text screen, Line 30 looks like this:

```
30  PRINT "A Screenful of Magic"
```

Run the program. Because of the WIDTH 40 line in Line 10 of the program, BASIC uses the 40 x 24 high-resolution text screen and displays this message on the screen:



Return to the uppercase-only mode by holding down (SHIFT) and pressing (0).

Colors (The CLS Command)

You can use the CLS command on both the low- and high-resolution screens. The way that CLS works, however, is slightly different at each screen.

- On the low-resolution text screen, CLS clears the screen and displays a color.
- On the high-resolution text screen, CLS clears the screen, displays a color, **and changes the background color.**

If you have an RGB monitor, type **PALETTE RGB** (ENTER) before reading further. This way your colors will agree with ours.

Now, use CLS on both screens. First, move to the low-resolution text screen and enter CLS with color number 8 (orange). Type:

```
WIDTH 32 (ENTER)
CLS8 (ENTER)
```

You see a clear orange screen with a stripe at the top. The stripe is green, the current background color.

Now, move to the high-resolution text screen and enter the same command. Type:

```
WIDTH 40 (ENTER)
CLS8 (ENTER)
```

You see simply a clear orange screen (no green stripe). The CLS command changed the background color to orange as well.

Try some other background colors. You can use any number between 1 and 8.

To see how CLS works in a program, add Line 15 to our Magic Program:

```
10  WIDTH 40
15  CLS3
20  LOCATE 10,12
30  PRINT "A Screenful of Magic"
```

Run the program. The computer shows "A Screenful of Magic" on a blue background.

Dramatic Highlights (The ATTR Command)

You can use a special command, ATTR, to highlight text on the high-resolution text screen. ATTR works **only** on the high-resolution text screen; it does not work on the low-resolution text screen. Its syntax is:

ATTR *c1*, *c2*, B, U Highlights text by setting the foreground to Color *c1* (0-7) and the background to Color *c2* (0-7). If you specify B, the text blinks. If you specify U, the text is underlined.

To see how ATTR works, type:

```
ATTR 3, 2 (ENTER)
```

Now, type some characters. The characters you type are highlighted. The foreground color is buff, and the background color is blue.

Now, type:

```
ATTR 2, 3, B (ENTER)
```

The characters you type are highlighted in a different way. The foreground color is black, the background color is red, and the characters are all blinking.

To underline your characters, type:

```
ATTR 2, 3, U (ENTER)
```

Try other combinations of foreground and background colors. You can use any number in the range 0-7 for the foreground and background colors.

This program uses ATTR to highlight the message produced by the Magic Program:

```
10  WIDTH 40
15  CLS4
20  LOCATE 10,12
25  ATTR 2,3,B
30  PRINT "A Screenful of Magic"
```

When you run the program, the words "A Screenful of Magic" are highlighted. The foreground is black, the background is red, and the characters are all blinking.

Color Numbers

You might have noticed that the color numbers produce different colors depending on whether you use them with CLS, ATTR as the foreground color, or ATTR as the background color.

For example, Color 3 produces:

- Blue, with CLS.
- Buff, with ATTR as the foreground color.
- Red, with ATTR as the background color.

The next chapter explains why this is so.

Learned in Chapter 7

COMMANDS	KEYS
WIDTH	(SHIFT)(0)
PRINT @	
LOCATE	
ATTR	
CLS	

8 / COLORS

Your color computer can produce 64 colors, but so far, you've used only nine of them. This chapter shows how to use the many other colors that are available.

If you have an RGB monitor, be sure to type **PALETTE RGB** (ENTER) every time you turn on your computer. This way your colors will agree with ours.

Specifying Colors for 40/80 Column Text (The Palette)

The color computer has a special area in memory called a *palette*. The palette contains 16 *slots*. Each slot contains a color.

When you use a color number in a BASIC command, you are specifying a palette slot. For example, type:

CLS3 (ENTER)

Your screen is now blue. This is because CLS3 specifies Slot 2 and Slot 2 contains the code for blue.

As another example, type:

ATTR 3, 3 (ENTER)

Then, type some characters. The foreground is buff and the background is red. This is because ATTR 3,3 specifies Slot 11 for the foreground and Slot 3 for the background. Slot 11 contains the code for buff. Slot 3 contains the code for red.

Notice that the color numbers specify different slots, depending on whether you use them with CLS, ATTR as the foreground color, or ATTR as the background color.

For example, Color 3 specifies:

- Slot 2, with CLS.
- Slot 11, with ATTR as the foreground color.
- Slot 3, with ATTR as the background color.

The following tables show which slot each color number specifies when used with CLS, ATTR as the foreground color, and ATTR as the background color. They also show the standard colors that are stored in each of these slots.

Table 8.1. CLS and the Palette

Color Number	Palette Slot	Standard Color
0	8	Black
1	0	Green
2	1	Yellow
3	2	Blue
4	3	Red
5	4	Buff
6	5	Cyan
7	6	Magenta
8	7	Orange

Table 8.2. ATTR Foreground and the Palette

Color Number	Palette Slot	Standard Color
0	8	Black
1	9	Green
2	10	Black
3	11	Buff
4	12	Black
5	13	Green
6	14	Black
7	15	Orange

Table 8.3. ATTR Background and the Palette

Color Number	Palette Slot	Standard Color
0	0	Green
1	1	Yellow
2	2	Blue
3	3	Red
4	4	Buff
5	5	Cyan
6	6	Magenta
7	7	Orange

Using Nonstandard Colors (The PALETTE Command)

You can change the color stored in a palette slot with the PALETTE command. The syntax of PALETTE is.

`PALETTE` *slot, color code* Stores a *color code* (0-63) into a palette *slot* (0-15).

For example, type:

`CLS3` (ENTER)

Your screen is now blue. As you learned earlier, CLS3 specifies the color stored in Slot 2 and Slot 2 contains the code for blue.

Now, use PALETTE to store a different code in Slot 2. Type:

`PALETTE 2, 2` (ENTER)

The color of your screen instantly changes to green.

Try another code. Type:

`PALETTE 2, 14` (ENTER)

Again, the color of your screen changes.

Some codes produce different colors on a CMP monitor than they produce on an RGB monitor. Try storing other codes in slot 2. You can use any code in the range 0 to 63.

Name the Colors!
(Naming Colors for Future Reference)

Each display may produce somewhat different colors for each palette code. Type in and run Sample Program number 23. This program will show you how each of the sixty-four colors appears on your display, eight at a time. Find the "Color Codes" section near the back of the book. There are sixty-four blank lines, labeled zero to sixty-three. Write a name for each of the colors you see, next to the appropriate number. Pat yourself on the back, and try out Sample Program number 24. You will see an exciting assortment of colors, selected at random from the sixty-four available colors. Excited? Keep reading, and learn how you can use these colors in your own programs.

Using Nonstandard Colors in a Program
(An Example of PALETTE)

This is the Magic program from the last chapter:

```
10  WIDTH 40
15  CLS4
20  LOCATE 10,12
25  ATTR 3,2,B
30  PRINT "A Screenful of Magic"
```

Suppose you want to change the palette so that the ATTR 3,2,B command produces a sunshine yellow foreground.

The steps are:

1. Determine which palette slot produces the foreground color (the "3" in the ATTR 3,2,B command).

   By referring to Table 8.2, you see that Slot 11 produces the foreground color.

2. Look up the code for sunshine yellow by referring to "Color Codes."

3. Store the code (from Step 2) in the specified palette slot (from Step 1), by adding this line to your program:

   `2 PALETTE 11, XX` (XX represents the color code)

4. Run the program.

Returning to Standard Colors (The PALETTE CMP and PALETTE RGB Commands)

After altering the palette, you might want to return it to its standard condition. You can do this with the PALETTE CMP and PALETTE RGB commands. Their syntaxes are:

PALETTE CMP Sets the computer to display the standard colors on a CMP monitor.

PALETTE RGB Sets the computer to display the standard colors on an RGB monitor.

If you have a CMP monitor, type:

PALETTE CMP (ENTER)

If you have an RGB monitor, you have already used the PALETTE RGB command. Type:

PALETTE RGB (ENTER)

Your palette is now back to normal. If you type **CLS3** (ENTER), for example, your screen again shows blue.

Learned in Chapter 8

COMMAND

PALETTE

9 / RANDOM CHANCE

Thanks to BASIC's random numbers, the computer can play almost any game of chance.

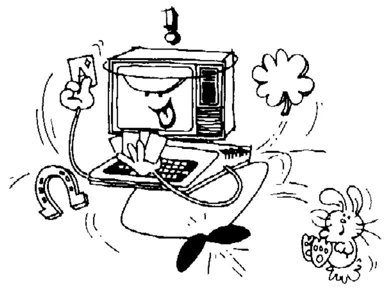
Picking a Random Number (The RND Function)

Type this program:

```
10 PRINT RND(10)
```

Run it. The computer just picked a random number from 1 to 10. Run it some more times.

It's as if the computer is drawing a number from 1 to 10 out of a hat. The number it picks is unpredictable.



Type and run this next program. Press BREAK when you're satisfied that the numbers are random.

```
10 PRINT RND(10);
20 GOTO 10
```

To get random numbers from 1 to 100, change Line 10 and run the program.

```
10 PRINT RND(100);
```

Unlike the other BASIC words you used, RND is a *function*. This means that RND returns a value—in this case, a number. Because RND returns a number, you can use RND in the same way that you would use a number.

These commands illustrate how you could use RND:

```
PRINT RND(10)+15
SOUND RND(255), RND(255)
CLSRND(8)
FOR N=1 TO RND(5) ... NEXT N
```

The syntax for RND is:

RND(*n*) Returns random number between 1 and *n* (if *n* is greater than 1) or between 0 and 1 (if *n* equals 0).

The rest of this chapter is just for fun. If you are in a hurry to learn more BASIC words, you can skip to the next chapter.

A Random Show (An Example of RND)

Have the computer compose a song made up of random tones Type:

```
10  T = RND(255)
20  SOUND T, 1
30  GOTO 10
```

Run it. Great music, eh? Press BREAK when you've heard enough.

DO-IT-YOURSELF PROGRAM 9-1

Add some lines to make the computer show a random color (1-8) right before it sounds each random tone.

Rolling the Dice (An Example of RND)

In this game, the computer rolls two dice. To do this, it must come up with two random numbers. Type:

```
10  CLS
20  X = RND(6)
30  Y = RND(6)
40  R = X + Y
50  PRINT X,
60  PRINT Y
70  PRINT "YOU ROLLED A" R
80  INPUT "DO YOU WANT ANOTHER ROLL?"; A$
90  IF A$ = "YES" THEN 10
```

Run the program.

Line 10 clears the screen.

Line 20 picks a random number from 1 to 6 for one die.

Line 30 picks a random number for the other die.

Line 40 adds the two dice to get the total roll.

Lines 50-70 print the results of the roll.

Line 80 lets you input whether you want another roll. If you answer YES at Line 90, the program goes to Line 10 and runs again. Otherwise, the program ends.

DO-IT-YOURSELF PROGRAM 9-2

Because you know how to roll dice, it will be easy to write a Craps program. These are the rules of the game (in its simplest form):

1. The player rolls two dice. If the first roll is a 2 (snake eyes), a 3 (cock-eyes), or a 12 (boxcars), the player loses, and the game is over.
2. If the first roll is either a 7 or 11 (a natural), the player wins, and the game's over.
3. If the first roll is any other number, the point goes to the player. The player must keep rolling until either "making the point" by getting the same number again to win, or rolling a 7, and losing.

   You already know more than enough to write this program. Make the computer print it in an attractive format on your screen, and keep the player informed about what is happening. It might take you a while to finish, but give it your best. Good luck!

Learned in Chapter 9

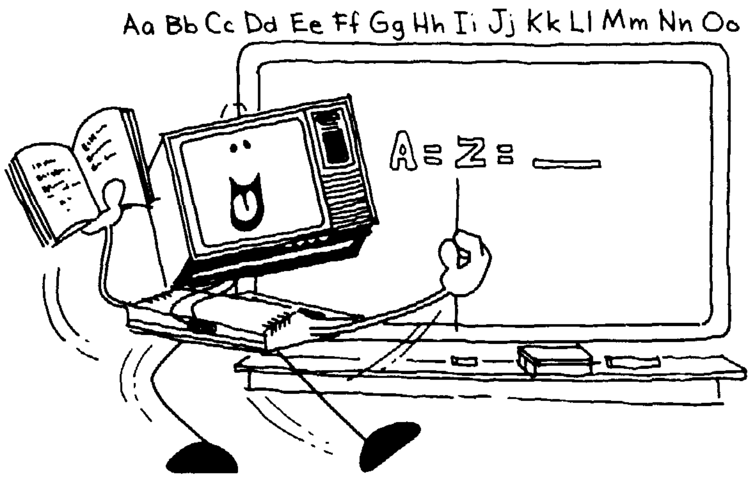
FUNCTION

RND

10 / READING

In this chapter, you teach the computer to read. You do this using three new commands, READ, DATA, and RESTORE. You also learn a new function: INT.

Reading Data (The DATA and READ Commands)



Type and run this program:

```
10  DATA APPLES, ORANGES, PEARS
20  FOR X = 1 TO 3
30  READ F$
40  NEXT X
```

Nothing appears to happen. To see what the computer is doing, add this line and run the program:

```
35 PRINT "F$ = :" F$
```

Line 30 tells the computer to:

1. Look for a DATA line.
2. READ the first item in the list, APPLES.
3. Give APPLES an F$ label.
4. Go to the next item.

The second time the computer gets to Line 30, it is told to do the same:

1. Look for a DATA line.
2. READ the first item. This time, it's ORANGES.
3. Give ORANGES the F$ label.
4. Go to the next item.

You can insert DATA lines wherever you want in the program. Run each of these programs. They all work the same.

```
10 DATA APPLES
20 DATA ORANGES
30 FOR X = 1 TO 3
40 READ F$
50 PRINT "F$ = :" F$
60 NEXT X
70 DATA PEARS
```

```
10 DATA APPLES, ORANGES
20 DATA PEARS
30 FOR X = 1 TO 3
40 READ F$
50 PRINT "F$ = :" F$
60 NEXT X
```

```
30 FOR X = 1 TO 3
40 READ F$
50 PRINT "F$ = :" F$
60 NEXT X
70 DATA APPLES
80 DATA ORANGES
90 DATA PEARS
```

```
30 FOR X = 1 TO 3
40 READ F$
50 PRINT "F$ = :" F$
60 NEXT X
70 DATA APPLES, ORANGES,
   PEARS
```

The syntax for DATA is:

DATA *data items* Inserts data items in the program.

The syntax for READ is:

READ *variable* Reads the next data item in the program and stores it in *variable*.

Reading the Same Data—Over and Over (The RESTORE Command)

Look at the original DATA program:

```
10 DATA APPLES, ORANGES, PEARS
20 FOR X = 1 TO 3
30 READ F$
40 NEXT X
```

What if you want the computer to read the same list over and over? Type:

```
60 GOTO 10
```

Run the program. The computer displays ?OD ERROR IN 30 (Out of Data Error in Line 30). The first time the computer reads the data items, it crosses them out. Then, when asked to go back to Line 30 and read the crossed-out data items, the computer displays ?OD ERROR.

Type this line and run the program:

```
50 RESTORE
```

Now, it's as if the computer never crossed out any data items. It reads the same data again and again.

The syntax for RESTORE is:

`RESTORE` Moves the computer's data pointer back to the first data item.

A Vocabulary Building Test (The INT Function)

This example program uses DATA, READ, and RESTORE to have the computer drill you on words and definitions. Here are the words and definitions we use:

```
10  DATA TACITURN, HABITUALLY UNTALKATIVE
20  DATA LOQUACIOUS, VERY TALKATIVE
30  DATA VOCIFEROUS, LOUD AND VEHEMENT
40  DATA TERSE, CONCISE
50  DATA EFFUSIVE, DEMONSTRATIVE OR GUSHY
```

To write a program in which the computer drills you on these words and definitions, you need to have it select words at random. Type:

```
60  N = RND(10)
70  FOR X = 1 TO N
80  READ A$
90  NEXT X
100  PRINT "THE RANDOM WORD IS:" A$
```

Run the program a few times. At this point, the program doesn't work quite right. The computer is just as likely to stop at a definition as at a word.

What the computer really needs to do is pick a random word only from items 1, 3, 5, 7, or 9, rather than from all the items. In other words, N (the random number) needs to always be an odd number.

Although BASIC does not have a function that converts even numbers to odd numbers, it does have an INT function that you can use to make this conversion.

INT converts a number to its "whole part" and deletes the decimal part. For example, INT(3.9) equals 3. The syntax for INT is:

`INT`(*n*) Returns the "whole part" of *n*. *n* can be any number.

You can use N to convert even numbers to odd numbers by typing this line:

```
65  IF INT(N/2) = N/2 THEN N = N - 1
```

This is what Line 65 does:

- If N equals an even number, Line 65 subtracts 1 from N to make N equal to an odd number.

  For example, if N equals 10, Line 65 makes the following calculation.

  INT(10/2) = 10/2
  INT(5) = 5
  5 = 5

  Because the results are true (5 does equal 5), Line 65 subtracts 1 from N to make N equal to 9.

- If N equals an odd number, Line 65 leaves N unchanged.

  For example, if N equals 9, Line 65 makes the following calculation.

  INT(9/2) = 9/2
  INT(4.5) = 4.5
  4 = 4.5

  Because the results are false (4 does not equal 4.5), Line 65 leaves N unchanged.

Now, add these lines so that the computer will read each word's definition:

```
110  READ B$
120  PRINT "THE DEFINITION IS :" B$
```

Add these lines so that the computer will read from the same list over and over:

```
130  RESTORE
140  GOTO 60
```

List the program. This is how it looks now:

```
10  DATA TACITURN, HABITUALLY UNTALKATIVE
20  DATA LOQUACIOUS, VERY TALKATIVE
30  DATA VOCIFEROUS, LOUD AND VEHEMENT
40  DATA TERSE, CONCISE
50  DATA EFFUSIVE, DEMONSTRATIVE OR GUSHY
60  N = RND(10)
65  IF INT(N/2) = N/2 THEN N = N - 1
70  FOR X = 1 TO N
80  READ A$
90  NEXT X
100  PRINT "A RANDOM WORD IS :" A$
110  READ B$
120  PRINT "ITS DEFINITION IS :" B$
130  RESTORE
140  GOTO 60
```

DO-IT-YOURSELF PROGRAM 10-1

Want to complete this program? Add lines to have the computer:

1. Print the definition *only.*
2. Ask you for the word.
3. Compare the word with the correct random word.
4. Tell you if your answer is correct. If your answer is incorrect, print the correct word.

Learned in Chapter 10

COMMANDS	FUNCTION
DATA	INT
READ	
RESTORE	

11 / HELP WITH ARITHMETIC

Solving long math problems fast and accurately is a task your computer does with ease. This chapter shows how to use some short cuts when typing long, difficult formulas.

Subroutines (The GOSUB and RETURN Commands)

An easy way to handle complicated math formulas is by using the GOSUB and RETURN commands to set up a *subroutine*. You always use GOSUB and RETURN together. Their syntaxes are:

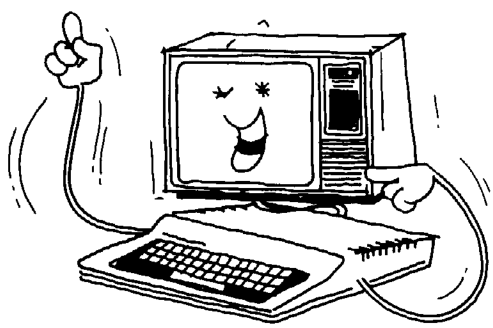
`GOSUB` *line number* Goes to the subroutine beginning at *line number*.

`RETURN` Returns from the subroutine to the command immediately following the corresponding GOSUB command.

Type and run this program:

```
10  PRINT "EXECUTING THE MAIN PROGRAM"
20  GOSUB 500
30  PRINT "NOW BACK IN THE MAIN PROGRAM"
40  END

500  PRINT "EXECUTING THE SUBROUTINE"
510  RETURN
```



GOSUB 500 tells the computer to go to the subroutine that starts at Line 500. RETURN tells the computer to return to the BASIC word that immediately follows GOSUB.

Labeling Subroutines (The REM Command)

This subroutine multiplies any number by 100:

```
10  INPUT "TYPE A NUMBER"; N
20  GOSUB 2000
30  PRINT N "TIMES 100  IS" R
40  GOTO 10
2000  REM FORMULA FOR MULTIPLYING A NUMBER BY 100
2010  R = N * 100
2020  RETURN
```

Notice the REM command in line 2000. REM lets you insert a comment in a program. Its syntax is:

REM *comment* Inserts any comment in a program line, without the comment having any effect on the program.

You can insert REM lines anywhere you want in your program. They make no difference in the way the program works. To see for yourself, add these lines and run the program:

```
5  REM THIS IS A PECULIAR PROGRAM,
17  REM WILL THIS LINE CHANGE THE PROGRAM?
45  REM THE NEXT LINE KEEPS THE SUBROUTINE SEPARATED
```

Using More Than One Subroutine (The ON GOSUB Command)

The ON GOSUB command makes it easy for you to include more than one subroutine in a program. The syntax for ON GOSUB is:

ON *n* **GOSUB** *line numbers* Goes to the subroutine beginning at the *n*th line number.

To see how ON GOSUB works, type this program:

```
10  INPUT "TYPE 1, 2, OR 3"; N
20  ON N GOSUB 100, 200, 300
30  GOTO 10

100  PRINT "YOU TYPED 1"
110  RETURN

200  PRINT "YOU TYPED 2"
210  RETURN

300  PRINT "YOU TYPED 3"
310  RETURN
```

Run it.

Line 20 works the same as these three commands:

```
18  IF N = 1 THEN GOSUB 100
20  IF N = 2 THEN GOSUB 200
22  IF N = 3 THEN GOSUB 300
```

ON GOSUB causes the computer to look at the line number following ON (in this case N).

- If N is **1,** the computer goes to the subroutine starting at the **first line number** following GOSUB.
- If N is **2,** the computer goes to the subroutine starting at the **second line number**.
- If N is **3,** the computer goes to the subroutine starting at the **third line number**.

What if N is 4? Because Line 20 doesn't have a fourth line number, the computer simply goes to the next line in the program.

Here is a program that uses ON GOSUB.

```
5   FOR P = 1 TO 600: NEXT P
10  CLS: X = RND(100): Y = RND(100)
20  PRINT "(1) ADDITION"
30  PRINT "(2) SUBTRACTION"
40  PRINT "(3) MULTIPLICATION"
50  PRINT "(4) DIVISION"
60  INPUT "WHICH EXERCISE(1-4)";R
70  CLS

80  ON R GOSUB 1000, 2000, 3000, 4000
90  GOTO 5

1000  PRINT "WHAT IS" X "+" Y
1010  INPUT A
1020  IF A = X + Y THEN PRINT "CORRECT" ELSE PRINT "WRONG"
1030  RETURN

2000  PRINT "WHAT IS" X "-" Y
2010  INPUT A
2020  IF A = X-Y THEN PRINT "CORRECT" ELSE PRINT "WRONG"
2030  RETURN

3000  PRINT "WHAT IS" X "*" Y
3010  INPUT A
3020  IF A = X*Y THEN PRINT "CORRECT" ELSE PRINT "WRONG"
3030  RETURN

4000  PRINT "WHAT IS" X "/" Y
4010  INPUT A
4020  IF A = X/Y THEN PRINT "CORRECT" ELSE PRINT "WRONG"
4030  RETURN
```

Going To More Than One Place (The ON GOTO Command)

The ON GOTO command is similar to the ON GOSUB command that you just learned about. Its syntax is:

`ON` *n* `GOTO` *line numbers* calls the subroutine beginning at the *n*th line number.

It works the same way as ON GOSUB, except that it performs a GOTO the selected line, instead of a GOSUB. In the following sample program, ON GOTO determines what the computer prints when you press 1-3.

```
10  A$=INKEY$:IF A$="" THEN 10
20  IF A$ < "1" OR A$ > "3" THEN 10
30  B=VAL(A$)
40  ON B GOTO 100,200,300
50  GOTO 10
100  PRINT "YOU PRESSED 1":GOTO 10
200  PRINT "YOU PRESSED 2":GOTO 10
300  PRINT "YOU PRESSED 3":GOTO 10
```

RUN the program and see what happens.

Give the Computer a Little Help (Parentheses)

As math formulas get more complex, your computer needs help understanding them. For example, what if you want the computer to solve this problem:

Divide the sum of 13 + 3 by 8

You might want the computer to arrive at the answer this way:

13 + 3 / 8 = 16/8 = 2

Instead, the computer arrives at another answer. Type this command line and see:

`PRINT 13 + 3 / 8` (ENTER)

The computer solves problems using these rules:

RULES ON ARITHMETIC

The computer solves arithmetic problems in this order:

1. First, it solves any exponentiation operations.
2. Second, it solves any multiplication and division operations.
3. Last, it solves addition and subtraction operations.
4. If there's a tie (that is, more than one exponentiation, multiplication/division, or addition/subtraction operation), it solves the operations from left to right.

The computer solves the problem above using its rules:

- First, it does the division (3/8 = .375)
- Then, it does the addition (13 + .375 = 13.375)

If you want the computer to solve the problem differently, you need to use parentheses. Type this line:

`PRINT (13 + 3) / 8` (ENTER)

Whenever the computer sees an operation in parentheses, it solves that operation before solving any others.

COMPUTER MATH EXERCISE

What do you think the computer will print as the answers to each of these problems?

```
PRINT 10 - (5 - 1) / 2 ______________________
PRINT 10 - 5 - 1 / 2 ______________________
PRINT (10 - 5 - 1) / 2 ______________________
PRINT (10 - 5) - 1 / 2 ______________________
PRINT 10 - (5 - 1 / 2) ______________________
```

Finished? Type each of the command lines to check your answers.

What if you want the computer to solve this problem?

Divide 10 minus the difference of 5 minus 1 by 2

You're actually asking the computer to do this:

(10 - (5 - 1)) / 2

When the computer sees a problem with more than one set of parentheses, it solves the inside parentheses and then moves to the outside parentheses. In other words, it does this:

5 - 1 = 4

10 - 4 = 6

6 / 2 = 3

RULES ON PARENTHESES

- The computer solves operations enclosed in parentheses first, before solving any others.
- The computer solves the innermost parentheses first. It then works its way out.

COMPUTER MATH EXERCISE

Insert parentheses in the problem below so that the computer prints 28 as the answer.

PRINT 30 - 9 - 8 - 7 - 6

Answer:

```
PRINT 30 - (9 - 8) - (7 - 6)
```

Displaying Large Numbers (E Notation)

Type and run this program to see how the computer displays large numbers:

```
10  X = 1
20  PRINT X;
30  X = X * 10
40  GOTO 20
```

The computer displays large and small numbers using *exponential (E) notation*. The computer displays one billion (1,000,000,000), for example, as 1E+09, which means the number one followed by nine zeros.

If the computer displays a number as 5E-06, you must shift the decimal point, which comes after the 5, six places to the left, inserting zeroes as necessary. Technically, this means $5*10^{-6}$, or 5 millionths (.000005).

Notice that when you run the above program, the computer displays an ?OV ERROR (Overflow Error) at the end of the program. The computer can't handle numbers larger than 1E+38 or smaller than -1E+38.

E notation is simple once you get used to it. You'll find it an easy way to keep track of very large or very small numbers without losing the decimal point.

Learned in Chapter 11

COMMANDS	SYMBOLS	CONCEPTS
GOSUB	( )	Order of operations
GOTO		E Notation
ON GOSUB		
ON GOTO		
RETURN		
REM		

12 / HELP WITH WORDS

BASIC has several functions for working with *strings*. Strings are special constants and variables that store characters. With string functions, you can program the computer to understand *yes* and *no*, or to tell you things in whole sentences!

Counting Characters (The LEN Function)

Type and run this program:

```
10  PRINT "TYPE A SENTENCE"
20  INPUT S$
30  PRINT "YOUR SENTENCE HAS " LEN(S$) " CHARACTERS"
40  INPUT "WANT TO TRY ANOTHER"; A$
50  IF A$ = "YES" THEN 10
```

Impressed? This program uses a function called LEN. The syntax of LEN is:

LEN(*string*) Returns the length of *string*.

In this program, LEN(S$) computes the length of string S$ (your sentence). The computer counts each character in the sentence, including spaces and punctuation marks.

Combining Words (The Concatenation Operator (+))

Erase the program, and run this one, which composes a poem (of sorts):

```
10  A$ = "A ROSE"
20  B$ = " "
30  C$ = "IS A ROSE"
40  D$ = B$ + C$
50  E$ = "AND SO FORTH AND SO ON"
60  F$ = A$ + D$ + D$ + B$ + E$
70  PRINT F$
```

You might encounter two problems when combining strings. Add the following line, and run the program. It shows both problems:

```
80  G$ = F$ + F$ + F$ + F$ + F$ + F$ + F$
```

When the computer gets to Line 80, it prints the first problem with this line: ?OS ERROR IN 80 (Out of String Space).

On startup, the computer reserves only 200 characters of space for working with strings. Line 80 asks it to work with 343 characters. To reserve room for this many characters and more (as many as 500), you can use the CLEAR command. Its syntax is.

CLEAR *n* Clears *n* characters of string space.

Add this line to the start of the program, and run it.

```
5  CLEAR 500
```

Now when the computer gets to Line 80, it has enough string space, but prints the second problem with this line: ?LS ERROR IN 80 (String Too Long).

A string can contain no more than 249 characters. When you want to store more than 249 characters, you need to divide the characters into smaller groups and store each group in its own string.

Twisting Words (The LEFT$ and RIGHT$ Functions)

Now that you can combine strings, try to take a string apart using two new functions: LEFT$ and RIGHT$. Their syntaxes are:

LEFT$(*string,n*) Returns the first *n* characters of *string*.

RIGHT$(*string,n*) Returns the last *n* characters of *string*.

Type and run this program:

```
10  INPUT "TYPE A WORD"; W$
20  PRINT "THE FIRST LETTER IS : " LEFT$ (W$,1)
30  PRINT "THE LAST 2 LETTERS ARE : " RIGHT$ (W$,2)
40  GOTO 10
```

Here's how the program works:

In Line 10 you input string W$. Assume the string is MACHINE:

COMPUTER MEMORY

W$ = MACHINE

In Lines 20 and 30, the computer computes the first **left** letter and the last two **right** letters in the string:

```
                  M  A  C  H  I  N  E
      LEFT$ (W$,1)                       RIGHT$ (W$,2)
```

Run the program a few more times to see how it works.

Now, add this line to the program:

```
5  CLEAR 500
```

The computer will now set aside plenty of space for working with strings. Run the program again. This time input a sentence rather than a word.

PROGRAMMING EXERCISE

How would you change Lines 20 and 30 so the computer will give you the first five letters and the last six letters of your string?

20 ______________________________

30 ______________________________

Answers:

```
20  PRINT "THE FIRST FIVE LETTERS ARE :" LEFT$ (W$,5)
30  PRINT "THE LAST SIX LETTERS ARE :" RIGHT$ (W$,6)
```

Isolating Words (The MID$ Function)

Another function that lets you isolate certain words is MID$. Its syntax is:

MID$(*string,n1,n2*) Returns a substring of *string* beginning with string's *n1*th character and continuing for *n2* characters.

Erase your program and type this one:

```
10  CLEAR 500
20  INPUT "TYPE A SENTENCE"; S$
30  PRINT "TYPE A NUMBER FROM 1 TO " LEN(S$)
40  INPUT X
50  PRINT "THE MIDSTRING WILL BEGIN WITH CHARACTER" X
60  PRINT "TYPE A NUMBER FROM 1 TO " LEN(S$) - X + 1
70  INPUT Y
80  PRINT "THE MIDSTRING WILL BE" Y "CHARACTERS LONG"
90  PRINT "THIS MIDSTRING IS :" MID$(S$, X, Y)
100  GOTO 20
```

Run this program a few times to see if you can deduce how MID$ works.

Here's how the program works.

- In Line 20, assume you input HERE IS A STRING:

YOUR COMPUTER'S MEMORY

S$ → HERE IS A STRING

- In Line 30, the computer computes the length of S$, which is 16 characters. It then asks you to choose a number from 1 to 16. Assume you choose 6.

- In Line 60, the computer asks you to choose another number from 1 to 11 (16-6 + 1). Assume you choose 4.

YOUR COMPUTER'S MEMORY

X = 6
Y = 4

- In Line 90, the computer gives you a "mid-string" of S$ that starts at the 6th character and is four characters long:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
H E R E   I S   A     S  T  R  I  N  G
          ← 4 →
          MID$(S$,6,4)
```

As another example of MID$, run this program:

```
10  INPUT "TYPE A SENTENCE"; S$
20  INPUT "TYPE A WORD IN THE SENTENCE"; W$
30  L = LEN(W$)
40  FOR X = 1 TO LEN(S$)
50  IF MID$(S$,X,L) = W$ THEN 90
60  NEXT X
70  PRINT "YOUR WORD ISN'T IN THE SENTENCE"
80  END
90  PRINT W$ "--BEGINS AT CHARACTER NO." X
```

Here's how the program works:

- In Line 20, you input a word as W$. Assume you input the word IS.
- In Line 30, the computer counts W$'s length: two characters.

YOUR COMPUTER'S MEMORY

S$ = HERE IS A STRING
W$ = IS
L = 2

- In Lines 40-90 (the FOR/NEXT loop), the computer counts each character in S$, starting with Character 1 and ending with Character LEN(S$), which is 16.

  Each time the computer counts a new character, it looks at a new mid-string. Each mid-string starts at character X and is L (2) characters long.

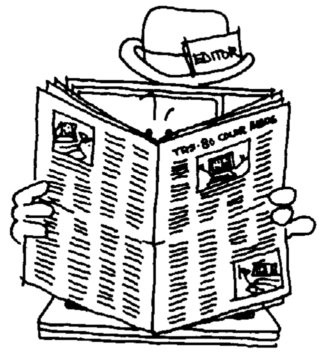
For example, when X equals 1, the computer looks at this mid-string:

```
1
H E R E   I S   A   S T R I N G
←2→
        MID$(S$,1,2)
```

The fourth time through the loop, when X equals 4, the computer looks at this mid-string:

```
            4
      H E R E   I S   A   S T R I N G
            <-2->
          MID$(S$,4,2)
```

When X equals 6, the computer finds IS, the mid-string for which it is searching.



DO-IT-YOURSELF PROGRAM 12-1

Start with a one-line program:

```
10  A$ = "CHANGE A SENTENCE."
```

Add a line that inserts this to the start of A$:

```
IT'S EASY TO
```

Add another line that prints the new sentence:

```
IT'S EASY TO CHANGE A SENTENCE
```

This is our program:

```
10  A$ = "CHANGE A SENTENCE."
20  B$ = "IT'S EASY TO"
30  C$ = B$ + " " + A$
40  PRINT C$
```

DO-IT-YOURSELF PROGRAM 12-2

Add to the above program to make it:

1. Find the start of this mid-string:

   A SENTENCE

2. Delete the above mid-string to form this new string:

   IT'S EASY TO CHANGE

3. Add these words to the end of the new string:

   ANYTHING YOU WANT

4. Print the newly formed string:

   IT'S EASY TO CHANGE ANYTHING YOU WANT

Hint: To form the string IT'S EASY TO CHANGE, you need to get the **left** portion of the string IT'S EASY TO CHANGE A SENTENCE.

Answer:

```
10  A$ = "CHANGE A SENTENCE."
20  B$ = "IT'S EASY TO"
30  C$ = B$ + " " + A$
40  PRINT C$
50  Y = LEN ("A SENTENCE")
60  FOR X = 1 TO LEN(C$)
70  IF MID$ (C$,X,Y) = "A SENTENCE" THEN 90
80  NEXT X
85  END
90  D$ = LEFT$ (C$,X - 1)
100  E$ = D$ + "ANYTHING YOU WANT"
110  PRINT E$
```

DO-IT-YOURSELF CHALLENGER PROGRAM

Write a program that:

- Asks you to input a sentence.
- Asks you to input (1) a phrase within the sentence to delete and (2) a phrase to replace it.
- Prints the changed sentence.

This might take a while, but you have everything you need to write it. Our answer's in the back.

Learned in Chapter 12

FUNCTIONS	COMMAND	SYMBOL
LEN LEFT$ RIGHT$ MID$	CLEAR	+

13 / SAVING PROGRAMS

As you know by now, each time you turn off the computer, your program disappears. To make a permanent copy of a program, you need a cassette recorder or a disk drive.

If you plan to use a cassette recorder, read this chapter. It shows how to use CLOAD, CSAVE, and SKIPF to save your BASIC programs onto cassette tape.

Saving BASIC Programs on Tape (The CSAVE Command)

To save your BASIC programs on tape, use the CSAVE command. Its syntax is:

`CSAVE` "*filename*" Saves a BASIC program named *filename* on cassette tape.

The steps are:

1. Connect the cassette recorder to your color computer. The introduction manual that comes with your color computer shows how to do this.
2. Type a BASIC program into the computer's memory.
3. Insert a blank cassette tape into the recorder.
4. Press the recorder's PLAY and RECORD buttons at the same time until they lock.
5. Choose a name for your BASIC program, which we'll refer to as a *filename*. You can use any *filename* with 8 or fewer letters. Examples of *filenames* are:

   NAME GAMES LETTERS

6. Use CSAVE to save the program on tape.

   For example, to save a program named LETTERS, type:

   ```
   CSAVE "LETTERS" (ENTER)
   ```

   As soon as you press (ENTER), the cassette recorder's motor starts. When the motor stops, BASIC displays the OK prompt on the screen. The program is still in the computer's memory, but it is also saved on cassette tape.

It's a good idea to make more than one copy of a program, preferably on separate cassettes, in case one is lost or inadvertently erased.

Loading BASIC Programs from Tape (The CLOAD Command)

To load your BASIC programs from tape, use the CLOAD command. Its syntax is:

`CLOAD` "*filename*" Loads a BASIC program named *filename* from cassette tape.

The steps are:

1. Be sure the tape is fully rewound and the connections are all in place.
2. Press the PLAY button on the recorder until it locks.
3. Erase any existing programs by typing:

   `NEW` (ENTER)

4. Use the CLOAD command to load the program from tape.

   For example, to load the program named LETTERS from tape, type:

   `CLOAD "LETTERS"` (ENTER)

   As soon as you press (ENTER), the tape recorder's motor starts. BASIC then begins searching for your program. While it is searching, BASIC displays the letter S on the upper left of your screen.

   When BASIC finds your program, it displays the letter F and the *filename* of the program at the top of your screen and begins loading your program. When it has finished loading your program, BASIC displays the OK prompt.

If you are certain your tape has only one program saved on it, you can type CLOAD without a filename. The computer loads the first program it encounters.

If you try to load a program from a blank tape, the color computer searches until the tape ends without giving any indication that the tape is blank. Press RESET to stop the loading process.

Saving Many Programs on Tape (The SKIPF Command)

When you save more than one program on the same tape, you need to use the SKIPF command. SKIPF lets you position the tape at the end of your last program so that you can be sure that you do not save your next program on top of your last program. Its syntax is:

`SKIPF` *filename* Skips through the tape until it finds the end of *filename*.

The steps are:

1. Rewind the tape to the beginning.
2. Press the PLAY button until it locks.
3. Enter the SKIPF command to find the end of the last program you have saved on the tape.

   For example, if the last program you saved on the tape is "LETTERS", type:

   `SKIPF "LETTERS"` (ENTER)

The computer notifies you when it finds the program called LETTERS. When it reaches the end of LETTERS, the recorder's motor stops and your screen displays the OK prompt.

4. Press the RECORD and PLAY buttons, and use CSAVE to save your next program.

   If you can't remember the name of your last program, use an improbable filename such as:

```
SKIPF "X" (ENTER)
```

Hints and Tips

Here are some tips for making good recordings:

- When you're not using the computer to SAVE or LOAD programs, don't leave the recorder's RECORD or PLAY buttons down. Press STOP.
- You can avoid many problems with tapes by using new, high-quality computer tapes.
- If you want to reuse a prerecorded tape, first erase the contents with a bulk tape eraser to be sure you erase everything. Even though the recording process erases the old recording, just enough information can be left to confuse the new recording.
- If you want to save a taped program permanently, break off the Erase Protect tab on the cassette. (See your tape recorder manual.) Without the tab, you can't press the RECORD button on your recorder. This keeps you from accidentally erasing that tape.

Learned in Chapter 13

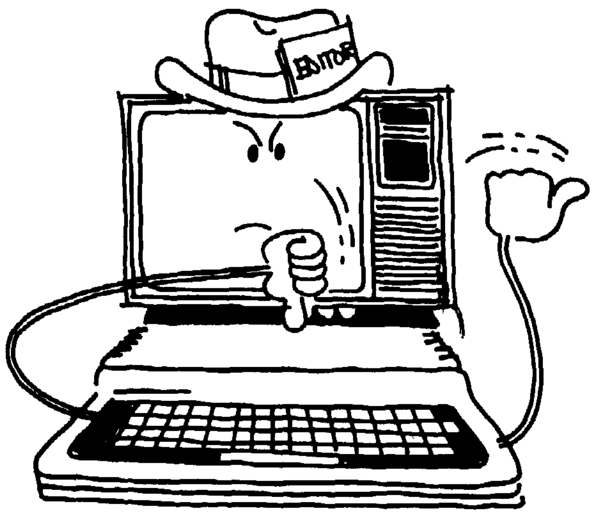
COMMANDS

CSAVE
CLOAD
SKIPF

14 / EDITING PROGRAMS

Up to now, you changed programs by retyping them. This chapter shows how to change programs the easy way, using the EDIT, DELETE, and RENUM commands.

Editing Lines (The EDIT Command)



To get into the *edit mode*, use the EDIT command. The syntax of EDIT is:

EDIT *line number* Enters the edit mode so you can edit *line number*.

In the edit mode, you can use any of the special *edit keys* listed in Table 14.1.

Table 14.1 Edit Keys

(*n* is a number. If you omit *n*, BASIC uses 1.)

Key	Action
L	Lists the line and moves to the start.
*n*C*characters*	Changes the next *n* characters to new *characters*.
I	Inserts characters.
*n*D	Deletes n characters.
H	"Hacks" the rest of the line and puts you in the insert mode.
X	Lets you extend the line.
*n*S*character*	Searches for the *n*th occurrence of *character*.
K	Kills rest of line.
*n*K*character*	Kills (deletes) up to the *n*th occurrence of *character*.
*n*SPACEBAR	Moves *n* spaces forward.
n←	Moves *n* spaces backward.
SHIFT ↑	Return to line mode.

Make a mistake typing a program. Type:

```
50  DABA EFFFUSIVE, GIMPY MUSHY
```

Enter the edit mode. Type:

```
EDIT 50 (ENTER)
```

You see:

```
50  DABA EFFFUSIVE, GIMPY MUSHY
50
```

Start by pressing (L), the List key. The (L) key displays the entire line and puts you back at the start.

Moving the Cursor (The Space Bar, (←), and (S) Keys)

Press **Space Bar** a few times. This key moves you to the right. To move to the left, press (←). Note that while in the edit mode, (←) merely backspaces; it doesn't delete characters.

Move to the start of Line 50 and type (5) space bar. This moves you **5** spaces to the right—**all at once.** Do the same with (←). Type a number, such as (3), and (←) and move that many spaces to the left.

Move to the start of Line 50. To move to the first **E**, press (S) (for "search"). Then type E (the character for which you want to search). There are two ways to move to the second **E**:

- Type (S) E to search for the first **E** after the current cursor position.
- Move back to the start, and type 2 (S) E.

Changing Characters (The (C) Key)

Make your first change to Line 50. Change DABA to DATA:

1. Move to the "wrong" character, the B in DABA.
2. Press (C) for change.
3. Type the new character, in this case, T.
4. To be sure the change is made, press (L) and you see:

```
50  DATA EFFFUSIVE, GIMPY MUSHY
```

Now make the next change: Change GIMPY to GUSHY. This time you'll change **3** characters at a time:

1. Move to the first wrong character, the I in GIMPY.
2. Type 3 (C) for change three characters.
3. Type the three new characters, USH. Line 50 is now:

```
50  DATA EFFFUSIVE, GUSHY MUSHY
```

If this were all you needed to do to Line 50, you could press (ENTER) and get out of the edit mode. As you can see, though, you have more work.

Deleting Characters (The (D) Key)

You need to delete a character, one of the F's in EFFFUSIVE:

1. Move to the excess character, the third F in EFFFUSIVE.
2. Press (D) for delete.
3. It's done. To confirm this, press (L) again:

```
50  DATA EFFUSIVE, GUSHY MUSHY
```

You can delete more than one character at a time. For example, if you type **4** (D), you delete four characters at once.

Inserting Characters (The (I) Key)

You now need to insert some characters: GUSHY needs to be DEMONSTRATIVE OR GUSHY.

1. Move to where you want to insert characters, the space before the G in GUSHY.
2. Press (I) for *insert mode*.
3. Type your insert, **DEMONSTRATIVE OR**

At this point, you're still in the insert mode. For example, if you press Space Bar, you insert a blank space; if you press (L), you insert an L. Therefore, you need to:

1. Press (SHIFT)(↑) to get out of the insert mode.
2. Now, you can press (L) to list the line:

```
50  DATA EFFUSIVE, DEMONSTRATIVE OR GUSHY MUSHY
```

Hacking Characters (The (H) Key)

With hack you alter a line by hacking the end of it and inserting new characters. Try hacking at Line 50:

1. Move to the first character you want hacked off, the M in MUSHY.
2. Press (H) for hack. This deletes the rest of the line and puts you in the insert mode.
3. Type your insert, in this case, **CRUSTY**.
4. Press (SHIFT)(↑) to get out of the insert mode.
5. List the line now (by pressing (L)), and you see:

```
50  DATA EFFUSIVE, DEMONSTRATIVE OR GUSHY CRUSTY
```

Killing Characters
The (K) Key

Kill is almost the opposite of hack. It "kills" everything up to the *n*th occurrence of a character. Suppose that you want to kill the first half of Line 50, everything up to the comma.

1. Move to the start of Line 50 and press these keys: (K)(,)

2. List Line 50 now, and you see:

```
50  ,DEMONSTRATIVE OR GUSHY CRUSTY
```

Extending Characters
(The (X) Key)

Perhaps you want to extend Line 50:

1. Press (X) for extend. The cursor moves to the end of the line and you enter the insert mode.
2. Type your insert: **AND MUSHY**
3. Press (SHIFT)(↑) to get out of the insert mode.

```
50  ,DEMONSTRATIVE OR GUSHY CRUSTY AND MUSHY
```

Deleting Lines
(The DEL Command)

So far, you deleted lines the simple way, like this:

50 (ENTER)

This works fine for one or two lines, but what if you want to delete 50 or 60 lines?

To delete more than one line, you can use the DEL command. The syntax for DEL is:

DEL *line numbers* Deletes the lines specified by the *line numbers*.

For example, to delete Lines 30-50, type:

DEL 30-50 (ENTER)

Renumbering Lines (The RENUM Command)

The RENUM command lets you change a program's line numbers. To see how RENUM works, type this small program:

```
10  PRINT "THIS IS THE FIRST LINE"
20  PRINT "THIS IS THE SECOND LINE"
30  PRINT "HERE'S ANOTHER LINE"
40  GOTO 10
```

Now, renumber it. Type:

```
RENUM 100 (ENTER)
```

List the program, and you see the new line numbers beginning with 100. Line 100 is what we call the *newline*:

```
100  PRINT "THIS IS THE FIRST LINE"
110  PRINT "THIS IS THE SECOND LINE"
120  PRINT "HERE'S ANOTHER LINE"
130  GOTO 100
```

Notice that even the GOTO line number reference is renumbered.

Renumber the program again with a newline of 200. Type:

```
RENUM 200,120 (ENTER)
```

Here, the *newline* is 200, but the renumbering starts with Line 120. Line 120 is what we call the *startline*.

```
100  PRINT "THIS IS THE FIRST LINE"
110  PRINT "THIS IS THE SECOND LINE"
200  PRINT "HERE'S ANOTHER LINE"
210  GOTO 100
```

Renumber the program one more time, giving it an *increment* of 50 between each line:

```
RENUM 300,,50 (ENTER)
```

Here the newline is 300. Since you omitted the startline, BASIC renumbers the entire program. The increment between the lines is 50:

```
300  PRINT "THIS IS THE FIRST LINE"
350  PRINT "THIS IS THE SECOND LINE"
400  PRINT "HERE'S ANOTHER LINE"
450  GOTO 300
```

Here is the syntax of the RENUM command:

RENUM *newline, startline, increment* Renumbers a program.

newline	The first new renumbered line. If you omit *newline*, BASIC uses 10.
startline	Where the renumbering starts. If you omit *startline*, BASIC renumbers the entire program.
increment	The increment between each renumbered line. If you omit *increment*, BASIC uses 10.

Note: RENUM does not rearrange the order of lines.

Try some other variations of this command. Type:

```
RENUM ,,20
```

This renumbers your entire program. The newline is 10, and the increment is 20:

```
10  PRINT "THIS IS THE FIRST LINE"
30  PRINT "THIS IS THE SECOND LINE"
50  PRINT "HERE'S ANOTHER LINE"
70  GOTO 10
```

Type **RENUM 40,30,** (ENTER). Here, the newline is 40, the startline is 30; and the increment is 10:

```
10  PRINT "THIS IS THE FIRST LINE"
40  PRINT "THIS IS THE SECOND LINE"
50  PRINT "HERE'S ANOTHER LINE"
60  GOTO 10
```

Type **RENUM 5,40** (ENTER), and you get a ?FC Error. This is because the result would move Line 40 ahead of Line 10.

Learned in Chapter 14

COMMANDS

EDIT
DEL
RENUM

15 / A POP QUIZ

We have just about reached the end of the first part of this book, so it's time for a "pop quiz." In this chapter, you'll learn how to use the INKEY$ and VAL functions to set up tests for yourself.

Watching the Keyboard (The INKEY$ Function)

By using a word named INKEY$, you can get the computer to constantly watch, time, or test what you're typing. Its syntax is:

INKEY$ Returns the key currently being pressed or, if no key is being pressed, returns nothing (" ").

Type and run this program:

```
10  A$ = INKEY$
20  IF A$ <>"" GOTO 50
30  PRINT "YOU PRESSED NOTHING"
40  GOTO 10
50  PRINT "THE KEY YOU PRESSED IS---" A$
```

INKEY$ checks to see if you're pressing a key. It does this in a split second. At least the first 20 times it checks, you've pressed nothing (" ").

Line 10 labels the key you press as A$. Then the computer makes a decision:

- If A$ equals nothing (" "), it prints YOU PRESSED NOTHING and goes back to Line 10 to check the keyboard again.
- If A$ equals something (anything but " "), the computer goes to Line 50 and prints the key. (The <> notation means "not equal to.")

Add this line and run the program:

```
60  GOTO 10
```

No matter how fast you are, the computer is faster! Erase Line 30 to see what keys you're pressing.

Beat the Computer (An Example of INKEY$)

Type this program:

```
10  X = RND(4)
20  Y = RND(4)
30  PRINT "WHAT IS" X "+" Y
40  T = 0
50  A$ = INKEY$
60  T = T + 1
70  SOUND 128,1
80  IF T = 15 THEN 200
90  IF A$ = "" THEN 50
100  GOTO 10

200  CLS 7
210  SOUND 180, 30
220  PRINT "TOO LATE"
```

Here's how the program works:

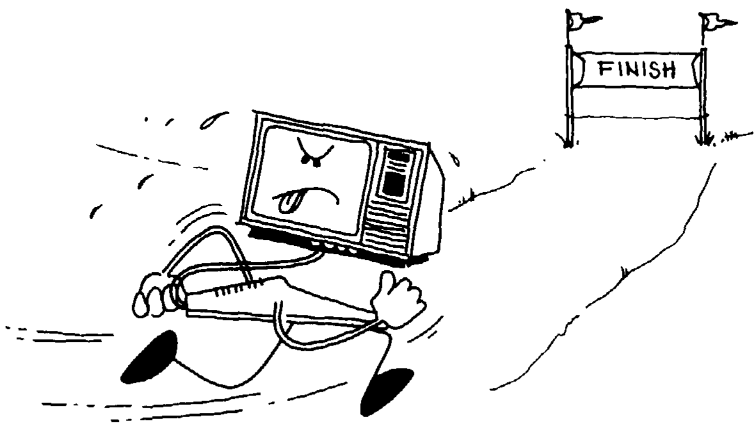
- Lines 10, 20, and 30 tell the computer to print two random numbers and ask you for their sum.
- Line 40 sets T to 0. T is a timer.
- Line 50 gives you your first chance to answer the question.
- Line 60 adds one to T, the timer. T now equals 1. The next time the computer gets to line 60 it again adds one to the timer to make T equal 2. Each time the computer runs Line 60 it adds one to T.
- Line 70 beeps.
- Line 80 tells the computer you have 15 chances to answer. Once T equals 15, time's up. The computer insults you with Lines 200, 210, and 220.
- Line 90 says if you haven't answered yet, the computer needs to go back and give you another chance.
- The computer gets to Line 100 only if you do answer. Line 100 sends it back for another problem.

How can you get the computer to give you three times as much time to answer each question?

Answer:

By changing this line:

```
80  IF T = 45 THEN 200
```



Checking Your Answers (The VAL Function)

How can you get the computer to check to see if your answer is correct? Would this work?

```
100  IF A$ = X + Y THEN 130
110  PRINT "WRONG", X "+" Y "=" X + Y
120  GOTO 10
130  PRINT "CORRECT"
140  GOTO 10
```

If you run this program (and answer on time), you get this error message:

```
?TM ERROR IN 100
```

That's because you can't make a string (A$) equal to a number (X + Y). Somehow, you must change A$ to a number.

BASIC has a function for this called VAL. Its syntax is:

VAL(*string*) Returns the numeric value of *string*.

Change Line 100 by typing:

```
100  IF VAL(A$) = X + Y THEN 130
```

VAL(A$) converts A$ into its numeric value. If A$ equals the string "5", VAL(A$) equals the number 5. If VAL(A$) equals the string "C," VAL(A$) equals the number 0. ("C" has no numeric value.)

To make the program more challenging, change these lines:

```
10  X = RND(49) + 4
20  Y = RND(49) + 4
90  B$ = B$ + A$
100  IF VAL(B$) = X + Y THEN 130
```

Then add these lines:

```
45  B$ = ""
95  IF LEN(B$) <> 2 THEN 50
```

A Computer Typing Test (An Example of INKEY$)

Here's a program that times how fast you type:

```
10  CLS
20  INPUT "PRESS <ENTER> WHEN READY TO TYPE THIS
    PHRASE"; E$
30  PRINT "NOW IS THE TIME FOR ALL GOOD MEN"
40  T = 1
50  A$ = INKEY$
60  IF A$ = "" THEN 100
70  PRINT A$;
80  B$ = B$ + A$
90  IF LEN(B$) = 32 THEN 120
100  T = T + 1
110  GOTO 50
120  S = T/74
130  M = S/60
140  R = 8/M
150  PRINT
160  PRINT "YOU TYPED AT--"R"--WDS/MIN"
```

Line 40 sets T, the timer, to 1.

Line 50 gives you your first chance to type a key (A$). If you're not fast enough, Line 60 sends the program to Line 100 and adds one to the timer.

Line 70 prints the key you typed.

Line 80 forms a string named B$. Each time you type a key (A$), the program adds this to B$. For example, if the first key you type is "N," then:

A$ = "N"
and
B$ = B$ + A$
B$ = "" + "N"
B$ = "N"

If the next key you type is "O," then:

A$ = "O"
and
B$ = B$ + A$
B$ = "N" + "O"
B$ = "NO"

If the third key you type is "W," then:

A$ = "W"
and
B$ = "NO" + "W"
B$ = "NOW"

When the length of B$ is 32 (the length of NOW IS THE TIME FOR ALL GOOD MEN), the program assumes you finished typing the phrase and goes to Line 120 to compute your words per minute.

Lines 120, 130, and 140 compute your typing speed. They divide T by 74 (to get the seconds) and S by 60 (to get the minutes). They then divide the eight words by M to get the words per minute.

Learned in Chapter 15

BASIC WORDS

INKEY$
VAL

PART 2/ HAVING FUN

Have you reached your fill of BASIC basics? In this part of the book, you take a break and learn to:

- Compose a song.
- Draw a picture.
- Play a game with the joysticks.

16 / MUSIC

In this chapter, you use the PLAY command to play some of your favorite tunes.

The syntax for PLAY is:

PLAY *string* Plays *string*. *String* can consist of any of the following options:

note (a letter from "A" to "G" or a number from 1 to 12).

octave (O followed by a number from 1 to 5). If you omit *octave*, the computer uses Octave 2.

note-length (L followed by a numeral from 1 to 255). If you omit *note-length*, the computer uses the current length.

tempo (T followed by a number from 1 to 255). If you omit *tempo*, the computer uses T2.

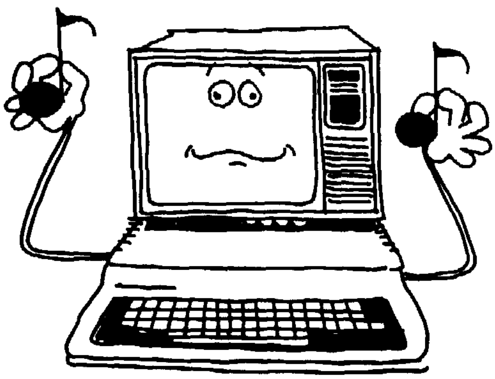
volume (V followed by a number from 1 to 31). If you omit *volume*, the computer uses V15.

pause-length (P followed by a number from 1 to 255).

substrings. Precede substrings with an X and follow them with a semicolon. Example: XA$;

Notes
(The NOTE Option)

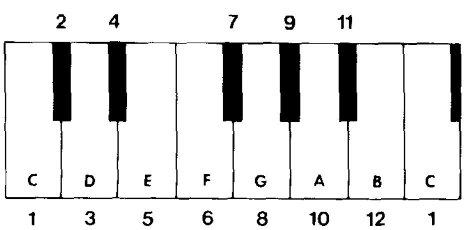
You can specify a musical note in two ways. The first is to enter the note's letter: A, B, C, D, E, F, or G. To indicate a sharp, use the plus (+) or pound (#) sign. To indicate a flat, use the minus (−) sign.



For example, A represents A natural, A# is A sharp, and A− is A flat. Type the following commands to hear what we mean:

```
PLAY "A" (ENTER)
PLAY "A;A#" (ENTER)
PLAY "A-;A;A#;A;A-" (ENTER)
```

The second way to specify a musical note is to use a number in the range 1-12, preceded by the letter N. (You can omit the N, if you wish.)



For example, to hear the full 12-tone scale, run the following Scale program:

```
5  CLS
10  FOR N = 1 to 12
15  PRINT "NOTE#"; N
20  PLAY STR$(N)
30  NEXT N
```

Note: STR$ converts numbers to strings. (If you are really curious, peek ahead to Chapter 37.)

Add a delay in the program so you can compare the numbers to the notes as the scale goes up from 1 to 12 (C to B).

```
25  FOR I = 1 TO 500: NEXT I
```

PLAY does not recognize the notation B# or C−. Substitute C for B# and B for C−.

DO-IT-YOURSELF PROGRAM 16-1

Modify the Scale program so it goes down instead of up.

Whole Notes, Half Notes, Quarter Notes... (The NOTE LENGTH Option)

Because the Scale program does not specify note length, the computer automatically uses quarter notes, the initial *current value*.

You can specify a different note length with L followed by a number in the range 1 to 255. The number 1, for instance, denotes a whole note, 2 a half note, 4 a quarter note, 8 an eighth note, 16 a sixteenth note, and so on.

L*number*	Note Length	Note
L1	Whole note	𝅝
L2	Half note	𝅗𝅥
L3	Dotted quarter note	♩.
L4	Quarter note	♩
L8	Eighth note	♪
L16	1/16 note	𝅘𝅥𝅯
L32	1/32 note	𝅘𝅥𝅰
L64	1/64 note	𝅘𝅥𝅱
.		
.		
.		
L255	1/255 note	

Vary the note lengths to produce a drum roll. Type:

```
PLAY "L2;A;L4;A;A;L2;A;A" (ENTER)
```

Notice that you needn't repeat the L option for each note. PLAY uses the current note value until you enter another L command to tell it otherwise.

Just for fun, try playing three 1/255 notes:

```
PLAY "L255;A;A+;A-" (ENTER)
```

That's staccato!

Dotted Notes (NOTE LENGTH'S "." Notation)

A dotted note tells you to increase the length of the note by one half its normal value. For example, a dotted quarter note is equal to a 3/8 note.

You can play a dotted note by adding a period (.) or a series of periods (...) to the note length (L). Each period increases the note length by 1/2 its normal value. For example:

```
PLAY "L4.; A" (ENTER)
```

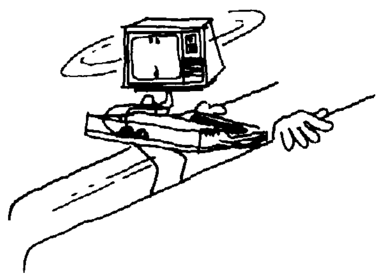
This plays a 3/8 note (1/4 + 1/8 = 3/8).

Try this:

```
PLAY "L4.;A;L8;C;L4.;E;L8;C;E;C;E;C;L4;A" (ENTER)
```

Octaves (The OCTAVE Option)

To change octaves, use the letter O followed by a number in the range 1 to 5. If you don't specify the octave, the computer automatically uses Octave 2, which includes middle C.



For example, try to play a simple C scale:

```
PLAY "CDEFGABAGFEDCBA" (ENTER)
```

What happened? G is the highest note in Octave 2, so when the computer reaches A, it starts over at the beginning of the octave. To get into Octave 3, try this:

```
PLAY "CCDEFG;O3;ABAO2;FEDCBA" (ENTER)
```

Volume
(The VOLUME Option)

To adjust the volume, use V followed by a number in the range 0 to 31. If you don't specify V, the computer uses V15.

For example, run this program:

```
5  CLS
10  PLAY "V5;A; V10;A; V15;A; V20;A; V25;A; V30;A"
20  GOTO 10
```

Press (BREAK) when you've heard enough.

Rests
(The PAUSE Option)

To put a pause between notes, use P followed by a number in the range 1 to 255. Pause lengths correspond to note lengths with one important difference. You can't use dots (periods) with P. To compensate, just type a series of pauses. For example, to get a 3/8 pause, type P4P8.

Change Line 10 in the last program to read:

```
10   PLAY "V5;A; P2; V10;A; P2; V15;A; P2; V20;A; P2;
     V25;A; P2; V30;A; P2"
```

Tempo
(The TEMPO Option)

You can increase or decrease the tempo with T and a number in the range 1 to 255. If you don't specify a tempo, your computer automatically uses T2.

Our program now looks this:

```
5    CLS
10   PLAY "V5;A;P2; V10;A;P2; V15;A;P2; V20;A;P2;
     V25;A;P2; V30;A;P2"
20   GOTO 10
```

Slow down the tempo by changing Line 10 to:

```
10   PLAY "T1; V5;A;P2; V10;A;P2; V15;A;P2; V20;A;P2;
     V25;A;P2; V30;A;P2"
```

Now, speed it up by changing T1 to T15. That's more like it!

Substrings (The SUBSTRING (X) OPTION)

PLAY has a substring option that lets you execute a substring and then return to the original string and complete it.

The execute function takes the following form:

XA$;

Variable A$ contains a string of normal play options. X tells the computer to PLAY the string of options stored in A$.

Rearrange the demonstration program so it executes a substring:

```
5   CLS
10  A$ = "A;A#;A-"
20  B$ = "O5;XA$;"
30  C$ = "O1;XA$;XB$;"
40  PLAY C$
```

Run the program and follow its execution.

Note: Whenever you use the substring function, a semicolon (;) must follow the dollar sign ($). In this example, you can delete all the other semicolons.

One Further Note... (+, –, <, >)

No, we're not going to spring a new note, like H or J, on you. We simply have one final way you can use some of PLAY's options.

With O (octave), V (volume), T (tempo), and L (note length), you can use one of the following suffixes instead of adding a numeral:

Suffix	Purpose
+	Adds 1 to the current value.
–	Subtracts 1 from the current value.
>	Multiplies the current value by 2.
<	Divides the current value by 2.

Use the sample program to learn about these features.

```
5   CLS
10  PLAY "T2"
20  PLAY "A;A#;A-"
30  GOTO 20
```

Notice that Line 10 sets the tempo. Run the program once to get an ear for it. Nothing changed; it's the same as always. Now, insert T in Line 20.

```
20  PLAY "T+;A;A#;A-"
```

Run the program. The plus sign automatically increases T by 1 each time Line 20 plays.

Now reduce the tempo, using a minus sign (−):

```
5    CLS
10   PLAY "T255"
20   PLAY "T-;A;A#;A-"
30   GOTO 20
```

Isn't multiplication faster than addition? In Line 10, reset the tempo to 2. Change T in Line 20 to T>, and let it run.

```
10   PLAY "T2"
20   PLAY "T>; A;A#;A-"
```

You started out with T2, right? The computer multiplied that value by 2 to 4, 4 x 2 to 8, 8 x 2 to 16, and so on until it reached 255.

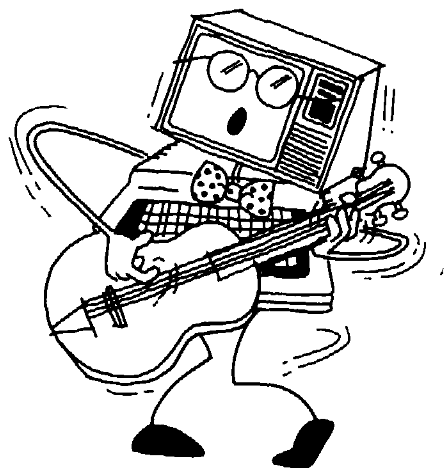
You can slow the tempo quickly by using "<" to divide the current tempo by 2.

```
10   PLAY "T255"
20   PLAY "T<; A;A#;A-"
```

Remember, you can do the same thing with L, V, and O to change the note length, the volume, and the octave.

Roll Over, Beethoven (An Example of the PLAY Command)

After all this hard work, you deserve a serenade. Type the following program and see if you can name this tune.



```
5    CLS
100  A$ = "T5;C;E;F;L1;G;P4;L4;C;E;F;L1;G"
105  B$ = "P4;L4;C;E;F;L2;G;E;C;E;L1;D"
110  C$ = "P8;L4;E;E;D;L2.;C;L4;C;L2;E"
115  D$ = "L4;G;G;G;L1;F;L4;E;F"
120  E$ = "L2;G;E;L4;C;L8;D;D+;D;E;G;L4;A;L1;O3;C"
125  X$ = "XA$;XB$;XC$;XD$;XE$;"
130  PLAY X$
```

Do you recognize the song? Dress it up by adding these lines:

```
10   PRINT @ 96, STRING$ (32,"*")
20   PRINT @ 167, "WHEN THE SAINTS"
30   PRINT @ 232, "GO MARCHING IN"
35   PRINT @ 288, STRING$ (32,"*")
40   FOR X = 1 TO 500: NEXT X
45   CLS
50   PRINT @ 128, "OH WHEN THE SAINTS"
55   PRINT @ 169, "OH WHEN THE SAINTS"
60   PRINT @ 192, "OH WHEN THE SAINTS GO MARCHIN IN"
65   PRINT @ 224, "YES I WANT TO BE IN THAT NUMBER"
70   PRINT @ 256, "WHEN THE SAINTS GO MARCHIN IN"
```

Run the program now and sing along with the color computer. What? You liked it so much you want to hear it again. Okay, add these lines:

```
150  CLS
160  PRINT @ 130, "PLAY IT AGAIN, COCO"
165  FOR X = A TO 500: NEXT X
170  CLS
175  PRINT @ 233, "I'D BE GLAD TO"
180  FOR I = 1 TO 500: NEXT I
185  GOTO 5
```

DO-IT-YOURSELF PROGRAM 16-2

Our rendition of "Saints" sounds fine, but it isn't true New Orleans style. Jazz it up to suit your own musical tastes. Try changing octaves or adding a few sharps or flats.

DO-IT-YOURSELF PROGRAM 16-3

Try some musical arrangements of your own. We've included several in the Sample Programs at the back of the book.

Learned in Chapter 16

BASIC WORDS

PLAY

17 / PICTURES

This chapter has you draw a picture on the low-resolution text screen. You start by setting a tiny dot on the screen. You then set more dots, and finally, you combine these dots into a picture.

Before you start, be aware that this chapter describes the most primitive way of drawing pictures on the screen. Parts 3 and 4 of this book deal with the color computer's sophisticated graphics capabilities.

Setting A Dot (The SET Command)

To set a dot on the screen, you use the SET command. The syntax for SET is:

> SET (*x*,*y*,*c*) Sets a dot on the low-resolution text screen at Column *x*, Row *y*, using Color *c*. *x* is a number in the range 0-63, *y* is a number in the range 0-31, and *c* is a number in the range 0-8.

You can use SET only on the low-resolution text screen. So, move to the low-resolution text screen by typing:

```
WIDTH 32 (ENTER)
```

Then, type and run this program:

```
10  CLS0
20  SET(0,0,3)
30  GOTO 30
```

See the blue dot at the top left corner? To put the dot on the bottom right corner, change Line 20, and run the program:

```
20  SET(63,31,3)
```

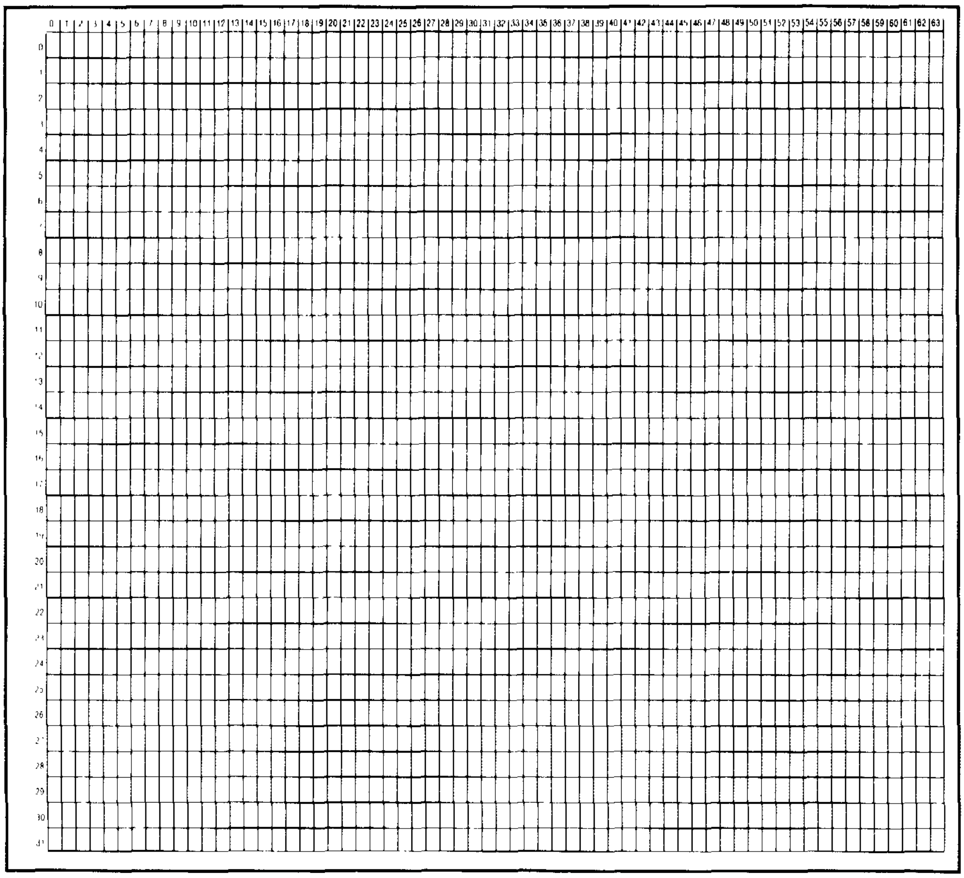
Want to center the dot? Use this for Line 20:

```
20  SET(31,14,3)
```

SET tells the computer to set a dot on your 32 x 16 low-resolution text screen.

BASIC makes it easy to control the screen. SET only needs three numbers to work its magic. Lets see how it all works.

The first two numbers specify the column and row position, using the following SET/RESET grid:



The last number specifies the color, using the palette as shown in this table:

Table 17-1
The SET Command's Use of the Palette

Color #	Palette Slot	Standard Color
0	8	Black
1	0	Green
2	1	Yellow
3	2	Blue
4	3	Red
5	4	Buff
6	5	Cyan
7	6	Magenta
8	7	Orange

SET uses the palette in the same way as CLS. For example, SET(31,14,8) and CLS8 both produce the same color, which, if you are using the standard colors, is orange.

Look at the grid that we showed earlier in this chapter. Notice that the darker lines group the dots into "blocks." Each block contains four dots. For instance, the block in the middle of the grid contains these four dots:

	Horizontal	Vertical
Position	32	14
Position	33	14
Position	32	15
Position	33	15

Each dot within a block must either be the same color or black.

Change line 30 to this:

```
30 SET(30,14,4)
```

Run the program. What happened? Line 30 asked the computer to set two different colored dots (red and blue) within the same block. Because the computer couldn't set them in different colors, it set them both the second color, red.

Type and run this program:

```
30  SET(34,14,4)
```

Because the dot in Position 34, 14 is in a different block, the computer can set the two dots in different colors.

The Computer's Face (An Example of SET)

In this example, we use SET to draw a picture of the computer's face. First, be sure your computer is set up to produce the standard colors.

- If you have an RGB monitor, type PALETTE RGB (ENTER)
- If you have a CMP monitor and used the PALETTE command to alter the palette, type PALETTE CMP (ENTER)

Now, type these lines to create the top and bottom of the head:

```
5   CLS0
10  FOR H = 15 TO 48
20  SET(H,5,5)
30  SET(H,20,5)
40  NEXT H
```

Run the program. You see buff lines, rather than white.

Lines 10 and 40 set up a FOR/NEXT loop for H, making the horizontal positions 15 through 48 for the top and bottom lines. Line 20 sets the top line. Line 30 sets the bottom line.

To set the left and right sides of the head, type these lines:

```
50   FOR V = 5 TO 20
60   SET(15,V,5)
70   SET(48,V,5)
80   NEXT V
```

To make an orange nose, type:

```
90   SET(32,13,8)
```

To make a red mouth, type:

```
100   FOR H = 28 TO 36
110   SET(H,16,4)
120   NEXT H
```

To make blue eyes, type:

```
130   SET(25,10,3)
140   SET(38,10,3)
150   GOTO 150
```

Run the program.

A Blinking Computer (The RESET Command)

By using another command called RESET, you can make the computer blink. The syntax of RESET is:

> **RESET** *x,y* Resets a dot on the low-resolution text screen at Column *x*, Row *y*. *x* is a number in the range 0-63. *y* is a number in the range 0-31.

Type:

```
150  RESET(38,10)
```

Run the program. You now see the same face, except the right eye is missing. RESET erases the dot in Position 38,10. That's the right eye.

To make the eye blink, set and reset the eye by adding this line:

```
160  GOTO 140
```

Reading the Dots (The POINT Function)

Now that you have learned how to SET and RESET points, let's learn how to read them, too. The POINT function lets you read each graphic character on the screen, and tell whether it is SET, RESET, or if there is a text character in that position. The syntax for POINT is:

> **POINT** (*x*,*y*)
> Restores information on point x,y from the low-resolution text screen.
> −1 Point is part of a text character.
> 0 Point is RESET.
> Code Point is SET (Code is color code).

Let's use POINT in a program. First, we'll clear the screen, then draw a horizontal line on it. Then we'll use POINT to read each position on the screen and reverse them. Type in:

```
10  CLS 0
20  FOR Z = 0 TO 63
30  SET (Z,16,2)
40  NEXT Z
100  FOR X = 0 TO 63
110  FOR Y = 0 TO 31
120  A = POINT(X,Y)
130  IF A = -1 THEN 200
140  IF A = 0 THEN SET (X,Y,2):GOTO 200
150  IF A = 2 THEN RESET (X,Y)
200  NEXT Y,X
210  FOR T = 1 TO 500
220  NEXT T
```

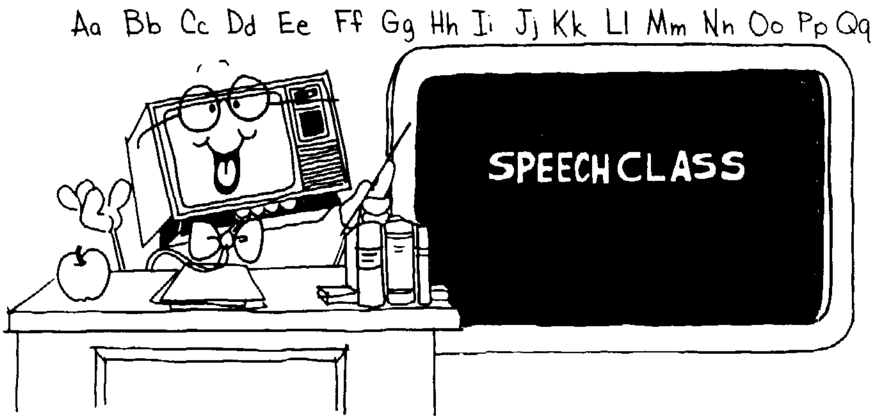
Run the program and watch POINT at work.

Learned in Chapter 17

COMMANDS	FUNCTIONS
SET	POINT
RESET	

18 / THE TALKING COMPUTER TEACHER

Who says the computer can't talk? It's voice, though, will sound similar to your own. You can make the computer talk by using your own tape recorded voice. Your programs will be a lot more interesting and fun when they talk back to you. Let's get started.



Unplug the three pronged cable connecting your tape recorder to the computer. Plug a microphone into the tape recorder if it doesn't have one built in. Put a blank tape into the tape recorder. Press the PLAY and RECORD buttons on the recorder and start talking. Say whatever you want. Press the STOP button on the recorder, and REWIND the tape. Type in this program:

```
 5  CLS
10  INPUT "PRESS <ENTER> TO HEAR THE RECORDING";A$
20  MOTOR ON
30  AUDIO ON
```

Unplug the microphone from the tape recorder. Plug the three-pronged cable from the computer into the tape recorder. Press the PLAY button on the tape recorder. Turn up the volume on your display or amplifier. RUN the program. You will hear your own voice.

`MOTOR ON`	Turns on the tape recorder
`AUDIO ON`	Connects the tape recorder sound to the display speaker or amplifier.

There is a way to program the tape recorder to turn off, but for now press the computer RESET button. The RESET button is on the back right side of the computer, when you are facing it. LIST your program. RESET did not erase it. Add these program lines:

```
35  CLS
40  A$ = INKEY$
50  PRINT @ 225, "PRESS <X> TO TURN OFF RECORDER"
60  IF A$ <> "X" THEN 40
70  AUDIO OFF
80  MOTOR OFF
```

Prepare your tape for playing and RUN the program.

Line 40 tells the computer to read the keyboard without pausing like INPUT.

Line 60 looks at what line 40 reads, and decides whether or not you pressed the X key. If you did not press the X key, the computer goes back to line 40 and looks again. If you did press the X key, the computer goes on to line 70.

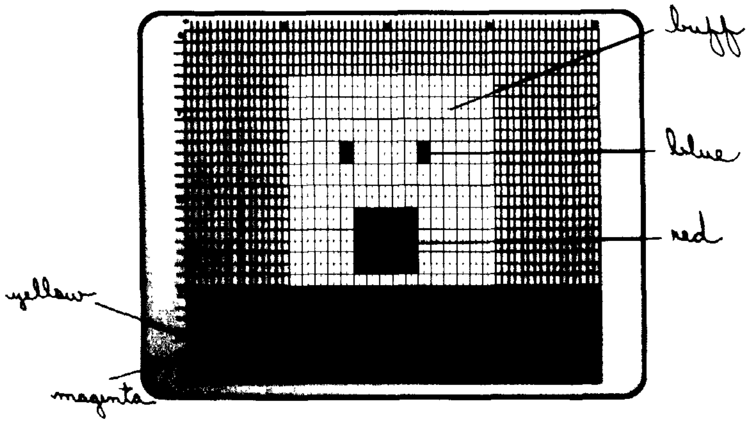
Line 70 turns off the tape recorder sound.

Line 80 turns the tape recorder off.

Now that you understand how it works, you are ready to record the computer teacher. Here is the script:

> "Hi, I'm your talking computer teacher. The first lesson is math. I will give you a series of addition problems. Press the 'W' key..."
>
> (Pause for a few seconds)
>
> "You will hear that every time that you give me an incorrect answer. Press the 'R' key..."
>
> (Pause for a few seconds)
>
> "I will make that sound every time you answer correctly. You will not hear my voice again until you give me three correct answers. Good luck. Press the 'G' key to begin."
>
> (Pause for a few seconds)
>
> "Hello again. I hope that you enjoyed your lesson. Press the 'E' key to turn off the tape recorder."

Finished? The next thing to do is to draw the talking teacher. Here is our grid of what it will look like:



Draw the mouth first. Erase memory and type:

```
  5  CLS 0
200  FOR H = 26 TO 35
210  FOR V = 16 TO 21
220  SET(H,V,4)
230  NEXT V,H
```

That's a closed mouth. To make it talk, type:

```
500  RESET(30,18):RESET(30,19)
510  GOTO 200
```

RUN the program. Now draw the face. Type:

```
100  FOR H = 16 TO 47
110  FOR V = 4 TO 23
120  SET(H,V,5)
130  NEXT V,H
```

Draw the body. Type:

```
140  FOR H = 0 TO 60 STEP 4
150  FOR V = 24 TO 31
160  SET(H,V,2):SET(H+1,V,2)
170  SET(H+2,V,7):SET(H+3,V,7)
180  NEXT V,H
```

Draw the eyes. Type:

```
300  FOR V = 10 TO 11
310  SET(24,V,3):SET(25,V,3)
320  SET(36,V,3):SET(37,V,3)
330  NEXT V
340  PRINT @ 0,"THE TALKING COMPUTER TEACHER"
```

RUN the program. Make the eyes blink. Type:

```
505  IF RND(4) = 4 THEN SET(24,10,5):SET(37,10,5)
```

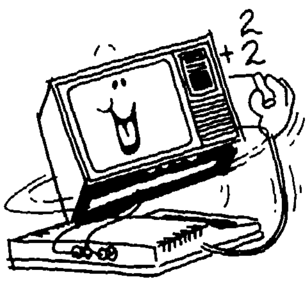
RUN the program. That is what the talking teacher looks like. Now, teach the teacher to talk. Type:

```
 400  MOTOR ON
 410  AUDIO ON
 420  A$ = INKEY$
 430  IF A$ = "G" THEN MOTOR OFF:END
 440  IF A$ = "W" THEN MOTOR OFF:GOSUB 2000
 450  IF A$ = "R" THEN MOTOR OFF:GOSUB 3000
2000  FOR T = 176 TO 86 STEP -10
2010  SOUND T,1
2020  NEXT T
2030  RETURN
3000  FOR T = 86 TO 176 STEP 10
3010  SOUND T,1
3020  NEXT T
3030  RETURN
```

Rewind the tape in your tape recorder. Connect the three-pronged cable from the tape recorder to the computer. Press the PLAY button on the tape recorder. RUN the program. Do what the talking teacher tells you to do.

Is everything working so far? When you press the W key, you should hear ascending tones. Pressing the R key makes descending tones. If you press the G key, the program ends. Now, program the computer to give you arithmetic problems. Type:

```
 430  IF A$ = "G" THEN MOTOR OFF:GOSUB 1000
 460  IF A$ = "E" THEN MOTOR OFF:END
1000  X = RND(100):Y = RND(100)
1010  PRINT @ 0,"WHAT IS" X "+" Y
```



Notice line 1015. It sets the PRINT position for what you type in line 1020.

```
1015  PRINT @ 20,
1020  INPUT A
1030  IF A = X + Y THEN GOSUB 3000:C = C + 1
1040  IF A <> X + Y THEN GOSUB 2000:PRINT @ 0,"WRONG -
      THE ANSWER IS" X + Y
1050  IF C = 3 THEN RETURN
1060  FOR P = 1 TO 500:NEXT P
1070  GOTO 1000
```

Rewind the tape and press PLAY, then RUN the program. Watch, listen, and learn with the talking computer teacher.

Learned in Chapter 18

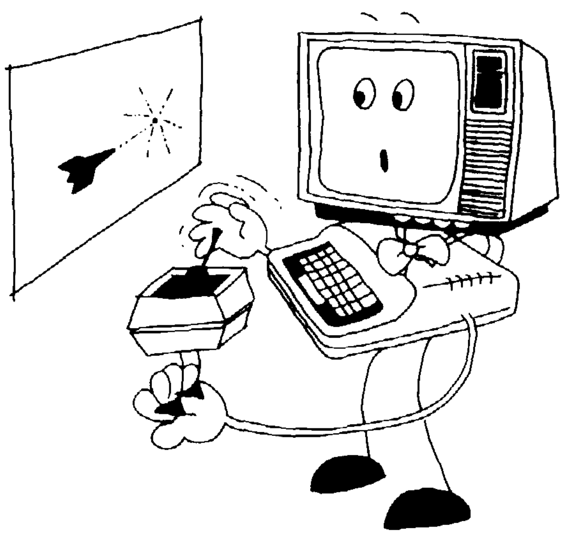
COMMANDS

AUDIO
MOTOR

19 / JOYSTICKS

If you have joysticks, connect them now by plugging them into the back of your computer. They fit in only the correct slots, so don't worry about plugging them into the wrong places.

This chapter shows how to use joysticks in a BASIC program. If you do not have joysticks, skip this chapter.



The Floating Switches (The JOYSTK Function)

You use the JOYSTK function to find the position of the joysticks' *floating switches*. The syntax of JOYSTK is:

JOYSTK(*n*) Returns the position of n, a floating switch on one of the joysticks. *n* is a number from 0 to 3:

- n = 0 Right joystick's horizontal coordinate.
- n = 1 Right joystick's vertical coordinate.
- n = 2 Left joystick's horizontal coordinate.
- n = 3 Left joystick's vertical coordinate.

To see how JOYSTK works, run this short program:

```
10  WIDTH 32
20  PRINT @ 0, JOYSTK(0);
30  PRINT @ 5, JOYSTK(1);
40  PRINT @ 10, JOYSTK(2);
50  PRINT @ 15, JOYSTK(3);
60  GOTO 20
```

See the four numbers on your screen? They're the horizontal and vertical positions of the two joysticks' floating switches.

Grasp the right joystick's floating switch. (The right joystick is the joystick that is connected to the RIGHT JOYSTICK socket on the back of the computer.) Keeping it in the center, move it from left to right. The first number on the screen changes from 0 to 63, going through all the numbers in between.

Move the left joystick's floating switch from left to right. The third number on the screen changes.

Now, move the floating switches up and down, keeping them in the center. Moving the right joystick up and down changes the second number from 0 to 63. Moving the left joystick up and down changes the fourth number from 0 to 63.

This is how the computer reads the joysticks' positions:

JOYSTK(0) and JOYSTK(1) read the right joystick's positions:

- JOYSTK(0) reads the horizontal (left to right) coordinate.
- JOYSTK(1) reads the vertical (up and down) coordinate.

JOYSTK(2) and JOYSTK(3) read the left joystick's positions:

- JOYSTK(2) reads the horizontal coordinate.
- JOYSTK(3) reads the vertical coordinate.

Whenever you read any of the joysticks, you must read JOYSTK(0). To find out for yourself, delete Line 50 and run the program. It works almost the same, except it doesn't read JOYSTK(3), the vertical position of your left joystick.

Delete Line 20 and change Line 60:

```
60  GOTO 30
```

Run the program. Move all the switches around. This time the program doesn't work at all. The computer won't read any coordinates unless you first have it read JOYSTK(0). Type these lines and run the program:

```
20  A = JOYSTK(0)
60  GOTO 20
```

Although the computer is not printing JOYSTK(0)'s coordinates, it's still reading them. Therefore, it can read the other joystick coordinates. Whenever you want to read JOYSTK(1), JOYSTK(2), or JOYSTK(3), you first need to read JOYSTK(0).

Painting with Joysticks (An Example of JOYSTK)

Type and run this program:

```
10  CLS(0)
20  H = JOYSTK(0)
30  V = JOYSTK(1)
40  IF V > 31 THEN V = V - 32
80  SET(H,V,3)
90  GOTO 20
```

Use the revolving switch of your right joystick to paint a picture. (Move the switch slowly so that the computer has time to read its coordinates.)

Line 20 reads H, the horizontal position of your right joystick. This can be a number in the range 0 to 63.

Line 30 reads V, its vertical position. This also can be a number in the range 0 to 63. Since the highest vertical position on your screen is 31, Line 40 is necessary: It makes V always equal a number in the range 0 to 31.

Line 80 sets a blue dot at H and V.

Line 90 goes back to get the next horizontal and vertical positions of your joysticks.

This program uses only the right joystick. Perhaps you could use the left one for color. Add these lines and run the program:

```
50  C = JOYSTK(2)
60  IF C < 31 THEN C = 3
70  IF C > = 31 THEN C = 4
80  SET(H,V,C)
```

Move your left joystick to the right, and the computer makes C equal to 4. The dots it sets are red. Move it to the left, and the computer makes C equal to 3. The dots it sets are blue.

The Joystick Buttons (The BUTTON Function)

Want to use your joystick buttons? Add these lines to the program:

```
90  IF BUTTON(0)=1 THEN 10
100  GOTO 20
```

Run the program and start "painting." Press the right button when you want to clear the screen and start again. (If you have a joystick with two buttons, press the button on the right side of the joystick.)

The syntax for the BUTTON function is:

BUTTON (*n*) Returns a 1 if Button n is on, and a 0 if button n is off. *n* is a number from 0-3:

- $n=0$ Right Button 1 (or single-button joystick)
- $n=1$ Right Button 2
- $n=2$ Left Button 1 (or single-button joystick)
- $n=3$ Left Button 2

Learned in Chapter 19

FUNCTIONS

JOYSTK
BUTTON

PART 3 / DRAMATIC IMAGES

Are you ready for a dramatic leap? In this part of the book you learn to use a new screen designed solely for graphics, the *low-resolution graphics screen*.

Using the low-resolution graphics screen, you'll find it easy to:

Draw a circle

Paint a box

Move a picture

And much more!

20 / LET'S GET TO THE POINT

One of the most exciting features of the color computer is its ability to display precise, varied, and easy-to-use graphics.

Just how easy is it to display these graphics? Well, let's start with the most basic element, a point or a dot, and build from there.

But First, A Word About Color . . . (Using Palette to Set up Standard Colors)

We do a lot of talking about colors in this part of the book, and it would help if your colors agree with ours. So, to avoid confusion, take a minute to be sure your computer's palette is set to produce the standard colors.

- If you are using an RGB monitor, type **PALETTE RGB** (ENTER)
- If you are using a CMP monitor and have altered the palette, type **PALETTE CMP** (ENTER)

Now, proceed with your first dot.

Your First Dot (or Point) (The PSET Command)

Your computer makes it simple to put a dot on the screen. Type the following program and see:

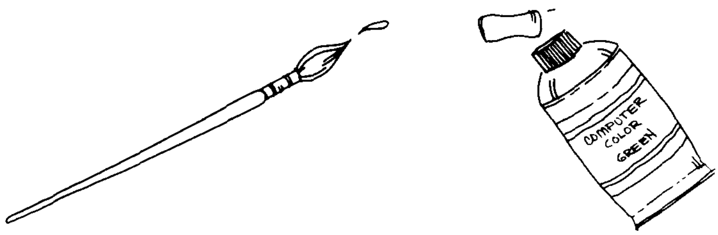
```
2   WIDTH 32
5   PMODE 1,1
10  PCLS
20  SCREEN 1,1
30  PSET (10,20,8)
40  GOTO 40
```

But What About the Color? (Specifying Colors with PSET)

By now, you probably figured out that you can change colors by changing *c* to a different number in the range 0-8.

Within limits, this is true. However—and it's a big however—you can produce only four colors.

There's a good reason for this, which we cover in Chapters 21 and 22 when we discuss *PMODES* and *color sets*. For now, don't worry if you can't always get the color you want.



In your current PMODE and color set, you can get these colors:

Color Number	Palette Slot	Standard Color
Color 1 or 5	Slot 4	Buff
Color 2 or 6	Slot 5	Cyan
Color 3 or 7	Slot 6	Magenta
Color 0, 4, 8	Slot 7	Orange

For example, in your current PMODE and color set, Color 1 and Color 5 both specify the color stored in Slot 4. If your palette is set up to produce the standard colors, this color is buff.

If you want, try changing the dots' color to cyan (2 or 6), and magenta (3 or 7). Then, change the color back to orange (0, 4, or 8) before proceeding.

Now You See It...Now You Don't (The PRESET Command)

Any guesses how to turn off a dot? Here's a hint: It's easy, and it has to do with color.

You don't really turn off the dot; you simply change its color so it blends into the background. You do this with a new command, PRESET (point reset). PRESET "knows" you want to use the background color, so you don't need to give the color.

PRESET (*x,y*) resets a point on the current low-resolution graphics screen to the current background color.

x is the horizontal position (0 to 255).
y is the vertical position (0 to 191).

DO-IT-YOURSELF PROGRAM 20-1

Do you remember the RND (random) function from Part 2? If not, review it; then write a short program that fills the screen with random dots of random colors.

Finding a Point (The PPOINT Function)

PPOINT is closely related to PSET and RESET. It lets you find the color of any dot on the screen.

PPOINT (*x*,*y*) tells what color a point is on the current graphics screen

x is the point's horizontal position (0 to 255).
y is the point's vertical position (0 to 191).

This example shows how PPOINT can be handy to include in a program:

```
2    WIDTH 32
5    PMODE 3,1
10   PCLS
15   SCREEN 1,1
30   X = RND(10)
35   Y = RND(10)
40   C = RND(8)
50   PSET (X,Y,C)
60   IF PPOINT (5,5)=8 THEN GOTO 105
70   GOTO 30
105   CLS
110   PRINT @ 100, "POSITION (5,5) IS NOW COLOR 8"
```

The computer fills a 10 x 10 square (in the upper left corner of the screen) with random colored dots. When the dot in Position (5,5) is Color 8, the computer displays the message **POSITION (5,5) IS NOW COLOR 8**.

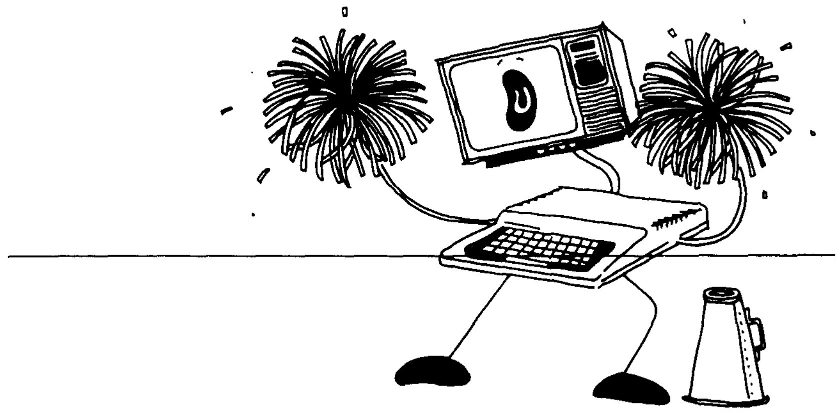
Learned in Chapter 20

COMMANDS	FUNCTION
PSET	PPOINT
PRESET	

21 / HOLD THAT LINE!

So, now you can put a dot on the screen—even several dots. But what kind of starting point is that, you might wonder, when you're eager to create some "real" graphics.

To answer that question, think of some of your very first drawings on paper. Remember the drawings you made by connecting a bunch of dots? That is exactly how your computer draws. You tell it which dots to connect, and it draws a line.



Drawing a Line (The LINE Command)

One way to tell the computer to draw a line between dots is to use the LINE command. To see LINE at work, modify the program that set the dots. (For the sake of convenience, call the program Lines.)

First change Line 30 as follows:

```
30  LINE (0,0) - (255,191),PSET
```

Then, delete Line 35 by typing:

```
35 (ENTER)
```

Your program now reads:

```
2   WIDTH 32
5   PMODE 1,1
10  PCLS
20  SCREEN 1,1
30  LINE (0,0)-(255,191),PSET
40  GOTO 40
```

Now, run the program. The screen shows a line that runs from the upper left to the lower right corner.

How about changing the direction of the line so that it runs from the lower left to the upper right corner?

You probably already figured out this one, but—just in case—here's the new Line 30:

```
30  LINE (0,191)-(255,0),PSET
```

Drawing Two Lines (An Example of LINE)

What about intersecting lines?

Insert the original Line 30 that drew the first line. (First, renumber it as Line 25.) Then run the program. Does your screen display two lines intersecting in the center?

In fact, you can put as many lines on the screen as you want, once you learn the syntax. Here it is:

LINE *(x1,y1)-(x2,y2),a,b* draws a line or a box on the current graphics screen.

> *(x1,y1)* is the line's start point.
> *(x2,y2)* is the line's end point.
> *a* is either PSET (set) or PRESET (reset).
> *b* is either B (box) or BF (box filled). This is optional.

Note: You can omit the start point as discussed below.

At times, you might want to start a second line where the first line ends. To do so, omit the start point. For example:

```
30  LINE (0,0)-(255,191),PSET
35  LINE -(191,0),PSET
```

Line 20 draws a line from (0,0) to (255,191). Line 30 then draws another line, this one from (255,191) to point (191,0).

Erasing a Line (The PSET and PRESET Options)

Maybe you noticed that LINE does not have a color option. Instead, it includes PSET and PRESET options that let you specify whether you want to use the foreground or background color.

Take another look at the program lines that created the intersecting lines:

```
30  LINE (0,0)-(255,191),PSET
35  LINE (0,191)-(255,0),PSET
```

From your experience turning on and off dots in Chapter 20, can you guess what would happen if you change PSET to PRESET? Try it and see. Type:

```
30  LINE (0,0)-(255,191),PRESET
```

If you guessed that the line that ran from the upper left to the lower right would disappear, you were right.

- PSET sets the line using the foreground color.
- PRESET resets the line to the background color.

Before proceeding, change the PRESET parameter in line 30 back to PSET.

Boxing a Line (The B Option)

We've almost made it through LINE, but a few items still need to be (to B?) covered.

B stands for box. With low-resolution graphics, you can make a box without writing a separate program line for each side. All you have to do is specify two opposing corners of the box, and add ,B to the statement. Then when you run the program, your computer creates a box instead of a line.

To illustrate, call your Lines program back into service.

```
2   WIDTH 32
5   PMODE 1,1
10  PCLS
20  SCREEN 1,1
25  LINE (0,0)-(255,191),PSET
30  LINE (0,191)-(255,0),PSET
40  GOTO 40
```

As is, the program creates two lines that intersect in the center of the screen. Delete Line 30 and add the suffix ,B to Line 25.

```
25  LINE (0,0)-(255,191),PSET,B
```

Now see what happens when you run the program. Did you box yourself in?

DO-IT-YOURSELF PROGRAM 21-1

Write a program that creates a box with a pair of lines intersecting in the center. We tell you why these are the only available colors when we discuss PMODE and SCREEN in the next chapters.

Fill A Box (The BF Option)

We're almost at the end of the LINE, so let's try to finish.

If you refer to the format of LINE, you can see you have the option of adding F to the optional suffix ,B.

F lets you fill the box with the foreground color. Try it. Change Line 25 as follows:

```
25  LINE (0,0)-(255,191),PSET,BF
```

How about that! You see a big box filled with color.

DO-IT-YOURSELF PROGRAM 21-2

Ready to try your own Lines program? Can you build a house? Start with Lines 5, 10, and 20 of the Lines program and take it from there. Be sure to add:

- A front door, of course.
- At least one window.
- A chimney.

The overall design is up to you (Cape Cod, Ranch, or whatever), but we've included a sample house (good view, no pets) program in the back of the book. Don't worry about doorknobs; we add those later.

Be sure to save this program on cassette, since you will need it later.

DO-IT-YOURSELF PROGRAM 21-3

This is a real challenge.

As you know, a straight line is the shortest distance between two points. Well, put a few extra miles between our two points. Use LINE to draw a crooked line.

To get started, use Lines 5, 10, and 20 from the Lines program.

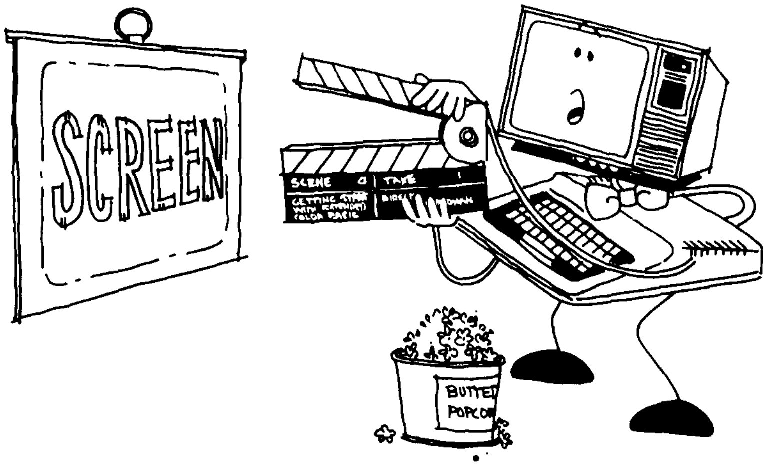
Learned in Chapter 21

COMMAND

LINE

22 / THE SILVER SCREEN

Are you ready to find out about another command? If so, turn down the lights, because we're about to raise the curtain on the silver screen.



Displaying the Graphics Screen (The SCREEN Command)

Take a look at the Lines program for a second. Concentrate on the SCREEN statement in Line 20:

```
2   WIDTH 32
5   PMODE 1,1
10  PCLS
20  SCREEN 1,1
25  LINE (0,0)-(255,191),PSET
30  LINE (0,191)-(255,0),PSET
40  GOTO 40
```

SCREEN tells the computer to display a screen. What kind of screen it displays depends on the instructions you give it:

- First, you tell the computer whether to display a text or a graphics screen.
- Second, you tell the computer what color set to use.

`SCREEN` *type, color set* displays the current graphics or text screen

type is 0 (text screen) or 1 (graphics screen)
color set is 0 or 1

Note: If *type* or *color set* is any positive number greater than 1, your computer uses 1.

In the Lines program, change Line 20 to:

```
20  SCREEN 0,0
```

Then, run the program. Does your computer "hang up"? (Press BREAK to regain control.)

Actually, the computer ran Lines, the same as before. This time, it did not show you the graphics screen. You asked to see the text screen instead.

Now change Line 20 to:

```
20  SCREEN 1,0
```

Notice that you have the graphics screen again, but this time, the color set is changed.

At first glance, it appears that you have only two color choices, 0 and 1. Actually, you're choosing from a much greater variety. You're switching color sets, not individual colors.

Tables 22.1-22.2 shows the two color sets and how the computer uses the palette in each color set.

Table 22.1
Color Set 0

Color Number	Palette Slot	Standard Color
Color 1 or 5	Slot 0*	Green
Color 2 or 6	Slot 1	Yellow
Color 3 or 7	Slot 2	Red
Color 0, 4, 8	Slot 3**	Blue

* Default background color.
** Default foreground color.

Table 22.2
Color Set 1

Number	Slot	Color
Color 1 or 5	Slot 4*	Buff
Color 2 or 6	Slot 5	Cyan
Color 3 or 7	Slot 6	Magenta
Color 0, 4, 8	Slot 7**	Orange

* Default background color.
** Default foreground color.

For example, in Color Set 0, the computer uses the color stored in Slot 1 as the default background color. Assuming your computer's palette is set to produce the standard colors, Color Set 0 produces a green background.

DO-IT-YOURSELF PROGRAM 22-1

Write a program that switches from the text screen to the graphics screen. You might want to put a loop in the program so that it changes the color set after it loops through the program. This way, you can see all the SCREEN features at work.

Changing the Foreground and Background Colors (The COLOR Command)

Notice that we use the word *default* to describe the foreground and background colors. The COLOR command lets you change these defaults.

The syntax for COLOR is:

> **COLOR** *c1,c2* sets the foreground and background colors on the current graphics screen
>
> > *c1* is the foreground color (0 to 8).
> > *c2* is the background color (0 to 8).

For example, insert Line 6 into the Lines program:

```
6   COLOR 6, 7
```

Run the program. The foreground color is Color 6. The background color is Color 7.

Do you want to reverse the colors? Change Line 6 to:

```
6   COLOR 7, 6
```

Before proceeding, delete Line 6 from your program.

Start With the Right Text Screen (The WIDTH Command)

Now, look at another command in Lines:

```
2   WIDTH 32
```

This line is purely a precaution. We want to be sure that when you run Lines, you are **not** at one of the high-resolution text screens (the 40 x 24 or 80 x 24 screen).

Why is this important? BASIC is unable to produce low-resolution graphics on a high-resolution text screen.

To see for yourself, delete Line 2, move to a high-resolution text screen, and run the program.

```
DEL 2 (ENTER)
WIDTH 40 (ENTER)
RUN (ENTER)
```

Your computer appears to hang up. It ran Lines, but it was unable to execute the SCREEN command from the high-resolution text screen.

Move back to the low-resolution text screen, and run the program.

```
2   WIDTH 32 (ENTER)
```

Clearing the Graphics Screen (The PCLS Command)

The Lines program looks like this:

```
2   WIDTH 32
5   PMODE 1,1
10  PCLS
20  SCREEN 1,1
25  LINE (0,0)-(255,191),PSET
30  LINE (0,191)-(255,0),PSET
40  GOTO 40
```

Look at Line 10. It contains the PCLS command. This command simply clears the graphics screen. (It serves the same function for the graphics screen as CLS does for the text screen.)

Here is the syntax for PCLS:

PCLS *color* clears the current graphics screen

> *color* is 0-8. If you omit the *color*, the computer clears the screen to the current background color.

The Lines program doesn't make use of PCLS's *color* option, so the computer uses the current background color. Retype Line 10, and run the program.

```
10  PCLS 6
```

The background is now Color 6.

Learned in Chapter 22

COMMANDS

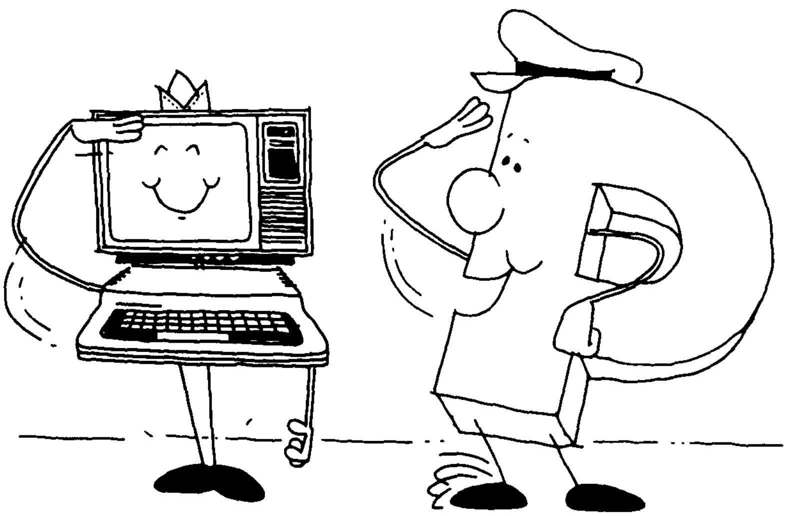
SCREEN
WIDTH
COLOR
PCLS

23 / MINDING YOUR PMODES

Whenever you write a low-resolution graphics program, you need to consider these three features:

- Screen positions—You can use as many as 256 x 192 positions at a time.
- Colors—You can use as many as four colors at a time.
- Screens—You can use as many as eight screens at a time.

The more you use of one feature (such as screen positions), the less you can use of the other two features (colors and screens).



PMODE, the unknown command in the Lines program, sets the features you want to use. You can choose from among five PMODE settings, shown in Table 23.1.

Table 23.1 / PMODE Settings

	Positions	Colors	Screens
PMODE 4	256 x 192	2	2
PMODE 3	128 x 192	4	2
PMODE 2	128 x 192	2	4
PMODE 1	128 x 96	4	4
PMODE 0	128 x 96	2	8

Lines in PMODE 4 (Changing PMODE Settings)

Bring back Lines and see what it looks like in a different PMODE. In case you've forgotten Lines, here it is:

```
2   WIDTH 32
5   PMODE 1,1
10  PCLS
20  SCREEN 1,1
```

Now change from PMODE 1 to PMODE 4.

```
5 PMODE 4,1
```

Run the program. You can see two feature changes right away:

- The lines are much finer because you shifted from a 128 x 192-position PMODE to a 256 x 192-position PMODE.
- The color changes because you shifted from a 4-color PMODE to a 2-color PMODE.

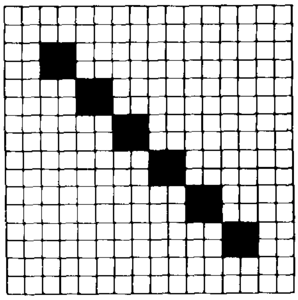
Changing Available Positions (PMODE Positions)

Notice that when you shift to a different PMODE, you do not have to change the positions of any of your dots. BASIC lets you use the same screen grid (a 256 x 192 grid), no matter how many screen positions you actually have available.

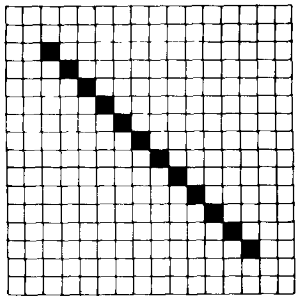
For example, (128,96) is **always** the center of the screen, no matter which PMODE you're using, and (256,192) is always the bottom-right corner of the screen. The way that BASIC uses the screen grid, depends on which PMODE you are using.

- In a 128 x 96-position PMODE, BASIC sets **four** dots for each dot you specify. For example, if you ask BASIC to set Dot (0,0), it also sets (1,0), (1,1), and (0,1).
- In a 128 x 192-position PMODE, BASIC sets **two** dots for each dot you specify. For example, if you ask BASIC to set Dot (0,0), it also sets (1,0).
- In a 256 x 192-position PMODE, BASIC sets **one** dot for each dot you specify.

Therefore, a diagonal line in a 128 x 96-position PMODE looks more like a stairstep on the screen than one drawn in a 256 x 192 position PMODE.

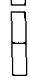


128 x 96-Position PMODE



256 x 192-Position PMODE

The number of different screen positions you can use in a 128 x 96-position PMODE is only one-fourth what you can use in a 256 x 192-position PMODE.

	Screen Positions Available	Size of Each Dot
High resolution	256 x 192	
Medium resolution	128 x 192	
Low resolution	128 x 96	

The "Graphics Screen Worksheets," in the "Odds and Ends" section show the positions available in each PMODE.

Changing Color Modes (PMODE Colors)

The 2-color mode, like the 4-color mode, has two color sets that you can use. Tables 23.2 and 23.3 show the two color sets you can use in a 2-color PMODE.

Table 23.2
Color Set 0

Color Number	Palette Slot	Standard Color
Color 2, 4, 6, 8	Slot 8*	Black
Color 1, 3, 5, 7	Slot 9**	Green

* Default background color.
** Default foreground color.

Table 22.3
Color Set 1

Color Number	Palette Slot	Standard Color
Color 2, 4, 6, 8	Slot 10*	Black
Color 1, 3, 5, 7	Slot 11**	Buff

* Default background color.
** Default foreground color.

Compare these to the tables in the last chapter, which show the two color sets you can use in a 4-color PMODE.

PMODE Boxes (An Example of Changing PMODES)

Here is a program that shows a box cycle through each mode. Notice that with each mode the box's lines go from thick to thin, and its colors go from two colors to four colors.

```
2   WIDTH 32
5   FOR MODE = 0 TO 4
10  PMODE MODE,1
20  PCLS
30  SCREEN 1,1
40  LINE (75,50)-(125,100),PSET,B
50  FOR Y = 0 TO 500: NEXT Y
60  NEXT MODE
70  GOTO 5
```

This is PMODE's syntax. Chapter 25 shows how to use the second parameter, *start page*.

PMODE *mode,start page* sets the current graphics screen in graphics memory

mode specifies the features you want to use in graphics memory. If you omit mode, the computer uses the last *mode* or (if none) Mode 2.

start page specifies on which page in graphics memory to start a graphics screen. If you omit *start page*, the computer uses the last *start page* or (if none) Page 1.

Therefore, if you omit PMODE, the computer uses PMODE 2,1.

Learned in Chapter 23

COMMAND

PMODE

24 / A DIFFERENT USE OF COLOR

In all our graphics programs so far, we stuck to the standard colors.

In this chapter, we introduce nonstandard colors. Before reading this chapter, you might want to refer to Chapter 8 to refresh your memory on color codes and the palette.

Lines in Hot Pink (Graphic's Use of the Palette)

Take another look at Lines. Use the version that has a PCLS6 command.

```
2   WIDTH 32
5   PMODE 1,1
10  PCLS6
20  SCREEN 1,1
25  LINE (0,0)-(255,191),PSET
30  LINE (0,191)-(255,0),PSET
40  GOTO 40
```

With the standard palette, the PCLS6 command makes the screen cyan. But, by storing hot pink in the palette slot that creates Color 6, the PCLS6 command makes the screen hot pink.

Try it. The steps are:

1. Note which PMODE and color set you are using.

   The above version of Lines is using a 4-color PMODE (PMODE 1,1) with Color Set 1 (SCREEN 1,1)

2. In the PMODE and color set you are using, find out which palette slot creates Color 6. (See Tables 22.1-22.2.)

3. Look up the color code for hot pink in the "Color Codes" section in the back of the book. Store this code in the proper palette slot with this program line:

   ```
   8 PALETTE 5, Color Code
   ```

4. Run the program. The crossing lines are orange, as before, but the background screen is now hot pink.

DO-IT-YOURSELF PROGRAM 24-1

Change Line 5 so you are in a 2-color PMODE:

```
5 PMODE 4,1
```

Now, figure out how to make the screen hot pink.

... And a Dash of Charcoal Brown (An Example of Medium Graphics and the Palette)

All low-resolution graphics commands use the palette in the same way. For example, change the PCLS command in Line 10 to:

```
10  COLOR 7,6
```

Lines now looks like this:

```
2   WIDTH 32
5   PMODE 1,1
10  COLOR 7,6
20  SCREEN 1,1
25  LINE (0,0)-(255,191),PSET
30  LINE (0,191)-(255,0),PSET
40  GOTO 40
```

The COLOR command makes the foreground magenta (Color 7) and, because of the way you altered the palette above, it makes the background hot pink (Color 6).

In this example, we alter the palette so that the foreground is charcoal brown, rather than magenta.

The steps are:

1. Note which PMODE and color set you are using.

   The above version of Lines is still using a 4-color PMODE with Color Set 1.

2. In the PMODE and color set you are using, find out which palette slot creates Color 7.

   As shown in Table 22.2, the palette slot that creates Color 7 is Slot 6.

3. Look up the code for charcoal brown.

4. Store this code in the proper palette slot.

   Type:

   ```
   9 PALETTE 6, Color Code
   ```

5. Run the program. The background is hot pink and the crossing lines are charcoal brown.

DO-IT-YOURSELF 24-2

Change Line 20 so you are in Color Set 0:

```
20 SCREEN 1,0
```

Now, figure out how to make the foreground charcoal brown.

Learned in Chapter 24

COMMAND

PALETTE

25 / FINDING THE RIGHT PAGES

In writing this book, we "stored" chapters in pages. Some chapters require more pages, some fewer.

In the same sense, BASIC stores low-resolution graphics screens in memory pages. Some screens require more memory pages; some fewer.

PMODE is what determines how many memory pages it takes to draw a screen. As shown in Table 25.1, a screen drawn in a higher PMODE requires more memory pages than a screen drawn in a lower PMODE.

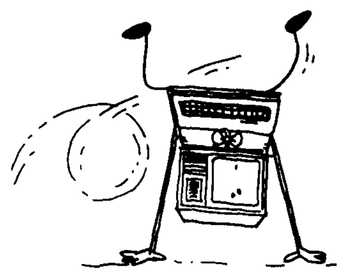


Table 25.1 / Pages Required for Graphics Screens

Screen	Pages Required
PMODE 4 Screen	4 pages
PMODE 3 Screen	4 pages
PMODE 2 Screen	2 pages
PMODE 1 Screen	2 pages
PMODE 0 Screen	1 page

As you learn shortly, PMODE also determines which pages are stored on a screen.

Changing Pages (The PMODE Start-Page Parameter)

See what happens if you store the Lines screen on a different group of pages.

```
5   PMODE 1,1
10  PCLS
20  SCREEN 1,1
25  LINE (0,0)-(255,191),PSET
30  LINE (0,191)-(255,0),PSET
40  GOTO 40
```

Focus on PMODE. As you know, the first PMODE parameter tells the computer to start a PMODE 1 screen. And, as Table 25-1 tells you, a PMODE 1 screen requires two pages. The second parameter tells the computer to start the screen on Page 1. So, the 2-page Lines screen is on Pages 1 and 2.

To put the 2-page Lines screen on Pages 3 and 4, type:

```
5  PMODE 1,3
```

Run the program. You see the same screen, but the screen is now on different pages.

Lines on Different Screens (Changing the Current Graphics Screen)

What about storing two screens, one on Pages 1 and 2, and the other on Pages 3 and 4? Type this program:

```
5  PMODE 1,1
10  PCLS                              stores screen on
25  LINE (0,0)-(255,191),PSET         Pages 1-2

27  PMODE 1,3
28  PCLS                              stores screen on
30  LINE (0,191)-(255,0),PSET         Pages 3-4

40  GOTO 40
```

The first part of the program starts a PMODE 1 screen on Pages 1-2. It clears this screen and puts a line on it.

The next part of the program starts another PMODE 1 screen on Pages 3-4. It clears this screen and puts a line on it.

Run the program and you won't see either screen, because there's no SCREEN statement. So, add this command:

```
35  SCREEN 1,1
```

The program now looks like this:

```
5  PMODE 1,1
10  PCLS                              stores screen on
25  LINE (0,0)-(255,191),PSET         Pages 1-2

27  PMODE 1,3
28  PCLS                              stores screen on
30  LINE (0,191)-(255,0),PSET         Pages 3-4

35  SCREEN 1,1

40  GOTO 40
```

Run the program. You see only one screen, the *current graphics screen*, which is the screen stored on Pages 3-4.

The computer uses your most recent PMODE command to determine the current graphics screen. In the above program, the most recent PMODE command is Line 27. It specifies the screen stored on Pages 3-4.

Insert another PMODE line right before SCREEN:

```
32  PMODE 1,1
```

Run the program again. Now you see a different current graphics screen, the screen stored on Pages 1-2.

DO-IT-YOURSELF 25-1

Have Color BASIC display a PMODE 2 screen that starts on Page 2. Any guesses as to what you'll see? Change Line 32 to PMODE 2,2 and run the program. Because PMODE 2 requires two pages, you see what's on Pages 2-3. And, because this is PMODE 2, you see this screen in two colors with low-resolution.

Flipping Screens (An Example of PMODE Start-Page Parameter)

Animators make cartoons by drawing many still pictures and then "flipping" through them. So, here's the moment you've been waiting for! This program flips screens to show two lines in motion:

```
5   PMODE 1,1
10  PCLS                          stores Page 1-2 screen
25  LINE (0,0)-(255,191),PSET

27  PMODE 1,3
28  PCLS                          stores Page 3-4 screen
30  LINE (0,191)-(255,0),PSET

32  PMODE 1,1
34  SCREEN 1,1                    displays Page 1-2
36  FOR I=1 TO 200:NEXT I         screen

38  PMODE 1,3
40  SCREEN 1,1                    displays Page 3-4
42  FOR I=1 TO 200:NEXT I         screen

44  GOTO 32
```

Adding Pages (The PCLEAR Command)

You can use a maximum of eight pages of graphics memory, pages 1-8. However, when you first start up, BASIC gives you only half that amount, Pages 1-4. For example, make this change to Lines:

```
5   PMODE 1,4
```

To remedy the problem, insert Line 4, and you now have all eight pages.

```
4   PCLEAR 8
```

PCLEAR lets you reserve one to eight pages of memory. If you use PCLEAR, it must be the first or second command in your program (after CLEAR, if you use CLEAR).

PCLEAR *pages* reserves pages of graphics memory

pages is the amount of graphics memory to reserve (0-8)

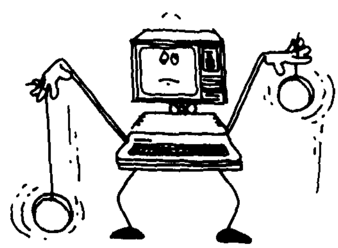
On startup, the computer automatically reserves four pages. Use PCLEAR to reserve more or fewer pages.

You might wonder why we don't use PCLEAR 8 all the time. PCLEAR 8 decreases program memory. Sometimes you need more **program** memory; other times you need more **graphics** memory. PCLEAR gives you the choice.

Up and Down, Up and Down (An Example of PCLEAR)

You can use the pages reserved with PCLEAR to store several screens. If you draw different pictures on each screen, you can flip through them with PMODE for exciting animation.

```
10  PCLEAR 8
20  FOR P=1 TO 8
30  PMODE 0,P
40  PCLS
50  LINE (128,0)-(138,10+(P-1)*15),PSET
60  CIRCLE (128,P*15),15
70  NEXT P
80  FOR P=1 TO 8:GOSUB 110:NEXT P
90  FOR P=7 TO 1 STEP -2:GOSUB 110:NEXT P
100  GOTO 80
110  PMODE 0,P
120  SCREEN 1,0
130  FOR T=1 TO 10:NEXT T
140  RETURN
```



With the exception of CIRCLE (see the next chapter), you know all the features used by this program.

Copying Pages The PCOPY Command

Using PCOPY (Page Copy) you can copy one page of graphics memory to another. Here is the format for PCOPY:

PCOPY *page 1* TO *page 2* copies *page 1* to *page 2*

For example, if you want to copy Page 3 to Page 8, type:

```
PCOPY 3 TO 8
```

One advantage of PCOPY is it can shorten your programs by eliminating repetition.

Keep in mind that PCOPY copies one graphics' memory page. Unless you're in PMODE 0, this is not one screen. For example, in PMODE 4, the above statement copies only one-fourth of a screen.

DO-IT-YOURSELF PROGRAM 25-2

The following program displays four squares that are on four different memory pages on the screen at the same time. Run it: then shorten the program using PCOPY.

```
4  PCLEAR 8
5  PMODE 3,4
10  PCLS
11  SCREEN 1,1
12  LINE (110,20)-(120,30),PSET,B
20  PMODE 3,3
21  SCREEN 1,1
22  LINE (110,20)-(120,30),PSET,B
30  PMODE 3,2
31  SCREEN 1,1
32  LINE (110,20)-(120,30),PSET,B
40  PMODE 3,1
41  SCREEN 1,1
42  LINE (110,20)-(120,30),PSET,B
50  GOTO 50
```

DO-IT-YOURSELF PROGRAM 25-3

Using LINE and *start page,* simulate a lightning storm. Put "crazy lines" at random positions on different pages. Then, switch back and forth between pages.

Learned in Chapter 25

COMMANDS

PMODE
PCLEAR
PCOPY

26 / GOING IN CIRCLES

Does all this talk about SCREEN, PMODE, and PCLEAR have you going in circles? If so, you haven't seen anything yet!

You can create a full circle, a partial circle, or an ellipse (an oblong circle) with one command, CIRCLE. Here is its syntax:

`CIRCLE` *(x,y),r,c,hw,start,end* draws a circle on the current graphics screen

> *x* is the horizontal position of the centerpoint (0 to 255).
> *y* is the vertical position of the centerpoint (0 to 191).
> *r* is the radius in screen points. (If *r* is larger than 95, the circle flattens against the edges of the screen.)
> *c* is any available color (0-8). If you omit *c*, the computer uses the foreground color.
> *hw* is the height to width ratio (0 to 255). If you omit *hw*, the computer uses 1.
> *start* is the starting point (0 to 1). If you omit *start*, the computer starts at 0.
> *end* is the ending point (0 to 1). If you omit *end*, the computer uses 1.
>
> If *start* equals *end* or if you omit both *start* and *end*, the computer draws the complete ellipse.

With CIRCLE, you only need to know the center of the circle and the *radius* (the distance from the center to the edge of the circle).

Bring your Lines program back into service.

```
5   PMODE 1,1
10  PCLS
20  SCREEN 1,1
25  LINE (0,0)-(255,191),PSET
30  LINE (0,191)-(255,0),PSET
40  GOTO 40
```

Delete Line 25, and change Line 30 as follows:

```
30  CIRCLE (128,96),95
```

Run the program. Your screen shows a scruffy circle. Are you wondering why the circle isn't truly round? Look at Line 5 and you see that the computer is in PMODE 1. (Only 128 x 96 positions are available).

Change PMODE1 to PMODE4 (256 x 192) as follows:

```
5   PMODE 4,1
10  PCLS
20  SCREEN 1,1
30  CIRCLE (128,96),95
40  GOTO 40
```

Run the program. Now, that's a circle!

DO-IT-YOURSELF PROGRAM 26-1

Using the program above, generate a bull's-eye. You can do this one of two ways:

- Add a separate program line for each concentric circle but use a common center (*h,v* coordinate).
- Use a FOR...NEXT loop with a STEP 10.

DO-IT-YOURSELF PROGRAM 26-2

Do you still have the program for the house you built? Use CIRCLE to put a doorknob on the front door. To add full detail to the circle, run the program in PMODE 4.

Coloring the Circle (The Color Option)

After you decide the circle's radius, choose its color. Using 2-color PMODE, you don't have much choice. If you use 4-color PMODE (PMODE 1 or 3), you have many options.

Your program reads:

```
5   PMODE 1,1
10  PCLS
20  SCREEN 1,1
30  CIRCLE (128,96),95
40  GOTO 40
```

First, make the circle a more manageable size:

```
30  CIRCLE (128,96),30
```

For a little variety, change the color to Color 6:

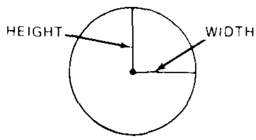
```
30  CIRCLE (128,96),30,6
```

It's as easy as that! In fact, you can make the circle any of the available colors.

Squeezing a Circle (The Height/Width Option)

Did you ever take a Hula-Hoop, bicycle tire, or buggy wheel and squeeze it with both hands to form an ellipse?

Similarly, you can change a circle on your screen to an ellipse by specifying a height/width ratio (*hw*).



When you specify *hw*, the width of the circle remains the same. The height, however, is determined by *hw*.

- If *hw* is 1, the height is the same as the width.
- If *hw* is greater than 1, the height is greater than the width.
- If *hw* is less than 1, the height is less than the width.

For example, in this program, the *hw* is 1; so the program draws a round circle.

```
2  WIDTH 32
5  PMODE 4,1
10  PCLS
20  SCREEN 1,1
30  CIRCLE (128,96),30,,1
40  GOTO 40
```

In this program, the *hw* is 3, so the program draws a vertical ellipse:

```
30  CIRCLE (128,96),30,,3
```

In this program, the *hw* is .25, so the program draws a horizontal ellipse:

```
30  CIRCLE (128,96),30,,.25
```

Note that the above lines do not specify the color (*c*). We still have to include a comma, to indicate that we are omitting *c*. Otherwise, the computer mistakes *hw* for *c*.

Change Line 30 in the following ways, and run the program:

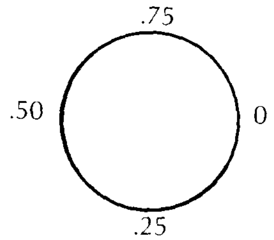
```
30  CIRCLE (128,96),30,,0
```

```
30  CIRCLE (128,96),30,,100
```

When *hw* equals 0, the ellipse is infinitely wide (a horizontal line). And, when *hw* equals a large number, the ellipse is infinitely long (a vertical line).

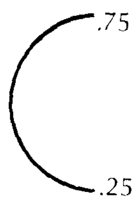
Splitting the Circle (The Arc Option)

Suppose you want to draw only part of a circle (an *arc*). To do this, specify the *start* and *end* of the arc, following the chart below. Keep in mind that the computer always draws clockwise.



Note: To draw an arc, you must specify *hw*. For a normal arc, use *hw* 1.

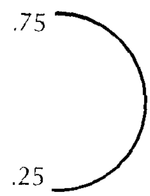
For example, suppose, you want to draw this arc:



To do so, use this command:

```
30  CIRCLE (128,96),30,1,1,.25,.75
```

Now change the command to draw this arc:



Is this your new Line 30?

```
30  CIRCLE (128,96),30,1,1,.75,.25
```

DO-IT-YOURSELF PROGRAM 26-3

Has night fallen on the house you built? If so, you might want to put a crescent moon in the corner. This requires two intersecting arcs and some trial and error on your part.

DO-IT-YOURSELF PROGRAM 26-4

Maybe it's cold, as well as dark, around your house. If so, show smoke coming out the chimney. (Use CIRCLE to generate a spiral that simulates the smoke.)

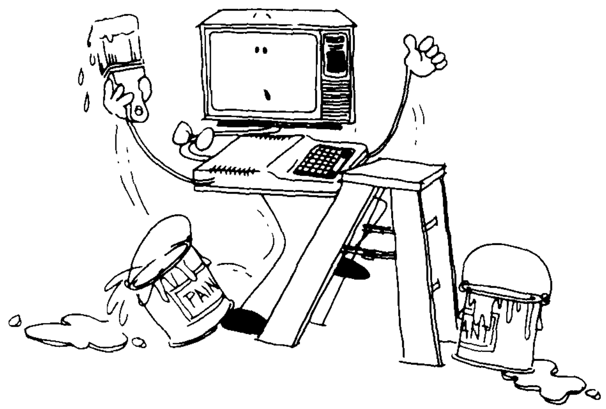
Learned in Chapter 26

COMMAND

CIRCLE

27 / THE BIG BRUSH-OFF

You might think we forgot that this is a color computer. So far, it's been a little dab of color here and a splotch or two of color there. You can never create a masterpiece that way! Well, it's time now to paint the screen.



The PAINT command lets you "paint" any shape with any available color. Its syntax is:

PAINT *(x,y),c,b* paints the current graphics screen

x is the horizontal position (0 to 255) of the point at which painting is to begin.
y is the vertical position (0 to 191).
c is the color (0 to 8).
b is the border color at which painting is to stop (0 to 8).

If the computer reaches a border other than that of the specified color, it paints over that border.

Change the Lines program as follows:

```
2   WIDTH 32
5   PMODE 3,1
10  PCLS
20  SCREEN 1,1
30  LINE (0,0)-(255,191),PSET
40  LINE (0,191)-(255,0),PSET
50  CIRCLE (128,96),90
60  PAINT (135,125),8,8
70  GOTO 70
```

Can you predict the results? Lines 30 and 40 draw intersecting lines. Line 50 draws a circle, and the circle's center is where the two lines intersect. That part is easy, but what about PAINT in Line 60?

If you guess that the computer goes to Position (135,125) and paints the screen Color 8 until it reaches a border that is Color 8, you're right!

Delete Line 30, and run the program. Now that you redefined the borders, the computer paints half the circle.

DO-IT-YOURSELF PROGRAM 27-1

Can you paint the entire circle? You can do this two ways. One involves adding a line; the other involves deleting a line.

DO-IT-YOURSELF PROGRAM 27-2

Do you still have your house? It probably looks fairly plain. Why not spruce it up with some paint?

DO-IT-YOURSELF PROGRAM 27-3

Add a garage to your house, then use PAINT to raise and lower the garage door. Since the painting action always goes up first, this takes a little refining on your part. Add a delay before and after the opening. With CIRCLE, add the sun.

Learned in Chapter 27

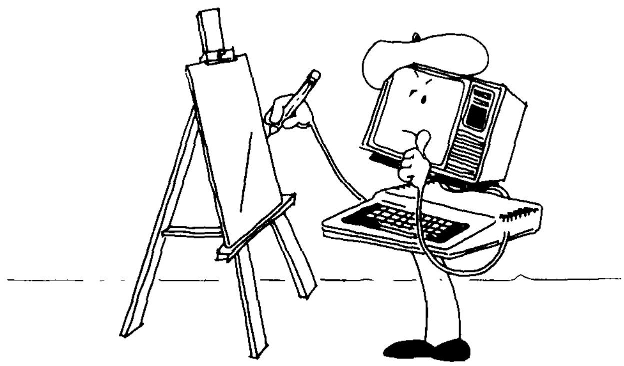
BASIC WORDS

PAINT

28 / DRAWING SHAPES

You already know how to create lines, ellipses, and boxes. How about learning a shortcut?

The shortcut is the DRAW command. DRAW lets you draw any shape by specifying direction, angle, and color—all in the same command!



Here is the syntax of DRAW:

DRAW *shape* draws a *shape* on the current graphics screen

shape is a string that can include the following motion subcommands, modes, and options:

Motion Subcommands
BM*x,y* = Move to Position *x,y*
U*n* = Up *n* points
D*n* = Down *n* points
L*n* = Left *n* points
R*n* = Right *n* points
E*n* = 45-degree angle *n* points
F*n* = 135-degree angle *n* points
G*n* = 225-degree angle *n* points
H*n* = 315-degree angle *n* points
X = Execute a substring and return

Modes
S*n* = Scale *n* (1-62)
C*n* = Color *n* (0-8)
A*n* = Angle *n* (0-3)

Options
N = No update of draw position
B = Blank (no draw, just move)

Note: If you omit the start point, the computer uses the last DRAW position or, if you haven't previously used DRAW, at the center of the screen. If you omit the number of points it should draw, the computer draws 1 point.

Drawing a Shape (The U, D, L, and R Motion Subcommands)

With DRAW, all you have to specify is where to start drawing, which direction to draw, and how far to do so.

Change the Lines program so it looks like this:

```
2    WIDTH 32
5    PMODE 3,1
10   PCLS
20   SCREEN 1,1
25   DRAW "BM128,96;U25;R25;D25;L25"
40   GOTO 40
```

Presto! Can you guess why the square's lower left corner is at (128,96)?

Line 25 tells the computer to start drawing at (128,96), draw up (U) 25 points, right (R) 25, down (D) 25, and finally, left (L) 25.

Stand the square on one of its corners. To do so, substitute E, F, G, and H for U, R, L, and D in Line 25:

```
25   DRAW "BM128,96;E25;F25;G25;H25"
```

The first line angles off at 45 degrees; the next, at 135 degrees; the next, at 225 degrees; and the last, at 315 degrees.

There is one slight hitch in drawing angles. If you are in PMODE 0 or 1 and draw an angle that has an odd-number length and at least one odd-number coordinate (*x,y*), Lines F and H have a slight hitch at the midpoint. If both coordinates are even-numbered, Lines E and G have the hitch. This is normal.

DO-IT-YOURSELF PROGRAM 28-1

You already know your computer is the star of the show. Now, prove it by drawing a star.

Starting at a Relative Position (The "+" and "−" Signs)

In the above examples, you told the computer to start at an *absolute position*. You can also tell the computer to start drawing at a *relative position*.

For example, bring out this version of Lines:

```
2    WIDTH 32
5    PMODE 3,1
10   PCLS
20   SCREEN 1,1
25   DRAW "BM128,96;U25;R25;D25;L25"
40   GOTO 40
```

If you wish to create a second square at a position relative to the first square, you could add this line:

```
30  DRAW "BM+15,+15;U25;R25;D25;L25"
```

The + is an *offset sign*. When the computer executes Line 30, the current draw position is (128,96), which is the last draw position in Line 25. So, to draw the new square, the computer starts at (128+15,96+15) or (143,111).

Another offset sign you can use is -. Change Line 30 as follows:

```
30  DRAW "BM+15,-15;U25;R25;D25;L25"
```

Run the program. The *start* point of the new square is (128+15,96-15) or (143,81).

Try this line:

```
30  DRAW "BM+15,15;U25;R25;D25;L25"
```

If you use an offset sign for the *x* coordinate, but omit the offset sign for the *y* coordinate, the computer uses a + offset sign for the *y* coordinate.

DO-IT-YOURSELF PROGRAM 28-2

After all this heated activity, you're probably ready to cool off. So why don't you use DRAW to create an ice cube?

You can generate the entire cube using DRAW, or you can incorporate a couple of LINE commands within the program. Try to use both absolute and relative motion.

Reducing and Enlarging a Shape (The Scale Mode)

What if the figures you draw turn out to be too big or too small?

The solution's easy. Use the DRAW command's scale mode.

S*n* lets you scale a display

n is a number in the range 1-62 that indicates the scale factor in units of 1/4 as shown here:

1 = 1/4 scale
2 = 2/4 scale
3 = 3/4 scale
4 = 4/4 (full) scale
5 = 5/4 (125%) scale
8 = 8/4 (double) scale
12 = 12/4 (triple) scale
etc.

If you omit *n*, the computer uses 4 (4/4 = 1).

After you enter S*n*, the computer scales all motion subcommands accordingly.

Make your refined Lines draw a single square again. Do this by deleting Line 30 and changing Line 25 as follows:

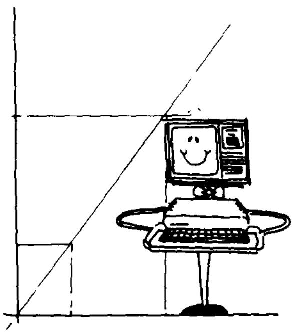
```
25  DRAW "S2;BM128,96;U25;R25;D25;L25"
```

Run the program. The square in the lower left corner is half the size you specified.

To see how small or large a square can be, run the following program:

```
5  PMODE 4,1
10  PCLS
20  SCREEN 1,1
25  FOR SCALE = 1 TO 62
30  S$ = "S" + STR$(SCALE) + ";"
35  DRAW S$ + "BM10,100U20R20D20L20"
40  NEXT SCALE
50  GOTO 50
```

Don't make the mistake of thinking that the smallest square is the one specified in Line 35. The one we specified is the fourth one from the edge.



When you use the scale-down option, the computer rounds the resulting line length to the nearest whole number, if it is not already a whole number.

For example, "S2U25R25D25L25" results in a 12 1/2 x 12 1/2 square. The computer draws a 13 x 13 square.

Coloring a Shape (The Color Option)

DRAW's C option lets you specify the color of what you are drawing.

First, list the Lines program:

```
5  PMODE 3,1
10  PCLS
20  SCREEN 1,1
30  DRAW "S2;BM128,96;U25;R25;D25;L25"
40  GOTO 40
```

Go back to full scale either by changing S2 to S4 or by deleting S2. Then, right inside the first set of quotation marks in Line 30, insert:

```
C6;
```

Run the program. The square is now Color 6.

Replace the C6 (in program Line 30) with C8. and run the program. The square is now Color 8.

You can insert C*n* anywhere in the DRAW command. All actions that follow are the color you specify. For instance, change Line 30 to read:

```
30  DRAW "C8; BM128,96;U25;R25;C6; D25;L25"
```

Run the program. The program displays a 2-color square. The first two lines drawn are Color 8. The second two are Color 6.

Drawing Angles (The Angle Mode)

The A mode lets you specify the angle at which a line is to be drawn. After you include A in the DRAW command. the computer draws all subsequent lines with the angle displacement specified by A*n* until you specify otherwise.

Here is the syntax of the A subcommand:

A*n* lets you specify the angle of a line

n is the angle code (0 to 3) All angles are measured clockwise.

0 = 0 degrees
1 = 90 degrees
2 = 180 degrees
3 = 270 degrees

If you omit A*n*, the computer uses A0.

For example, your program now reads:

```
5   PMODE 3,1
10  PCLS
20  SCREEN 1,1
30  DRAW "C6;BM128,96;U25;R25;D25;L25"
40  GOTO 40
```

Change Line 30:

```
30  DRAW "A0;BM128,96;U25"
```

Run the program. Your screen displays a vertical line that is 25 points long. Now change Line 30:

```
30  DRAW "A1;BM128,96;U25"
```

Run the program. The line is now horizontal.

Drawing a Blank (The Blank Option)

If you want the next line you draw to be a "blank" or an invisible line, include the B option.

For example, let's say you are drawing letters of the alphabet and are ready for the letter C, which is nothing but a square with the right side blank. Change Line 30 as follows so the program generates such a figure:

```
30  DRAW "BM128,96;U25;R25 ;B ;D25;L25"
```

Run the program. Remember, only the line immediately following the B is blank.

DO-IT-YOURSELF PROGRAM 28-3

Print your name on the screen using DRAW. This means you have to stay in the graphics screen. Sure, it would be easier to write your name on the text screen, but you can't have "true" text and graphics at the same time.

Drawing from the Same Point (The No Update Option)

Another of DRAW's many features is N, the *no update* option. N tells the computer to return to its original (current) position after it draws the next line. To see this, change Line 30 to read:

```
30  DRAW "M128,96; N; U25; N; R25; N; D25; N; L25;"
```

Run the program. The computer draws a 25-point line straight up from 128,96. It then returns to 128,96, draws the next line, returns, draws the next, and so on. As a result, four lines radiate from the center of the screen, each in a different direction (up, right, down, and left).

DO-IT-YOURSELF PROGRAM 28-4

Using DRAW's N option (and CIRCLE), draw a pie that has eight pieces. Once you've done that, cut out a piece of the pie and put it over to one side.

Using Substrings (The X Subcommand)

The *string* following DRAW can be either a *constant*, as in the previous examples, or a variable.

For example, add Line 25 and change Line 30 as follows:

```
25  A$="BM128,96;C8;U25;R25;D25;L25"
30  DRAW A$
```

Run the program. Following the instructions stored in A$, the computer draws the 25 x 25 square, starting at 128,96.

The X subcommand lets you execute one DRAW string within another DRAW string. To do this, leave Line 25 as it is so it defines A$. Then, change Line 30. The two lines read:

```
25  A$="BM128,96;C8;U25;R25;D25;L25"
30  DRAW "BM95,50;U25;R25; XA$; D25;L25"
```

Run the program. The computer starts drawing at 95,50 a line that extends up (U25) and then right (R25). It then executes A$ so that it draws a 25 x 25 square starting at (128,96). After executing A$, it returns to the original (current) string and completes its execution (D25,L25).

DO-IT-YOURSELF PROGRAM 28-5

Do-It-Yourself Program 28-3 shows that you can simulate text (letters) on the graphics screen by drawing the letters. Use DRAW to create all 26 letters of the alphabet. Store the DRAW subcommands in strings. Then use the X subcommand to arrange the letters into words.

DO-IT-YOURSELF PROGRAM 28-6

Do your still have your house? If so, load the program again and use DRAW to make the front door open and close.

Learned in Chapter 28

COMMAND

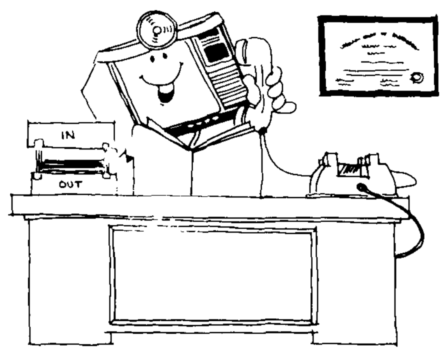
DRAW

29 / THE DISPLAY WENT THAT ARRAY

In previous chapters, you learned a few ways to move figures from one screen to another, but none is very efficient. There is a better array (groan). It has to do with GET and PUT.

How It Works (The GET and PUT Commands)

Using the GET and PUT commands, you can "get" a rectangular area from the screen, store its contents in an *array* (an area of memory), and then "put" it back anywhere you want on the screen. This is a good method for simulating motion.



The syntaxes for GET and PUT are:

GET *(x1,y1)-(x2,y2),array, G* gets a rectangle from the current graphics screen and stores it in an array

x1,y1 is the rectangle's upper-left corner.
x2,y2 is the rectangle's lower-right corner.
array is an area in memory that stores the rectangle.
G stores the array in full graphic detail. It is required when using high resolution (PMODE 4 or PMODE 3 with colors) or when using the PUT *action* parameters.

PUT *(x1,y1)-(x2,y2),array,action* puts a rectangle, stored in an array, on the current graphics screen

x1,y1 is the rectangle's upper-left corner.
x2,y2 is the rectangle's lower-right corner.
array is an area in memory where the rectangle is stored.
action (shown on Table 29-1) tells the computer what to do with the points stored in the rectangle.

Note: Be sure the computer is in the same PMODE for GET as it is for PUT. Otherwise, you might not "put" what you "got."

Type and run this program to see how GET and PUT work:

```
5   PCLEAR 4
10  PMODE 3,1
15  PCLS
20  SCREEN 1,1
25  DIM V(20,20)
30  CIRCLE (20,20),10
35  GET (10,10)-(30,30),V
40  PCLS
42  FOR DLAY = 1 TO 300: NEXT DLAY
45  PUT (110,110)-(130,130),V
50  FOR DLAY = 1 TO 300: NEXT DLAY
60  GOTO 60
```

The program draws a circle on one part of the screen and then moves it to another. To do this, the computer:

1. Creates an array named V in memory (Line 25). Array V is big enough to store a 20 x 20 rectangle.
2. Draws a circle on the screen (Line 30).
3. Gets a 20 x 20 rectangle containing the circle and stores it in the Array V (Line 35).
4. Clears the screen (Line 40).
5. Puts the 20 x 20 rectangle (stored in Array V) back on the screen.

Storing the Rectangle (The DIM Command)

Because GET and PUT use an array to store the rectangle, you need to reserve memory for this array before you use GET or PUT. The DIM command lets you do so.

DIM *array(length, width)* creates an array for storing a rectangle the size of *length* x *width* points

> **Note:** DIM needs to be one of the first lines in your program (after CLEAR and PCLEAR, if you use them).

How large does the array need to be? This depends on how large a rectangle you want to "get" or "put":

Width = *x2 - x1*
Length = *y2 - y1*

For example, this program's GET command uses (10,10) and (30,30) to specify a rectangle. So, the rectangle is 20 x 20. It has a *width* and *length* of 20. The PUT command uses the same size rectangle: 20 x 20. Some rectangles might be too large to store in an array. Each point consumes five bytes when stored in an array.

Another Kind of Action (The PSET, PRESET, AND, OR, and NOT Options)

So far, you've used only one action with PUT, the PSET *action.* When you don't specify an action, the computer uses PSET.

To see how the other *actions* work, start by running this program. It puts 15 rectangles on the screen using the PSET action.

```
5  PCLEAR 4
1Ø  DIM V (3Ø,3Ø)
15  PMODE 2,1
2Ø  PCLS
25  SCREEN 1,1
3Ø  CIRCLE (128,96),3Ø
35  PAINT (128,95),2,4
4Ø  PAINT (128,97),3,4
45  GET (98,81)-(128,111),V,G
5Ø  PCLS
55  FOR I = 15Ø TO 1 STEP -1Ø
6Ø  PUT (I,81-I/5)-(I+6Ø,111-I/5),V,PSET
65  NEXT I
7Ø  GOTO 7Ø
```

PSET sets and resets each point as it is in the array rectangle. Each rectangle it puts on the screen is the same as the one stored in the array.

Now, change Line 60 in various ways to try other actions. First, try PRESET.

```
6Ø  PUT (I/81-I/5)-(I+6Ø,111-I/5),V,PRESET
```

PRESET sets and resets the reverse of each point in the array rectangle. Each rectangle it puts on the screen is the reverse of the one stored in the array.

Try the OR action:

```
6Ø  PUT (I,81-I/5)-(I+6Ø,111-I/5),V,OR
```

OR sets each point that's either set in the array rectangle *or* already set in the position where it's putting the screen rectangle. Each rectangle it puts on the screen has all points set that are stored in the array plus what is currently on the screen.

For a strange effect, try the NOT action:

```
6Ø  PUT (I,81-I/5)-(I+6Ø,111-I/5),V,NOT
```

NOT sets and resets the reverse of what's on the screen. (NOT doesn't care what's stored in the array.) Each rectangle it puts on the screen is the reverse of the previous one.

Try the AND option with the program, and you won't see anything:

```
6Ø  PUT (I,81-I/5)-(I+6Ø,111-I/5),V,AND
```

AND sets each point that is set in the array *and* is already set on the screen in the position where it's putting the rectangle. Any points that don't meet both of those conditions are reset. In this case, each rectangle AND puts on the screen has all points reset. You see nothing.

Table 29.1 Put Actions

This is a summary of each action:

Option	Function
PSET	Sets each point that is set in the array.
PRESET	Resets each point that is set in the array; sets each point that is reset in the array.
AND	Compares each point in the array rectangle with the screen rectangle. If either or both are reset, the computer resets the screen point.
OR	Compares each point in the array rectangle to the screen rectangle. If either is set, the computer sets the screen point.
NOT	Reverses the state of each point in the screen rectangle regardless of the array rectangle's contents.

DO-IT-YOURSELF PROGRAM 29-1

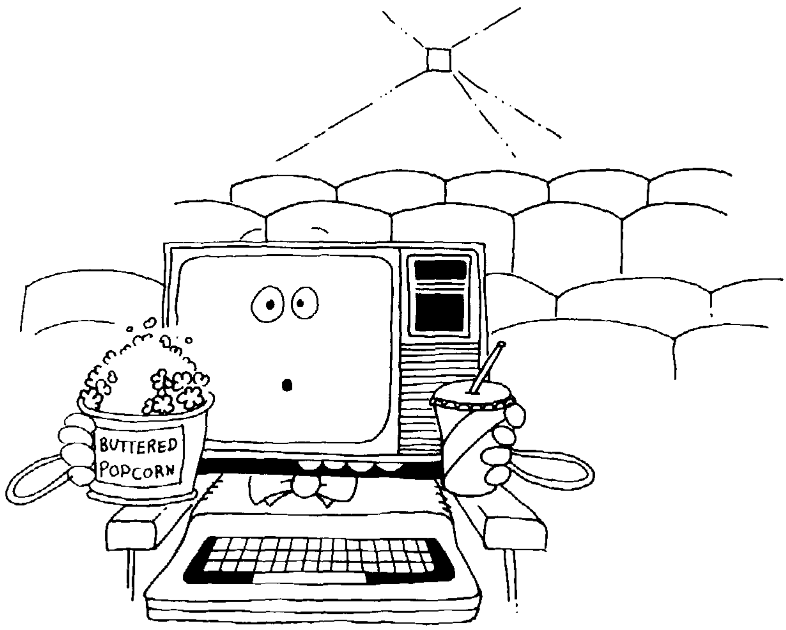
Use GET and PUT to send a spaceship up the screen. You might want to add a few asteroids and aliens to make the voyage more exciting!

Learned in Chapter 29

COMMANDS

GET
PUT
DIM

PART 4 / THE BIG PICTURE



In Chapter 17, you used the low-resolution text screen to draw simple dot-to-dot pictures. In Part 2, you used the low-resolution graphics screen to draw lines, circles, and cubes.

In this part of the book, you use the most powerful screen for graphics, the high-resolution graphics screen, to draw extremely detailed, colorful, and fast moving pictures.

Here's the Big Picture:

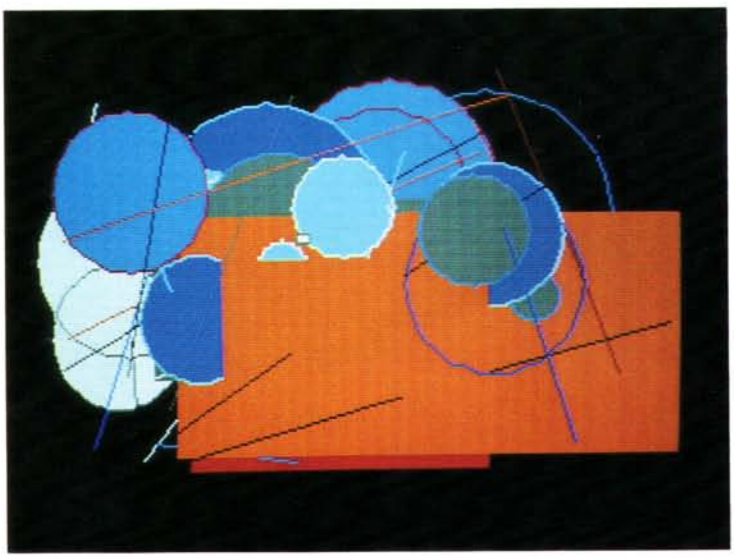
	Maximum Positions	Maximum Colors	Maximum Memory
Low-Resolution Text Screen	64x32	9	None
Low-Resolution Graphics Screen	256x192	4	Inside BASIC
High-Resolution Graphics Screen	640x192	16	Outside BASIC

THE REAL THING

Introducing . . .CoCo 3!

Sample program no. 22

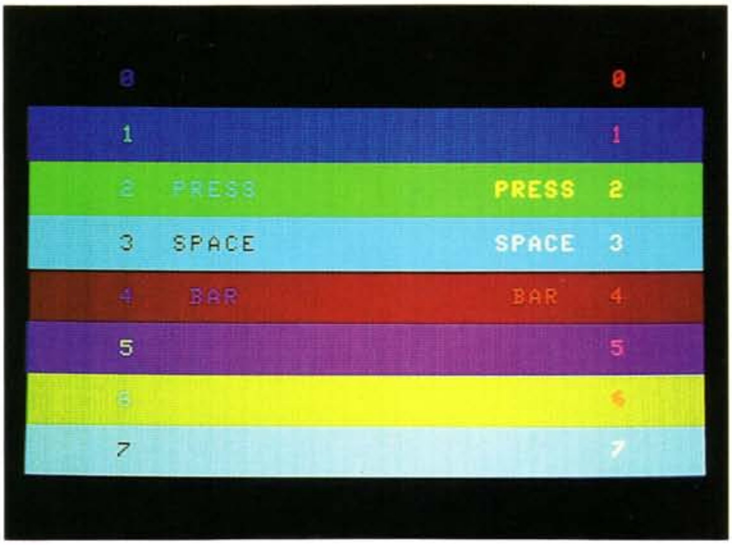
Intro highlights some of the new features of Color BASIC version 2.



Computer Art

Sample program no. 29

Random lines, circles, and boxes create colorful computer art. Try our program, then change it.



The CoCo Rainbow

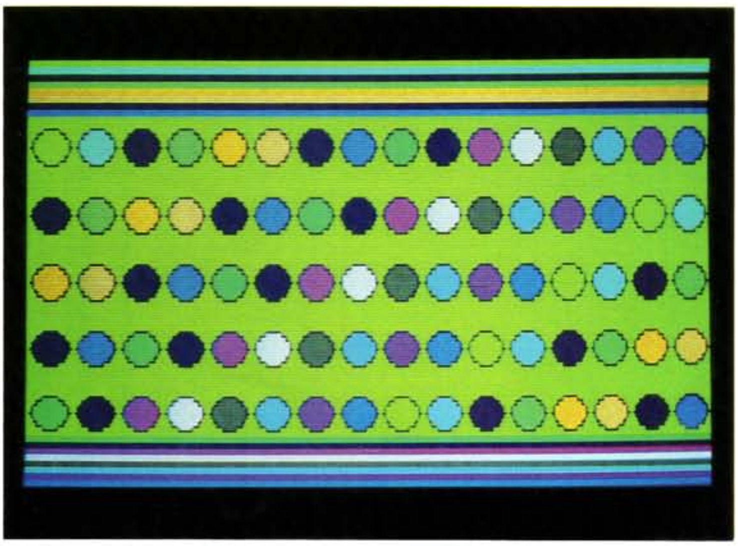
Sample program no. 23

Hcolors lets you see all sixty-four colors, eight at a time.

The Artists Palette

Sample program no. 24

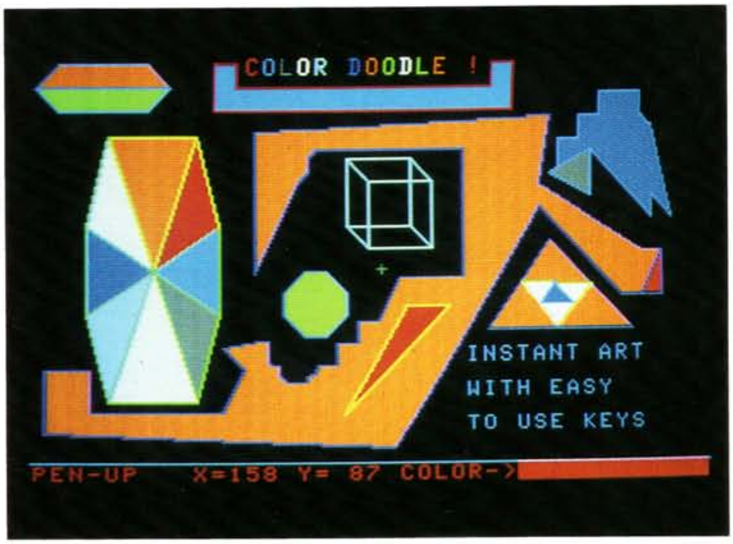
Palette shows you a mixture of sixteen colors randomly selected from the sixty-four available colors.

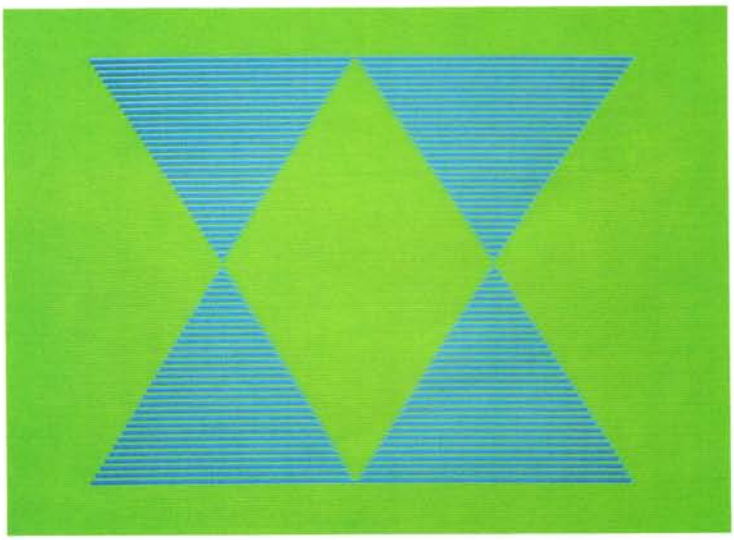


Color Doodle

Sample program no. 30

Color Doodle lets you draw your own computer art on the screen. Look at the program, and add your own special function keys.

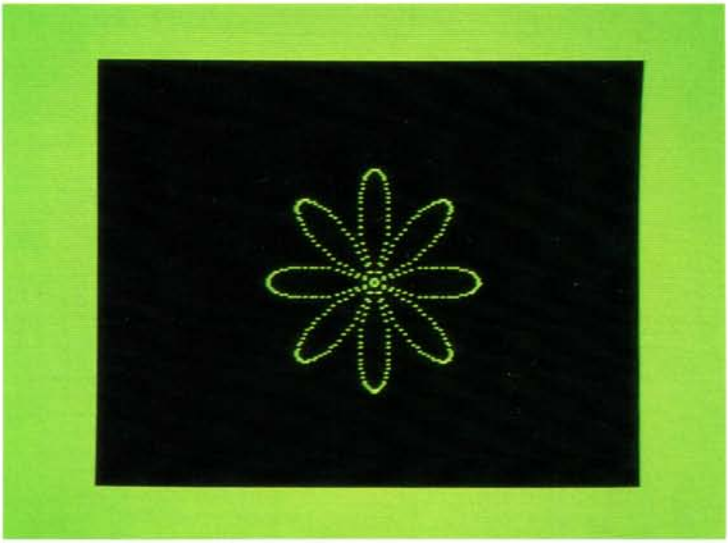




Going Everywhere at Once

Sample program no. 5

In-Out draws moving patterns of lines in exciting colors. Change the colors for more variety.



Pick a Lucky Clover

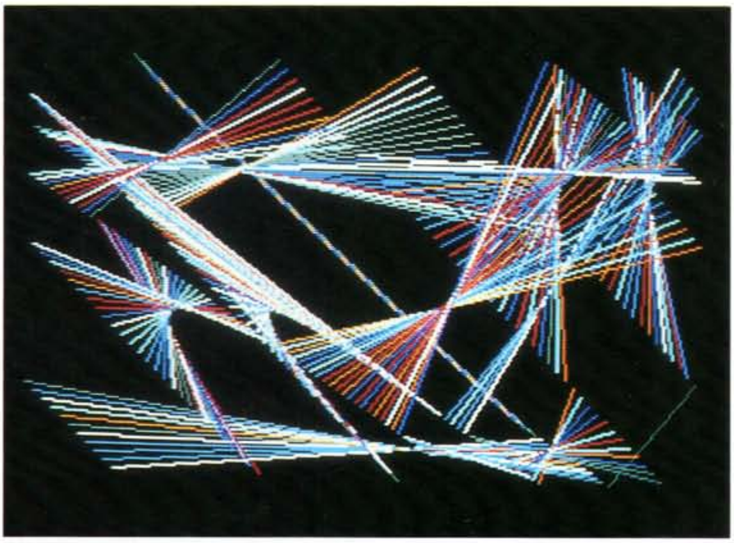
Sample program no. 17

Grow your own **8-Leaf Clover**. Transplant it to other parts of the screen.

String Art

Sample program no. 28

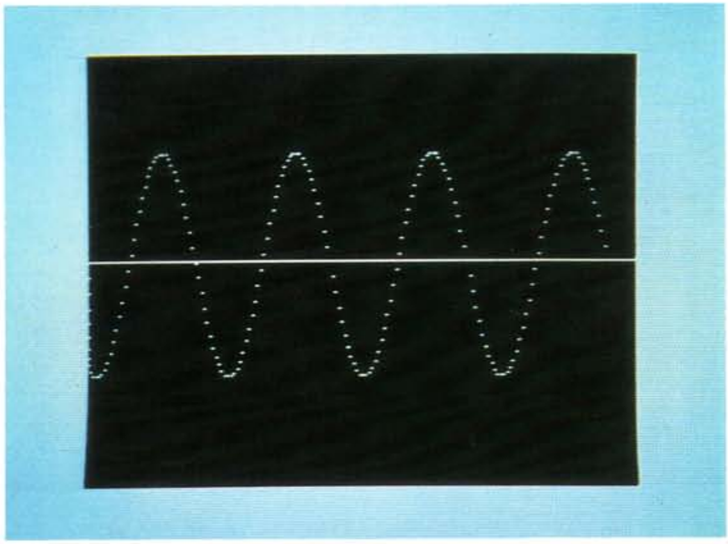
Moving lines draw colorful string art. The program uses random starting points for a line, and twists them until they bounce off the screen edges.

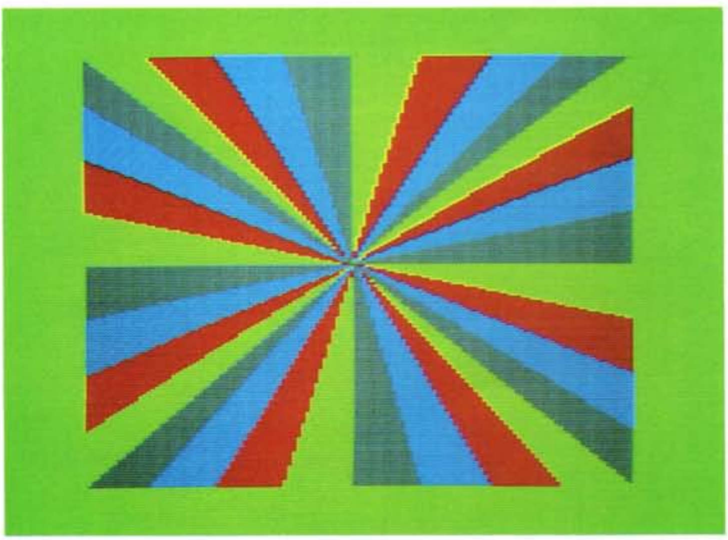


A Sine of the Times

Sample program no. 9

Weave a wave across your screen with a loop and some tricky trigonometry.

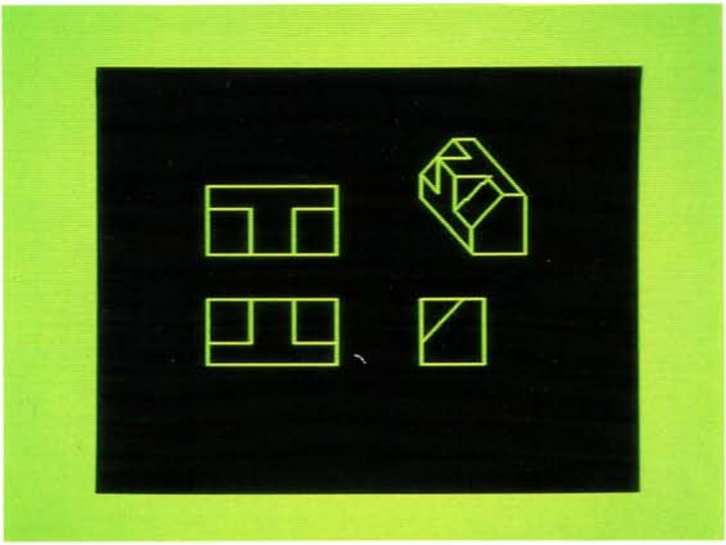




This Fan is a Breeze

Sample program no. 19

Cool off in front of your own computerized fan. Change the colors for different effects.



I CAD, Can You?

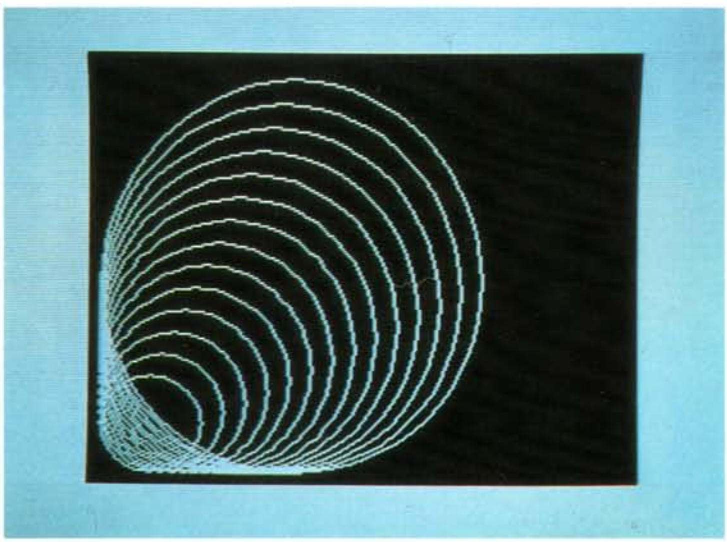
Sample program no. 7

Projection Studies gives you a first encounter with Computer Aided Design (CAD).

This is your BASIC tunnel . . .

See do it yourself program no. 24-1

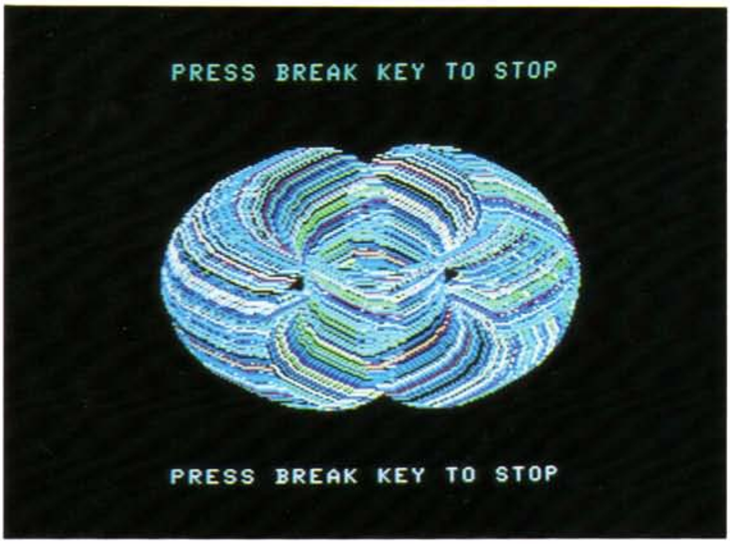
Dig an electronic tunnel. Look at how do it yourself program no. 24-1 draws chimney smoke. Delete lines 25-90 and design your own tunnel. Hint: Try a larger radius. Experiment with the program to change the depth and width of the tunnel.



Looping Loops

Sample program no. 27

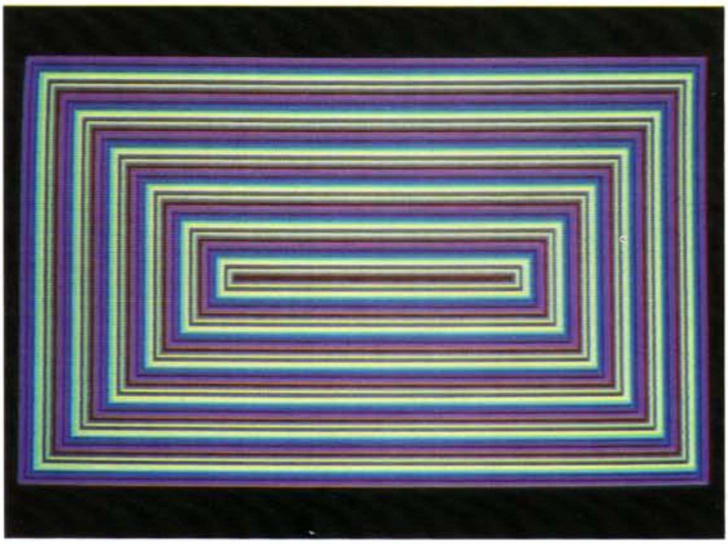
Watch circles change in color and grow. The sizes change at random.

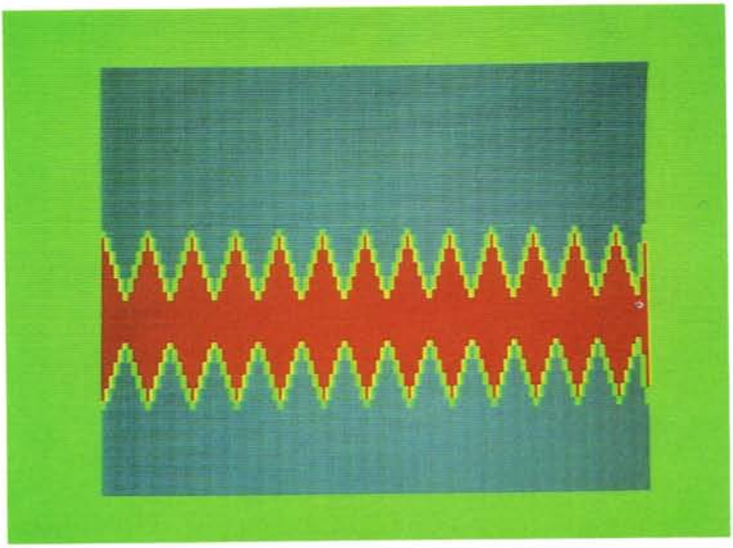


Colorful Boxes

Sample program no. 25

Colorbox draws a screen full of boxes, then changes the colors.





An Electronic Blanket

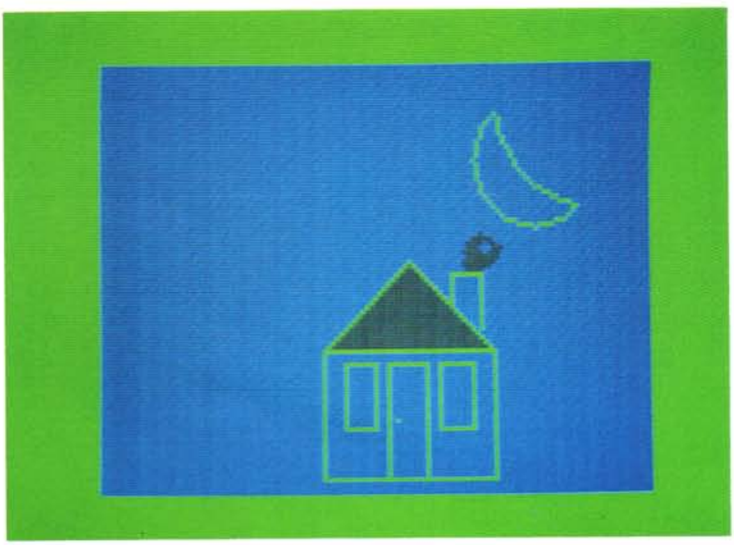
Sample program no. 13

Weave a **Navaho Blanket** to keep yourself warm. Experiment with different sizes and colors.

Build Your Own House

Do it yourself program no. 26-4

Build a house, complete with smoking chimney. Build your own additions.



30 / THOUSANDS OF DOTS

The high-resolution graphics screen works in much the same way as the low-resolution graphics screen. The only difference between the two screens is that the high-resolution screen, while offering more features, is actually easier to use.

Creating a Graphics Screen (The HSCREEN Command)

High-resolution graphics lets you create a graphics screen using only one command, the HSCREEN command. HSCREEN does the same tasks that three low-resolution commands do.

- It sets the features to be used on the graphics screen (as does the PMODE command).
- It displays the graphics screen (as does the SCREEN command).
- It clears the graphics screen (as does the PCLS command).

The syntax of HSCREEN is:

> **HSCREEN** *n* sets the features for displays, and clears the current high-resolution graphics screen.
>
> *n* specifies which features you want to use (0-4).

To see how HSCREEN works, type this program:

```
10  HSCREEN 1
20  GOTO 20
```

When you run the program, you see a blank high-resolution graphics screen. Press (BREAK) to return to the text screen.

HSCREEN, like PMODE, lets you choose from different settings that specify which features you want to use. The features you can use are:

- Grid positions—You can use a maximum of 640 x 192 positions at a time.
- Colors—You can use a maximum of 16 colors at a time.

The available settings are listed on Table 30-1.

Table 30.1 / HSCREEN Settings

	Grid Positions	Colors
HSCREEN 1	320 x 192	4
HSCREEN 2	320 x 192	16
HSCREEN 3	640 x 192	2
HSCREEN 4	640 x 192	4

The HSCREEN 1 setting that we used above specifies a 320 x 192 grid with a maximum of four colors.

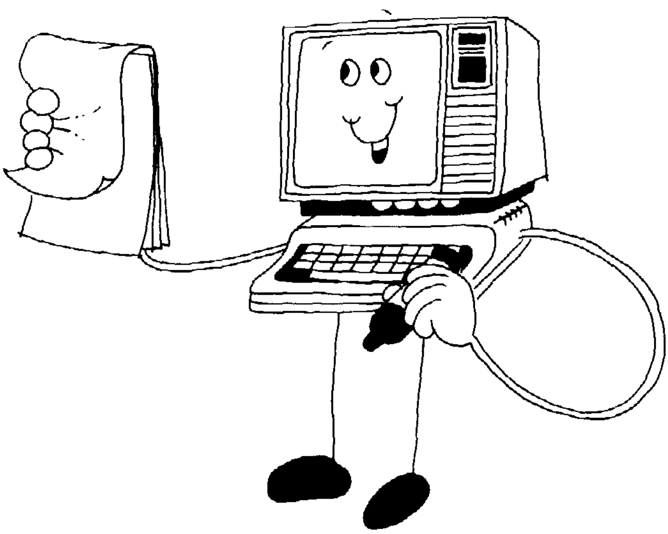
In addition to these four settings, you can use a fifth setting:

```
HSCREEN 0
```

This setting returns you to the low-resolution graphics mode. Once you have entered the high-resolution mode, by executing an HSCREEN 1 (or 2, 3, or 4) command, you cannot display a low-resolution graphics screen without first executing an HSCREEN 0 command.

Note that, unlike PMODE, HSCREEN does not use different pages of graphics memory. All high-resolution screens are stored in the same area of memory.

Producing Graphics on the Screen (The High-resolution Graphics Commands)



High-resolution graphics has a counterpart to almost all the low-resolution commands. The high-resolution commands, and their low-resolution counterparts, are listed in Table 30.2.

Table 30.2 / High- and Low-resolution Graphics Commands

High-Resolution	Low-Resolution
HCIRCLE	CIRCLE
HCLS	PCLS
HCOLOR	COLOR
HDRAW	DRAW
HLINE	LINE
HPAINT	PAINT
HRESET	PRESET
HSET	PSET
HPOINT	PPOINT

For example, the high-resolution HCIRCLE command corresponds to the low-resolution CIRCLE command. Both work in almost the same way. (The few differences are discussed below.)

Because high- and low-resolution commands are so similar, it is easy to convert a program from one mode to another.

Look at the first program in Chapter 26, which draws a circle on the low-resolution screen.

```
5    PMODE 4,1
10   PCLS
20   SCREEN 1,1
30   CIRCLE (128,96), 95
40   GOTO 40
```

To convert this program to draw a circle on the high-resolution screen, type:

```
10   HSCREEN 2
30   HCIRCLE (160,96), 95
40   GOTO 40
```

A Different Use of Grids (High- v Low-resolution Commands)

Unlike the low-resolution commands, the high-resolution commands use exact, rather than scaled, grid positions. This means that if you change to a different high-resolution grid, you need to change the dot positions in the high-resolution program.

For example, the Circle program is currently using a 640 x 192 grid.

```
10   HSCREEN 4
30   HCIRCLE (160,96), 95
40   GOTO 40
```

Change the HSCREEN setting so that it uses a 320 x 192 grid:

```
10   HSCREEN 1
```

Now, run the program. Because you changed to a 320 x 192 grid, the circle appears at a different place on the screen. To make it appear at the same place, change the dot positions in the HCIRCLE command:

```
30   HCIRCLE (80,96), 95
```

A Different Use of Color (High- v Low-resolution Commands)

High-resolution graphics commands use the palette in a different way than their low-resolution counterparts.

For example, add a line to the "Circle" program so that the computer paints the circle Color 1.

```
35   HPAINT (160,96),1,1
```

Now, run the program. In the low-resolution mode, the circle would have been painted one of several different colors, depending on which color mode (2-color or 4-color) and color set (Color Set 1 or 2) you were using.

In the high-resolution mode, Color 1 always specifies the same color: the color stored in Palette Slot 1.

To paint the circle a nonstandard color, such as color code 23, store the color code in Palette Slot 1. Type this line and run the program.

```
3  PALETTE 1, 23
```

Table 7.9 in "Odds and Ends" shows how the high-resolution graphics commands use the palette in each HSCREEN color mode.

A Better Way of Printing (The HPRINT Command)

One major drawback of using a low-resolution graphics screen is that you cannot easily print a message on it. Sure, you can use the DRAW command to make each letter, but this is tedious and requires a lot of memory.

In the high-resolution mode, the HPRINT command prints a message directly on the high-resolution graphics screen. Its syntax is:

HPRINT(*x*,*y*), *message* Prints *message* at Text Position *x*,*y* on the high-resolution graphics screen.

HPRINT requires that you give text positions, rather than graphics positions. A 320 x 192 graphics screen corresponds to a 40 x 24 text screen. A 640 x 192 graphics screen corresponds to an 80 x 24 text screen.

So, in HSCREEN 1 or HSCREEN 2, 40 columns are available. In HSCREEN 3 or HSCREEN 4, 80 columns are available. If the message is longer than the number of columns available, the last part of the message is clipped.

These are examples of HSCREEN commands:

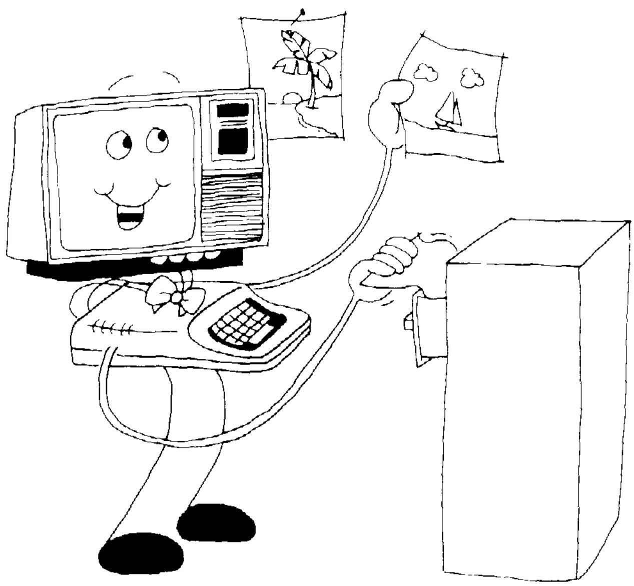
```
HPRINT(Ø,2Ø), "The Score is ", SC
HPRINT(Ø,Ø), "Your name is "; A$
HPRINT (1Ø,1Ø), A$+B$
```

Learned in Chapter 30

COMMANDS	FUNCTION
HSCREEN	HPOINT
HCIRCLE, HCLS,	
HCOLOR, HDRAW	
HLINE, HPAINT,	
HRESET, HSET,	
HPOINT	
PALETTE	
HPRINT	

31 / GRAPHICS STORAGE

GET and PUT are useful commands in the low-resolution mode. These commands let you "get" a rectangle, store it in an array, and "put" it somewhere else on the screen.



The problem is that an array consumes valuable memory space and, by doing so, takes away memory space that the BASIC program could be using.

The high-resolution counterparts of GET and PUT do not use an array to store the rectangle. In fact, they do not use any storage within the BASIC area. Instead, they use special *GET/PUT buffers*, which are areas of memory outside BASIC.

Reserving a Buffer (The HBUFF Command)

Before you can use a GET/PUT buffer, you need to tell BASIC know you plan to use it. You can do this using a special high-resolution command called HBUFF.

The syntax of HBUFF is:

HBUFF *buffer*, *size* Reserves a GET/PUT buffer.

buffer is a number that labels the GET/PUT buffer.
size is the size of the GET/PUT buffer, where size=(number of bytes-1).

To use HBUFF, BASIC needs to know how much buffer memory to set aside to store the rectangle. The unit of measure for memory is called a *byte*. Depending on the HSCREEN mode you are using, each byte can store 2, 4, or 8 dots from the screen. Table 31.1 shows how many dots BASIC stores in a byte, depending on the HSCREEN mode you are using.

Table 31.1 / Memory Required for Graphics

Screen mode	Dots per Byte
HSCREEN 1	4
HSCREEN 2	2
HSCREEN 3	8
HSCREEN 4	4

As an example, suppose you want to do an HGET of the rectangle (10,5)-(30,15) in HSCREEN 1. BASIC rounds the X coordinates down to the nearest byte, according to the graphics screen mode you are using. BASIC reads the first X value of 10, and divides it by the number of dots per byte for HSCREEN 1 (See table 31.1).

10 / 4 = 2.5

Because BASIC rounds down for HGET/HPUT X coordinates, ignore the fraction to the right of the decimal point. This gives you an answer of 2. Multiply this number by the number of dots per byte for HSCREEN 1. The result is the first X coordinate that BASIC will use for the HGET.

2 * 4 = 8

You have now solved almost half of the mystery. The other X coordinate is solved the same way.

30 / 4 = 7.5 (Ignore the fraction)

7 * 4 = 28

So, in HSCREEN 1, if you tell BASIC to HGET(10,5)-(30,15) it will actually HGET(8,5)-(28,15). Now, solve the last part of the mystery. What size does HBUFF need to be to store HGET(8,5)-(28,15) in HSCREEN 1? Here is how to figure it out.

Subtract the smaller X coordinate value (8) from the larger X coordinate value (28).

28 - 8 = 20

Add 1 to the result.

20 + 1 = 21

Divide the new result (21) by the number of dots per byte for HSCREEN 1 (4).

21 / 4 = 5.25

If there is a fraction in the answer, round the answer up to the next higher whole number. Because you got an answer of 5.25, and not 5, round the answer up to 6. This is the number of bytes required to store each row, or the width of the HGET rectangle in bytes. Now, subtract the smaller Y coordinate value (5) from the larger Y coordinate value (15).

15 - 5 = 10

Add 1 to the result.

10 + 1 = 11

This answer is the height of the HGET rectangle in dots. To find out the total number of bytes required to store the rectangle, multiply the width in bytes (6), by the height in dots (11).

6 * 11 = 66

Because *size* equals the number of bytes minus 1, subtract 1 from this value.

66 - 1 = 65

Therefore, an HGET(10,5)-(30,15) in HSCREEN 1 requires an HBUFF with a size value of 65. To see what HBUFF looks like in a program line, examine the line below. This line assigns HBUFF *buffer* number 1 a *size* of 65. Remember that *size* is 1 less than the number of bytes the buffer can store. In this example, the HBUFF can store 66 bytes.

```
10  HBUFF 1,65
```

Getting a Rectangle into the Buffer (The HGET Command)

To get a rectangle and put it into the buffer, you use the HGET command. HGET is the counterpart to GET. Its syntax is:

HGET*(x1,y1)-(x2,y2),buffer* Gets a rectangle from the high-resolution graphics screen and stores it in *buffer*.

The following program draws a small box near the top left corner of the screen and gets it into buffer 1.

```
10  HBUFF 1, 43
20  HSCREEN 4
30  HLINE(10,0)-(20,10),PSET,B
40  HGET(10,0)-(20,10),1
50  GOTO 50
```

Putting the Rectangle on the Screen (The HPUT Command)

To put the rectangle on the screen, you use the HPUT command, which corresponds to PUT. Its syntax is:

HPUT*(x1,y1)-(x2,y2),buffer,action* Puts a rectangle from *buffer* on the high-resolution graphics screen using the specified *action*. The *action* can be PSET, PRESET, AND, OR, and NOT.

This line puts the box elsewhere on the same screen:

```
45  HPUT(26,20)-(36,30),1,PSET
```

Remember that the color codes that are put on the screen refer to palette slots. Try using AND and OR. You might get some surprising and colorful results.

Learned in Chapter 31

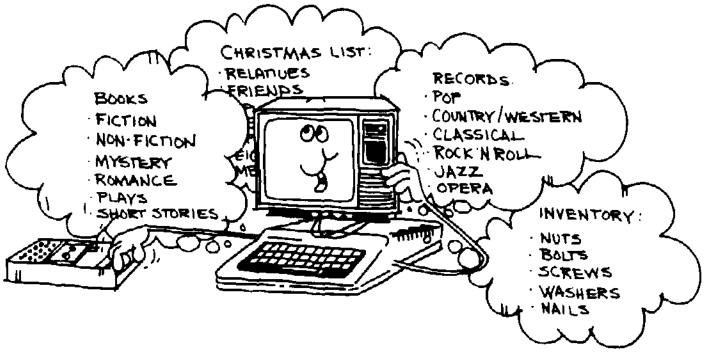
BASIC Words	Concepts
HBUFF	Reserving a buffer
HGET and HPUT	Using the buffer for moving or duplicating a rectangle

PART 5 / GETTING DOWN TO BUSINESS

In this part of the book, we get down to business and deal with data—for example, checkbook receipts, tax records, address, and books. You'll learn to organize this data by writing programs that file, update, print, and analyze.

32 / STORING DATA

Storing a BASIC program on tape is easy. You simply use the CSAVE command. Storing data on tape takes a little more effort. You need a program.



This chapter shows how to write two programs for storing data on tape. The first program stores data on tape. The second retrieves data from tape.

A Program to Output Data (The OPEN, PRINT #, and CLOSE Commands)

Assume you want to store these checks on tape:

-----------CHECKS-----------

PUBLIC HOSPITAL
PAUL'S GROCERY
CHARITY FUND
DANDY OFFICE SUPPLY

Start with a short, simple program that stores the first check, "PUBLIC HOSPITAL," to tape. Type:

```
10  OPEN "O", #-1, "CHECKS"
20  PRINT #-1, "PUBLIC HOSPITAL"
30  CLOSE #-1
```

Prepare the tape recorder for recording:

1. Connect your tape recorder.
2. Insert a tape in your tape recorder, and rewind the tape.
3. Press your recorder's PLAY and RECORD buttons until they lock.

Run the program. The tape recorder turns on while the computer does several tasks.

- It opens communication with the tape recorder.
- It prepares to store data.
- It labels the area of the tape where it will store data as a *file* named "CHECKS."

This all happens in Line 10. Note the meaning of the #-1, "O", and "CHECKS":

- #-1 specifies the tape recorder.
- "O" stands for output.
- "CHECKS" specifies a filename.

The next line, Line 20, sends "PUBLIC HOSPITAL" to the tape recorder.

The last line, Line 30, closes communication with the tape recorder.

The program we wrote, which we call an output program, uses three new commands. Their syntaxes are:

`OPEN` *mode, device, file* Opens communication with *device* so you can transmit information to *file* using the specified *mode* of data transmission.

mode can be "I" (Input) or "O" (Output) *device* can be #0 (screen or keyboard), #-1 (cassette), or #-2 (printer).

`PRINT` *#device, message* Prints a *message* to *device*.

`CLOSE` *#device* Closes communication with *device*.

A Program to Retrieve Data (The INPUT # Command)

To load data back into memory, you need an input program. Erase the output program you now have in memory, and type:

```
100  OPEN "I", #-1, "CHECKS"
110  INPUT #-1, A$
120  PRINT A$
130  CLOSE #-1
```

Prepare the recorder for loading data:

1. Rewind the tape.
2. Press the PLAY button.
3. Then, run the program.

Line 100 opens communication with the tape recorder, this time to retrieve ("I") data from a file named "CHECKS".

Line 110 inputs a data item from "CHECKS" and labels this item as A$. Line 120 displays A$.

Line 120 closes communication with the tape recorder.

The input program uses another new command. Its syntax is:

INPUT *#device* Retrieves information from device until one of the following characters is encountered: a comma (,), a semicolon (;), or a carriage return (ENTER).

Finding the End of the File (The EOF Function)

Suppose you don't know how many data items are stored in "CHECKS". You want to retrieve all the data items until you reach the end of the file.

You can do this by adding these lines to the input program you now have in memory:

```
105  IF EOF(-1) = -1 THEN 130
125  GOTO 105
```

Line 105 checks to see if you reached the end of the file.

- If you have, EOF(-1) equals -1. The computer goes to Line 30 and closes communication with the file.
- If you have not, EOF(-1) does not equal -1. The computer retrieves the next data item in the file.

To see if you have reached the end of the file, you used a new function. Its syntax is:

EOF(*device*) Returns a number indicating whether you've reached the end of the file on *device*. The number returned is -1 (end of file) or 0 (not the end of file).

Storing More Data

So far, "CHECKS" has been easy to handle, but not very useful. Suppose you want to store all this information in "CHECKS":

CHECKS

PAYABLE TO	AMOUNT	EXPENSE
PUBLIC HOSPITAL	45.78	MEDICAL
PAUL'S GROCERY	22.50	FOOD
CHARITY FUND	20.00	CONTRIBUTION
SALES OFFICE	13.67	BUSINESS

Here is an output program that lets you store all the above information, not only for the four checks listed above, but for as many checks as you want:

```
5   CLS
10  OPEN "O", #-1, "CHECKS"
20  INPUT "CHECK PAYABLE TO :"; A$
30  IF A$ = "" THEN 80
40  INPUT "AMOUNT : $"; B
50  INPUT "EXPENSE :"; C$
60  PRINT #-1, A$, B, C$
70  GOTO 20
80  CLOSE #-1
```

Here is a complementary input program that retrieves all the checks you stored until it reaches the end of the file:

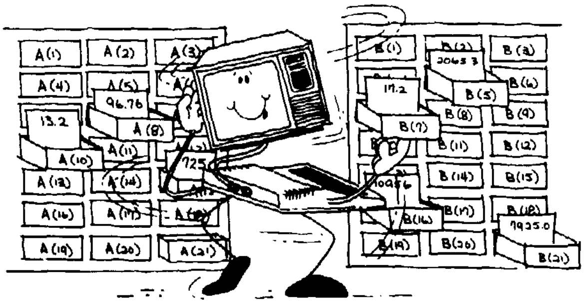
```
100   OPEN "I", #-1, "CHECKS"
110   IF EOF(-1) = -1 THEN 150
120   INPUT #-1, A$, B, C$
150   PRINT A$; B; C$
160   GOTO 110
170   CLOSE #-1
```

Learned in Chapter 32

COMMANDS	FUNCTION
OPEN	EOF
CLOSE	
PRINT #	
INPUT #	

33 / NUMERIC ARRAYS

In this chapter, we show how to organize large groups of numbers using a new kind of variable and a new way of organizing variables.



A New Kind of Variables (Subscripted Variables)

Assume you want to store all the votes for Districts 1-14 into variables:

ELECTION RETURNS

District	Votes for Candidate
1	143
2	215
3	125
4	331
5	442
6	324
7	213
8	115
9	318
10	314
11	223
12	152
13	314
14	92

One way is to store them in the same kind of variables you've been using all along, *simple variables*. For example, store the votes for the first three districts into simple variables by typing:

```
A = 143 (ENTER)
B = 215 (ENTER)
C = 125 (ENTER)
```

A better way is to use *subscripted variables*. Type:

```
A(1) = 143 (ENTER)
A(2) = 215 (ENTER)
A(3) = 125 (ENTER)
```

Subscripted variables have subscripts such as (1), (2), and (3). Other than the subscripts, they work the same as simple variables. To see for yourself, type both of these lines:

```
PRINT A; B; C (ENTER)
PRINT A(1);A(2);A(3) (ENTER)
```

A Way of Organizing Subscripted Variables (Arrays and the DIM Command)

Take a quick look, and compare the two programs below. Both work the same. Program 1 uses simple variables; Program 2 uses subscripted variables.

PROGRAM 1

```
10  DATA 143,215,125,331,442
20  DATA 324,213,115,318,314
30  DATA 223,152,314,92
40  READ A,B,C,D,E
50  READ F,G,H,I,J
60  READ K,L,M,N
70  INPUT "DISTRICT NO. (1-14)";Z
75  IF Z>14 THEN 70
80  IF Z=1 THEN PRINT A "VOTES"
90  IF Z=2 THEN PRINT B "VOTES"
100  IF Z=3 THEN PRINT C "VOTES"
110  IF Z=4 THEN PRINT D "VOTES"
120  IF Z=5 THEN PRINT E "VOTES"
130  IF Z=6 THEN PRINT F "VOTES"
140  IF Z=7 THEN PRINT G "VOTES"
150  IF Z=8 THEN PRINT H "VOTES"
160  IF Z=9 THEN PRINT I "VOTES"
170  IF Z=10 THEN PRINT J "VOTES"
180  IF Z=11 THEN PRINT K "VOTES"
190  IF Z=12 THEN PRINT L "VOTES"
200  IF Z=13 THEN PRINT M "VOTES"
210  IF Z=14 THEN PRINT N "VOTES"
220  GOTO 70
```

PROGRAM 2

```
10  DATA 143,215,125,331,442
20  DATA 324,213,115,318,314
30  DATA 223,152,314,92
40  DIM A(14)
50  FOR X=1 TO 14
60  READ A(X)
70  NEXT X
80  INPUT "DISTRICT NO (1-14)";Z
85  IF Z>14 THEN 80
90  PRINT A(Z) "VOTES"
100  GOTO 80
```

Program 1 is cumbersome to write. Program 2 is short and simple to write.

Enter and run Program 2. Here's how it works:

- Line 40 reserves, or *dimensions*, space for an *array* named A with subscripted variables ranging from A(0) to A(14).
- Lines 50 and 70 set up a loop to count from 1 to 14.

  Line 60 reads all 14 votes into Array A:

```
              YOUR COMPUTER'S MEMORY
A(1) = 143                         A(8)  = 115
A(2) = 215                         A(9)  = 318
A(3) = 125                         A(1Ø) = 314
A(4) = 331                         A(11) = 223
A(5) = 442                         A(12) = 152
A(6) = 324                         A(13) = 314
A(7) = 213                         A(14) = 92
```

- Line 80 asks you to enter a subscript, and Line 90 prints the item you requested.

Line 40 uses a new BASIC command, DIM. DIM's syntax is:

`DIM`*variable*(*n*) Dimensions *variable* as an array with *n* subscripts.

> **Note:** Actually, you are only required to use DIM when you plan to use subscripts higher than 10. However, even if you're not using subscripts higher than 10, it's a good idea to use DIM anyway, for reserving exactly the right amount of memory.

Now that you stored information in an array, it's easy to manage the information. For instance, add these lines, to change the information:

```
92  INPUT "DO YOU WANT TO ADD TO THIS";R$
94  IF R$="NO" THEN 8Ø
96  INPUT "HOW MANY MORE VOTES";X
97  A(Z)=A(Z) + X
98  PRINT "TOTAL VOTES FOR DISTRICT "Z"IS NOW "A(Z)
```

Or add these lines to display the information:

```
72  INPUT "DO YOU WANT TO SEE ALL THE TOTALS";S$
74  IF S$="YES" THEN GOSUB 11Ø
1ØØ  GOTO 72
11Ø  PRINT "DISTRICT", "VOTES"
12Ø  FOR X=1 TO 14
13Ø  PRINT X,A(X)
14Ø  NEXT X
15Ø  RETURN
```

Adding a Second Array (Using 2 Arrays)

Assume you also want to keep track of a second candidate's votes, Candidate B:

ELECTION RETURNS

District	Votes for Candidate A	Votes for Candidate B
1	143	678
2	215	514
3	125	430
4	331	475
5	442	302
6	324	520
7	213	613
8	115	694
9	318	420
10	314	518
11	223	370
12	152	412
13	314	460
14	92	502

Add another array to the program. Call it Array B. The following program records the votes for Candidate A (Array A) and Candidate B (Array B):

```
10  DATA 143,215,125,331,442
20  DATA 324,213,115,318,314
30  DATA 223,152,314,92
40  DATA 678,514,430,475,302
50  DATA 520,613,694,420,518
60  DATA 370,412,460,502
70  DIM A(14), B(14)
80  FOR X = 1 TO 14
90  READ A(X)
100  NEXT X
110  FOR X = 1 TO 14
120  READ B(X)
130  NEXT X
140  INPUT "DISTRICT NO.";Z
145  IF Z>14 THEN 140
150  INPUT "CANDIDATE A OR B";R$
160  IF R$="A" THEN PRINT A(Z)
170  IF R$="B" THEN PRINT B(Z)
180  GOTO 140
```

DO-IT-YOURSELF PROGRAM 33-1

Write an inventory program that keeps track of 12 items (numbered 1-12) and the quantity you have of each item.

Dealing The Cards (An Example of Arrays)

To keep track of 52 cards, you need to use an array. Erase your program. Type and run this one:

```
40   FOR X=1 TO 52
50   C=RND(52)
90   PRINT C;
100   NEXT X
```

The computer deals 52 random cards, but if you look closely, you see that some cards are the same.

To be sure the computer deals each card only once, you can build another array, Array T, that keeps track of each card dealt. Add these lines:

```
5   DIM T(52)
10   FOR X=1 TO 52
20   T(X)=X
30   NEXT X
```

The previous lines build Array T and put all 52 cards in it: T(1) = 1, T(2) = 2, T(3) = 3 . . . T(52) = 52.

Now, add some lines that "erase" each card in Array T after it's dealt. Type:

```
60   IF T(C)=0 THEN 50
80   T(C)=0
```

Now the computer can't deal the same random card twice. For example, assume the computer first deals a two. Line 80 changes T(2)'s value from 2 to 0.

Then, assume the computer deals another two. Since T(2) now equals 0, Line 60 goes back to Line 50 to deal another card.

Run the program. Note how the computer slows down at the end of the deck. It must try many different cards before it finds one that it hasn't dealt yet.

To play a card game, you need to keep track of which cards were dealt. You can do this by building another array, Array D. Add these lines that store all the cards in the order they are dealt in Array D:

```
7   DIM D(52)
70   D(X) = T(C)
90   PRINT D(X);
```

DO-IT-YOURSELF PROGRAM 33-2

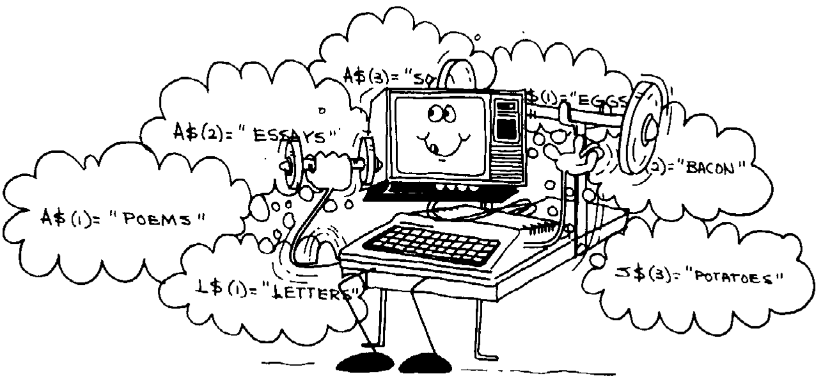
Add lines to the program so it displays only your "hand," the first 5 cards dealt.

LEARNED IN CHAPTER 33

COMMAND	CONCEPT
DIM	arrays

34 / STRING ARRAYS

In the last chapter, you used arrays to manage numbers. Here, you use arrays to manage words by editing, updating, and printing an entire essay.



Storing Words Into Variables (String Arrays)

Start with a simple list of words, a shopping list.

1. EGGS
2. BACON
3. POTATOES
4. SALT
5. SUGAR
6. LETTUCE
7. TOMATOES
8. BREAD
9. MILK
10. CHEESE
11. FISH
12. JUICE

Assign each word to a subscripted variable. This time use a subscripted string variable. For example, for the first three items, type:

```
S$(1) = "EGGS" (ENTER)
S$(2) = "BACON" (ENTER)
S$(3) = "POTATOES" (ENTER)
```

To see how the items are stored, type:

```
PRINT S$(1), S$(2), S$(3) (ENTER)
```

Now build a program that reads these words into an array named S$ and then displays them.

```
5   DIM S$(12)
10  DATA EGGS, BACON, POTATOES, SALT
20  DATA SUGAR, LETTUCE, TOMATOES, BREAD
30  DATA MILK, CHEESE, FISH, JUICE
40  FOR X=1 TO 12
50  READ S$(X)
60  NEXT X
70  PRINT "SHOPPING LIST:"
80  FOR X=1 TO 12
90  PRINT X; S$(X)
100  NEXT X
```

DO-IT-YOURSELF PROGRAM 34-1

Add some lines to the above program so you can change any item on this list.

DO-IT-YOURSELF PROGRAM 34-2

Here is a program that uses an array to write song lyrics.

```
5   DIM A$(4)
10  PRINT "TYPE 4 LINES"
20  FOR X=1 TO 4
30  INPUT A$(X)
40  NEXT X
50  CLS
60  PRINT "THIS IS YOUR SONG"
70  PRINT
80  FOR X=1 TO 4
90  PRINT X; " "; A$(X)
100  NEXT X
```

Add some lines so you can revise any line.

Writing an Essay (An Example of String Arrays)

Now that you learned how to use string arrays, it is easy to write a program that stores and edits what you type. Type this program:

```
1   CLEAR 500
5   DIM A$(50)
10  PRINT "TYPE A PARAGRAPH"
20  PRINT "PRESS </> WHEN FINISHED"
30  X = 1
40  A$ = INKEY$
50  IF A$ = " " THEN 40
60  PRINT A$;
70  IF A$ = "/" THEN 110
80  A$(X) = A$(X) + A$
90  IF A$ = "." THEN X = X + 1
100  GOTO 40
110  CLS
120  PRINT "YOUR PARAGRAPH:"
130  PRINT
140  FOR Y = 1 TO X - 1
150  PRINT A$(Y);
160  NEXT Y
170  PRINT
```

Run the program. To see how each sentence is stored, type these lines:

```
PRINT A$(1) ENTER
PRINT A$(2) ENTER
PRINT A$(3) ENTER
```

Here's how the program works:

1. Line 1 clears plenty of string space.
2. Line 5 saves room for an array named A$ that can have up to 50 sentences.
3. Line 30 makes X equal to 1. X will be used to label all the sentences.
4. Line 40 checks to see which key you are pressing. If it is nothing (" "), Line 50 sends the computer back to Line 40.
5. Line 60 prints the key you pressed.
6. Line 70 sends the computer to the lines that print your paragraph when you press the "/" key.
7. Line 80 builds a string and labels it with number X. S is equal to 1 until you press a period (.). Then Line 90 makes X equal to X + 1.
8. Lines 140—160 print your paragraph.

For example, if the first letter you press is "R,"

A$(1) EQUALS "R".

If the second letter you press is "O",

A$(1) EQUALS A$(1) - which is "R" + "O"
or
"RO".

Assume when A$(1) equals ROSES ARE RED, you press a period. A$(1) then equals the entire sentence, ROSES ARE RED. The next letter you press is in A$(2).

DO-IT-YOURSELF CHALLENGER PROGRAM 34-3

Here's a tough one (but not impossible) for those intrigued with word processing. Change the previous program so you can:

- Print any sentence
- Revise any sentence

You might need to review the challenger program in Chapter 12. Our answer's in the back.

Using the Printer (The PRINT # and LLIST Commands)

If you have a printer, connect it now by plugging it into the jack marked SERIAL I/O. Turn on the printer and insert paper. The manual that comes with the printer shows how. Ready? Type this short program:

```
10  INPUT A$
20  PRINT #-2,A$
```

Now type:

```
LLIST (ENTER)
```

If your program doesn't list on the printer, be sure the printer is *on-line* (on) and connected to your keyboard. Then, type `LLIST` (ENTER) again.

Run the program and watch the printer work. PRINT #-2, tells the computer to print, not on the screen, but on device #-2, which is the printer. Be sure to type a comma after the -2, or you get a syntax error.

DO-IT-YOURSELF PROGRAM 34-4

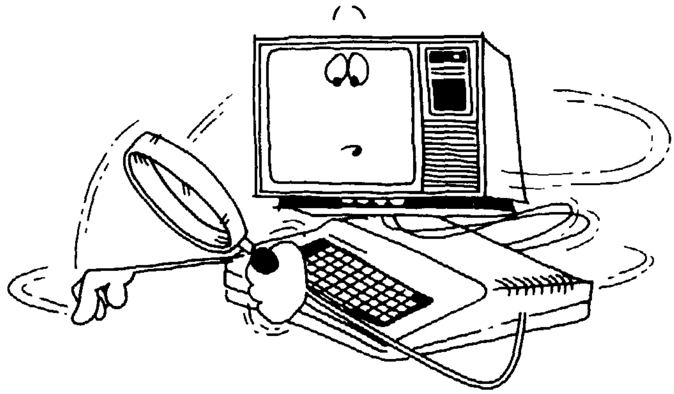
Look at the "Writing an Essay" program earlier in this chapter. Change Lines 140-160 so that the paragraph prints on the printer rather than on the screen.

Learned in Chapter 34

COMMANDS	**CONCEPT**
LLIST PRINT #-2	string arrays

35 / MULTIDIMENSIONAL ARRAYS

Arrays provide an easy way to analyze information. By giving each item more than one subscript, you can see it through different dimensions.



Storing Tables of Numbers (2-Dimensional Arrays)

District	Votes for Candidate 1	Votes for Candidate 2
1	143	678
2	215	514
3	125	430

In this chapter, you'll store information in one easy-to-manage *2-dimensional array*, Array V.

The following program puts the items in Array V.

```
5  DIM V(3,2)
10  DATA 143,678,215,514,125,430
20  FOR D=1 TO 3
30  FOR C=1 TO 2
40  READ V(D,C)
50  NEXT C
60  NEXT D

70  INPUT "DISTRICT NO. (1-3)";D
80  IF D<1 OR D>3 THEN 70
90  INPUT "CANDIDATE NO. (1-2)";C
100  IF C<0 OR C>2 THEN 90
110  PRINT V(D,C)
120  GOTO 70
```

Type and run the program. Notice that each item is labeled by two subscripts.

Here's how the program works:

Line 5 reserves space in memory for Array V. Each item in Array V can have two subscripts. The first can be no higher than 3. The second, no higher than 2.

Lines 20-60 read all the votes into Array V, giving them each two subscripts:

- The first subscript is the district (Districts 1-3).
- The second subscript is the candidate (Candidates 1-2).

```
              YOUR COMPUTER'S MEMORY
V(1,1) = 143                      V(1,2) = 678
V(2,1) = 215                      V(2,2) = 514
V(3,1) = 125                      V(3,2) = 430
```

For example, 678 is labeled V(1,2). This means 678 is from District 1 and is for Candidate 2.

With all the votes in a two-dimensional array, it's simple to analyze them—in two dimensions. By adding these lines, for example, you can print all the votes in two ways: by district and by candidate.

(First delete lines 70-120)

```
70  INPUT "TYPE <1> FOR DISTRICT OR <2> FOR
    CANDIDATE";R
80  IF R<1 OR R>2 THEN 70
100  ON R GOSUB 1000, 2000
110  GOTO 70

1000  INPUT "DISTRICT NO(1-3)";D
1010  IF D<1 OR D>3 THEN 1000
1015  CLS
1020  PRINT @132, "VOTES FROM DISTRICT" D
1030  PRINT
1040  FOR C=1 TO 2
1050  PRINT "CANDIDATE:" C,
1060  PRINT V(D,C)
1070  NEXT C
1080  RETURN

2000  INPUT "CANDIDATE NO(1-2)";C
2010  IF C<1 OR C>2 THEN 2000
2015  CLS
2020  PRINT @132, "VOTES FOR CANDIDATE"C
2030  PRINT
2040  FOR D=1 TO 3
2050  PRINT "DISTRICT"D,
2060  PRINT V(D,C)
2070  NEXT D
2080  RETURN
```

The Third Dimension (3-Dimensional Arrays)

You can continue with as many dimensions as you want. You're limited only by how much information you can fit into the computer's memory.

Add a third dimension to Array V: interest groups. Here's the information:

VOTES FROM INTEREST GROUP 1

	Candidate 1	Candidate 2
District 1	143	678
District 2	215	514
District 3	125	430

VOTES FROM INTEREST GROUP 2

	Candidate 1	Candidate 2
District 1	525	54
District 2	318	158
District 3	254	200

VOTES FROM INTEREST GROUP 3

	Candidate 1	Candidate 2
District 1	400	119
District 2	124	300
District 3	75	419

To get all this into your computer's memory, erase your program and type:

```
5  DIM V(3,3,2)
10  DATA 143,678,215,514,125,430
20  DATA 525,54,318,158,254,200
30  DATA 400,119,124,300,75,419
40  FOR G=1 TO 3
50  FOR D=1 TO 3
60  FOR C=1 TO 2
70  READ V(G,D,C)
80  NEXT C
90  NEXT D
100  NEXT G
110  INPUT "INTEREST GROUP NO (1-3)";G
120  IF G<1 OR G>3 THEN 110
130  INPUT "DISTRICT NO. (1-3)"; D
140  IF D < 1 OR D > 3 THEN 130
150  INPUT "CANDIDATE NO. (1-2)";C
160  IF C<1 OR C>2 THEN 150
170  PRINT V(G,D,C)
180  GOTO 110
```

Run the program, and test the subscripts. Lines 40-100 read all the votes into Array V, giving them each three subscripts:

- The first subscript is the interest group (Interest Groups 1-3).
- The second subscript is the district (Districts 1-3).
- The third subscript is the candidate (Candidates 1-2).

```
                YOUR COMPUTER'S MEMORY
V(1,1,1) = 143                V(1,1,2) = 678
V(1,2,1) = 215                V(1,2,2) = 514
V(1,3,1) = 125                V(1,3,2) = 430
V(2,1,1) = 525                V(2,1,2) = 54
V(2,2,1) = 318                V(2,2,2) = 158
V(2,3,1) = 254                V(2,3,2) = 200
V(3,1,1) = 400                V(3,1,2) = 119
V(3,2,1) = 124                V(3,2,2) = 300
V(3,3,1) = 75                 V(3,3,2) = 419
```

For example, 678 is now labeled V(1,1,2). This means 678 is from Interest Group 1, is from District 1, and is for Candidate 2.

To take advantage of all three dimensions, delete Lines 110-180 and type:

```
110  PRINT: PRINT "TYPE <1> FOR GROUP"
120  PRINT "<2> FOR DISTRICT OR <3> FOR CANDIDATE"
130  P=224 : INPUT R
140  ON R GOSUB 1000,2000,3000
150  GOTO 110

1000  INPUT "GROUP (1-3)";G
1010  IF G<1 OR G>3 THEN 1000
1020  CLS
1030  PRINT @102, "VOTES FROM GROUP"G
1040  PRINT @168, "CAND. 1"
1050  PRINT @176, "CAND. 2"
1060  FOR D=1 TO 3
1070  PRINT @P, "DIST."D
1080  FOR C=1 TO 2
1100  PRINT @P + 8*C, V(G,D,C);
1110  NEXT C
1120  P=P+32
1130  NEXT D
1140  RETURN
```

```
2000  INPUT "DISTRICT (1-3)";D
2010  IF D<1 OR D>3 THEN 2000
2020  CLS
2030  PRINT @102, "VOTES FROM DIST."D
2040  PRINT @168, "CAND. 1"
2050  PRINT @176, "CAND. 3"
2060  FOR G=1 TO 3
2070  PRINT @P, "GROUP"G
2080  FOR C=1 TO 2
2100  PRINT @P + 8*C,V(G,D,C);
2110  NEXT C
2120  P=P+32
2130  NEXT G
2140  RETURN

3000  INPUT "CANDIDATE (1-2)";C
3010  IF C<1 OR C>2 THEN 3000
3020  CLS
3030  PRINT @102, "VOTES FOR CAND." C
3040  PRINT @168, "DIST. 1"
3050  PRINT @176, "DIST. 2"
3060  PRINT @184, "DIST. 3"
3070  FOR G=1 TO 3
3075  FOR D=1 TO 3
3080  PRINT @P + 8*D, V(G,D,C);
3110  NEXT D
3120  P=P+32
3130  NEXT G
3140  RETURN
```

Run the program. You can now get three perspectives on the information.

DO-IT-YOURSELF PROGRAM 35-1

Write a program to deal the cards using a two-dimensional array. Make the first dimension the card's suit (1-4) and the second dimension the card's value (1-13).

Learned in Chapter 35

CONCEPT

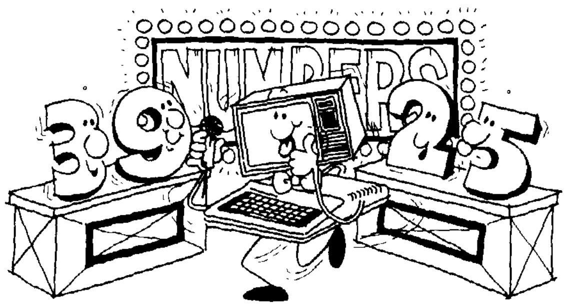
Multidimensional arrays

PART 6 / BACK TO BASICS

Ready for more basics? In this part of the book, you learn some new BASIC words that will help you refine and polish your programs.

36 / NUMBERS

You can use many numeric functions to help with mathematical calculations. This chapter lists these functions and also shows how you can create some functions of your own.



Arithmetic Functions (The SQR, FIX, ABS, and SGN Functions)

The first group of functions help with arithmetic problems. They are the SQR, FIX, ABS, and SGN functions.

SQR lets you find the square root of a number. Its syntax is:

SQR *(number)*

number is zero or any positive number.

As an example SQR, type:

```
PRINT SQR (100) (ENTER)
```

The computer displays 10, the square root of 100.

FIX converts a number to a whole-number by chopping off all the digits to the right of the decimal point. Its syntax is:

FIX *(number)*

As an example of FIX, type:

```
PRINT FIX (2.7643951) (ENTER)
```

The computer displays 2, the whole portion of 2.7643951.

As another example FIX, this program breaks a number into its whole and fractional portions.

```
10  CLS
20  INPUT "A NUMBER LIKE X.YZ"; X
30  W=FIX (X)
40  F=ABS(X)-ABS(W)
50  PRINT "WHOLE PART="; W
60  PRINT "FRACTINAL PART="; F
70  GOTO 20
```

SGN tells you whether a number is positive, negative, or zero. Its syntax is:

SGN(*number*)

As an example of SGN, run this program:

```
10   INPUT "TYPE A NUMBER"; X
20   IF SGN(X) = 1 THEN PRINT "POSITIVE"
30   IF SGN(X) = 0  THEN PRINT "ZERO"
40   IF SGN(X) = -1 THEN PRINT "NEGATIVE"
50   GOTO 10
```

ABS tells you the absolute value of a number (the magnitude of a number without respect to its sign). Its syntax is:

ABS(*number*)

As an example of ABS, run this program:

```
10   INPUT "TYPE A NUMBER"; N
20   PRINT "ABSOLUTE VALUE IS" ABS(N)
30   GOTO 10
```

Trigonometry Functions (The SIN, COS, TAN, and ATN Functions)

The next group of functions calculate *trigonometry* operations (calculating unknown sides and angles of a triangle). They are the SIN, COS, LOG, and EXP functions.

SIN calculates the *sine* of an angle. Its syntax is:

SIN *(angle)*
angle is the angle's size in radians.

COS calculates the *cosine* of a triangle. Its syntax is:

COS *(angle)*
angle is angle's size in radians.

TAN calculates the *tangent* of an angle. Its syntax is:

TAN *(angle)*

angle is angle's size in radians.

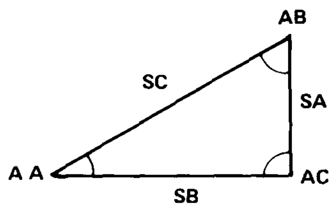
ATN calculates the *arctangent* of a triangle. Its syntax is:

ATN *(angle)*
angle is angle's size in radians.

The following trignometry programs use SIN, COS, TAN, and ATN to calculate unknown sides and angles of a triangle. You can use these programs for many practical applications, and you do not have to understand trigonometry to use them.

One practical application of these programs is building. For example, if you are building a stairway, you can use these programs to calculate the slope and height of the stairs.

Each program uses the labels SA, SB, SC, AA, AB, and AC to label the sides and angles of a triangle, as shown in this illustration.



The first program has you enter Side AC and Angles AA and AB. It then uses SIN to calculate Sides SA and SB.

```
5   CLS
10  INPUT "WHAT IS ANGLE AA (IN DEGREES)"; AA:
    IF AA<=0  OR AA>=180 THEN 100
20  INPUT "WHAT IS ANGLE AB (IN DEGREES)"; AB:
    IF AA <=0  OR AB >=180 THEN 100
30  INPUT "WHAT IS SIDE SC (SC)"; SC:
    IF SC<=0 THEN 100
40  AC=180-(AA+AB) 'VALUE OF ANGLE AC
50  IF (AA+AB+AC) <> 180 THEN 100
    'TRIANGLE=180 DEGREES
60  AA=AA/57.29577951: AB=AB/57.29577951:
    AC=AC/57.29577951
    'CONVERT DEGREES TO RADIANS
70  SA=(SIN(AA))/(SIN(AC))*SC: IF SA<0 THEN 100
80  SB=(SIN(AB))/(SIN(AC))*SC: IF SB<0 THEN 100
90  PRINT "SIDE SA (SA) IS" SA "LONG": PRINT "SIDE SB
    (SB) IS" SB "LONG": GOTO 10
100  PRINT "SORRY, NOT A TRIANGLE, TRY AGAIN": GOTO 10
```

The second program has you enter Sides SA and SB and Angle AC. It then uses COS to calculate Side SC.

```
5   CLS
10  INPUT "WHAT IS ANGLE C (AC)"; AC: IF AC<0 OR AC>180
    THEN 100
20  AC=AC/57.29577951: 'CONVERT DEGREES TO RADIANS
30  INPUT "WHAT IS SIDE A (SA)"; SA:IF SA<=0 THEN 100
40  INPUT "WHAT IS SIDE B (SB)"; SB:IF SB=<0 THEN 100
50  SC=((SA↑2)+(SB↑2))-(2*(SA*SB*COS(AC))): IF SC<0
    THEN 100
60  PRINT "SIDE C (SC) IS" SQR(SC) "LONG": GOTO 10
100  PRINT "SORRY, NOT A TRIANGLE, TRY AGAIN": GOTO 10
```

The third program has you enter Side SB and Angle AA. (Angle AC must be 90 degrees.) It then uses TAN to calculate Side SA.

```
5   CLS
10  INPUT "WHAT IS SIDE B (SB)"; SB: IF SB<=0 THEN 100
20  INPUT "WHAT IS ANGLE A (AA)"; AA: IF AA<=0 OR
    AA>=180 THEN 100
30  AA=AA/57.29577951 'CONVERT DEGREES TO RADIANS
40  SA=SB*(TAN(AA)): IF SA<=0 THEN 100
50  PRINT "SIDE A (SA) IS" SA "LONG": GOTO 10
100  PRINT "SORRY, NOT A TRIANGLE, TRY AGAIN": GOTO 10
```

The fourth program has you enter Sides SA and SC and Angle AB. It then uses TAN and ATN to calculate Angles AA and AC.

```
10  CLS
20  INPUT "WHAT IS SIDE A (SA)"; SA: IF SA<=0 THEN 150
30  INPUT "WHAT IS SIDE C (SC)"; SC: IF SC<=0 THEN 150
40  INPUT "WHAT IS ANGLE B (AB)"; AB: IF AB<=0 OR
    AB>=180 THEN 150
50  X=(180-AB): 'AA+AC=180-AB
60  X=X/57.29577951: 'CONVERT DEGREES TO RADIANS
70  Y=((SA-SC)/(SA+SC))*TAN(X/2)
80  Z=ATN(Y)
90  AA=(X/2)+(Z)
100  AC=(X/2)-(Z)
110  AA=AA*57.29577951: 'CONVERT RADIANS TO DEGREES
120  AC=AC*57.29577951: 'CONVERT RADIANS TO DEGREES
130  PRINT "ANGLE A (AA) IS" AA "DEGREES"
140  PRINT "ANGLE C (AC) IS" AC "DEGREES": GOTO 20
150  PRINT "SORRY, NOT A TRIANGLE, TRY AGAIN": GOTO 20
```

The trignometry functions use *radians*, rather than degrees, to measure an angle. So, each of the above programs converts degrees to radians and radians to degrees. These are the formulas we used to make these conversions:

Degrees to Radians: Degrees/57.29577951
Radians to Degrees: Radians*57.29577951

Logarithms and Exponentials (The LOG and EXP Functions)

The next group of functions let you calculate *natural logarithms* and *natural exponentials* of numbers. They help with higher mathematic operations.

LOG calculates the natural logarithm of a number. Its syntax is:

LOG *(number)*
number is greater than zero.

The natural logarithm of a number is the power to which 2.718281828 (the *base*) must be raised to result in the number. For example, type:

```
PRINT LOG (8) (ENTER)
```

The screen displays 2.07944154 (the natural logarithm). This is because 2.718281828 (the base) must be raised to the 2.07944154 power to result in 8 (the number).

The *logarithm* of a number is the same as the natural logarithm, except the base does not have to be 2.718281828; it can be any number. You can also use LOG to calculate the logarithm of a number by using this formula:

LOG (*number*)/LOG (*base*)

For example, type:

```
PRINT LOG(8)/LOG(2) (ENTER)
```

The screen displays 3 (the logarithm). This is because 2 (the base) must be raised to the third power to result in 8 (the number).

EXP calculates the *natural exponential* of a number. Its syntax is:

EXP (*number*)
number is less than 87.3365.

The natural exponential of a number is 2.718281828 raised to that number. For example, type:

```
PRINT EXP (8) (ENTER)
```

The screen displays 2980.95799 (the natural exponential). This is because 2.718281828 raised to the 8th power results in 2980.95799.

Creating Your Own Function
The DEF FN Command

The DEF FN command lets you create or *define* your own numeric function. Its syntax is:

DEF FN *name (dummy variables)* = *formula*
name is the name of your function.

dummy variables are the variables that your formula uses.

formula is the operation that your function does.

As an example of DEF FN, type this line:

```
10  DEF FNTWO(N)=N*2
```

This line defines a function named TWO. The TWO function does a simple operation. It multiplies any number by 2.

Add these lines to the program:

```
20  INPUT X
30  PRINT FNTWO(X)
40  GOTO 20
```

Run the program. Line 30 uses the TWO function to multiply the number you enter by 2.

As another example of DEF FN, consider a math operation that you had to do earlier in this chapter's trigonometry programs, converting between degrees and radians. With DEF FN, you can define and use your own function that does the conversion.

Try doing this in the first trigonometry program (the program that uses SIN to calculate Sides SA and SB). Add Line 7 to define the conversion function. Then change Line 60 to use the conversion function.

```
7  DEF FNR(X)=X/57.29577951
60  AA=FNR(AA): AB=FNR(AB): AC=FNR(AC)
```

Note that whenever you use DEF FN, be sure to define a function before using it. Otherwise, a ?UF ERROR (Undefined Function Error) occurs.

DO-IT-YOURSELF PROGRAM 36-1

Use DEF FN to:

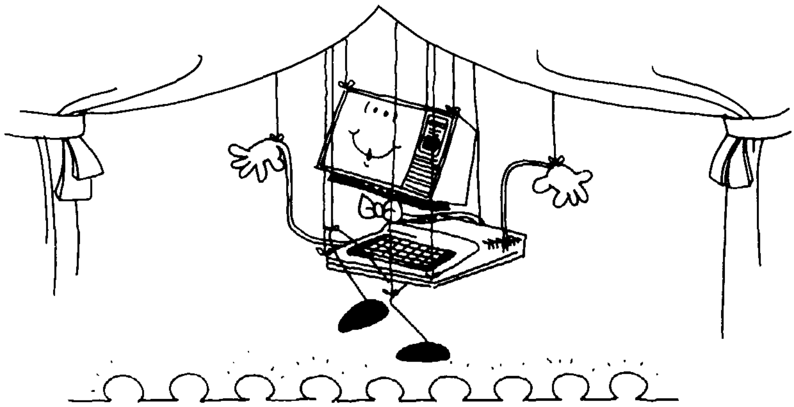
1. Convert radians to degrees.
2. Create a math function that cubes numbers.

A quick reference table of many useful mathematical formulas (plane geometry, trig, and algebra) is in "Odds and Ends."

Learned in Chapter 36

COMMAND	FUNCTIONS
DEF FN	SQR
	SIN
	COS
	TAN
	ATN
	LOG
	EXP
	FIX
	SGN
	ABS

37 / STRINGS



This chapter lists three functions and one command you can use to manipulate strings.

Displaying Strings of Characters (The STRING$ Function)

The STRING$ function lets you create a string of characters. You can produce graphs, tables, and any other text display. Its syntax is:

> STRING$ *(length,character)*
>
> *length* is a number from 0 to 255.
>
> *character* is either the character enclosed in quotes or the numeric code of the character. (For the numeric code of each character, see "Character Codes.")

As an example of STRING$, type:

```
PRINT STRING$(30,"A") (ENTER)
```

The screen displays 30 A's.

As another example, change the Lines program as follows:

```
5   CLS
6   X$=STRING$ (13,"*")
7   PRINT @96, X$; "LINES";X$
9   FOR X=1 TO 1000: NEXT X
10   PMODE 3,1
15   PCLS
20   SCREEN 1,1
25   LINE (0,0)-(255,191),PSET
30   LINE (0,191)-(255,0),PSET
40   GOTO 40
```

Line 6 assigns X$ the value STRING$(13,"*"), a string of 13 asterisks.

Line 7 tells the computer to print (starting at Print Screen Location 96) X$, then the word LINES, followed again by X$ again. (See the Text Screen Worksheet in the Part 7.) Because X$ equals 13 asterisks (*), those characters are printed before and after LINES.

To spruce up the title even more, add these two lines:

```
8   Y$=STRING$(31,42): PRINT @ 384,Y$
```

This time, you tell the computer to display the character represented by Code 42, which represents an asterisk.

DO-IT-YOURSELF PROGRAM 37-1

Have you ever written lists to check off jobs that you or other people have to do?

Using STRING$, write a program that creates a check-off list.

Converting Numbers to Strings (The STR$ Function)

The STR$ function converts a string to a number. Its syntax is:

STR$(*number*) Returns a string containing number.

This short program shows how STR$ works:

```
10   INPUT "TYPE A NUMBER"; N
20   A$=STR$(N)
30   PRINT A$ + " IS NOW A STRING"
```

Searching for Strings (The INSTR Function)

The INSTR function lets you search through one string for a another string. Its syntax is:

INSTR (*position, search-string,target*)

position specifies the position in the *search-string* where the search is to begin (0 to 255). If you omit *position*, the computer automatically begins at the first character.

search-string is the string to be searched.

target is the string for which to search.

INSTR returns a 0 if any of the following is true:

- The *position* is greater than the number of characters in the *search-string*.
- The *search-string* is null (contains no characters).
- INSTR cannot find the *target*.

This program shows how INSTR works:

```
5   CLEAR 500
10  CLS
15  INPUT "SEARCH TEXT";S$
20  INPUT "TARGET TEXT";T$
25  C=0: P=1 'P=POSITION
30  F=INSTR(P,S$,T$)
35  IF F=0 THEN 60
40  C=C+1
45  PRINT LEFT$ (S$,F-1)+STRING$(LEN(T$), CHR$(128)) +
    RIGHT$(S$,LEN(S$)-F-LEN(T$)+1)
50  P=F+LEN(T$)
55  IF P<=LEN(S$)-LEN(T$)+1 THEN 30
60  PRINT "FOUND";C; "OCCURRENCES"
```

Here is a sample run of the above program. However, you can enter whatever text you need.

```
SEARCH TEXT? YOU SHOULD TRY TO USE YOUR COLOR COMPUTER
AS MUCH AS POSSIBLE.
TARGET TEXT? CO
YOU SHOULD TRY TO USE YOUR [][]LOR COMPUTER AS MUCH AS
POSSIBLE. YOU SHOULD TRY TO USE YOUR COLOR [][]MPUTER AS MUCH AS
POSSIBLE
FOUND 2 OCCURRENCES
OK
```

This is how the program works:

1. Line 15 assigns S$ (search) the value, YOU SHOULD TRY TO USE YOUR COLOR COMPUTER AS MUCH AS POSSIBLE.
2. Line 20 assigns T$ (target) the value of CO.
3. Line 30 tells the computer to start searching for T$ at the first position (P) in S$.
4. In Lines 45 and 55, INSTR locates T$ and then prints and blocks out T$ (CHR$(128)). It searches for the next occurrence of T$ and does the same.
5. Line 60 tells the computer to display the number of occurrences of T$ in S$.

DO-IT-YOURSELF PROGRAM 37-2

Write a program that returns the first and second occurrences of the B in ABCDEB.

The following data storage program contains a mailing list of names and addresses. This is an easy way to store information. Notice that we saved storage space by not putting spaces between the words. Doing so makes it difficult for you to read, but not for the computer to do so.

Notice also that we assign a leading asterisk (*) to zip codes so the computer doesn't confuse them with street numbers.

In this case, we're looking for the names and addresses of all individuals who live in the area specified by zip code 650 —. Consequently *650 is the *target* (A$).

```
10  CLEAR 1000
20  CLS
30  A$="*650"
40  X$="JAMES SMITH,6550HARRISON,DALLASTX*75002:SUE
    SIM,RT3,GRAVIOSMO*65084:LYDIA LONG,3445SMITHST,
    ASBURYNJ*32044:JOHNGARDNER,BOX60EDMONTON ALBERTACA"
50  Y$="KERRY FEWELL,45GMAPLE,NEWORLEANS*89667:BILL
    DOLSEIN,6313E121KANSASCITYMO*64134:STEVE HODGES,
    RT4FLORENCEME*65088"
60  Z$="KAREN CROSS,314HURLEYWASHINGTONDC*10011:ASHER
    FITZGERALD,2338HARRISONFTWORTHTX*76101:LIZ DYLAN,
    BOX999NEWYORKNY*86866"
```

So that your computer can search X$, add this line:

```
70  PRINT INSTR(X$,A$)
```

Run the program. Your screen displays:

```
62
OK
```

This tells you the string contains a name and address you need.

What about Y$? Edit Line 50 so the computer searches through those addresses. Does it tell you it found the needed name?

Now, try Z$. Displaying a zero is your computer's way of saying, "There aren't any names you need on this list."

DO-IT-YOURSELF PROGRAM 37-3

Modify the mailing list program so the following are true:

- X$ contains two addresses that have a 650— zip.
- The computer looks for every occurrence of *650, not only for the first.

Replacing Strings (The MID$ Command)

The MID$ command gives you a powerful string editing capability by letting you replace a portion of one string with another. The syntax of MID$ is as follows:

MID$ (*oldstring,position,length*) = *newstring*

> *oldstring* is the variable-name of the string to replace.
>
> *position* is the number of the position of the first character to be changed.
>
> *length* is a number of characters to replace. If you omit *length*, the computer replaces all of *oldstring*.
>
> *newstring* is the string that replaces the specified portion of oldstring.
>
> **Note:** If *newstring* has fewer characters than *length* specifies, the computer substitutes all of *newstring*. *newstring* is always the same length as *oldstring*.

As an example of MID$, run this program:

```
5  CLS
10  A$="KANSAS CITY, MO"
20  MID$(A$,14)="KS"
30  PRINT A$
```

Line 10 assigns A$ the value KANSAS CITY, MO. Then Line 20 tells the computer to use MID$ to replace part of the *oldstring* (A$) with KS, starting at Position 14.

Change Position 14 to 8 and run the program. The result is:

```
KANSAS KSTY, MO
```

Now, add the *length* option to Line 20:

```
20  MID$(A4,14,2)="KS"
```

Notice that it doesn't affect the result because *newstring* and *oldstring* are both two characters long. Change length to 1:

```
20  MID$(A$,14,1)="KS"
```

The computer replaces only one character in *oldstring*, using the first character in KS.

MID$ is doubly effective when used with INSTR. Using the two, you can "search and destroy" text. INSTR searches; MID$ changes, or "destroys." The following program illustrates this:

```
5  CLS
10  INPUT "ENTER A MONTH AND DAY (MM/DD), ";X$
20  P=INSTR(X$,"/")
30  IF P=0 THEN 10
40  MID$(X$,P,1)="-"
50  PRINT X$ " IS EASIER TO READ, ISN'T IT?"
```

In this program, INSTR searches for a slash (/). When it finds one, MID$ replaces it with a hyphen(-).

Characters and Codes
The ASC and CHR$ Functions

Every character has a numeric value. These numeric values are called ASCII codes. (For you technical types, ASCII stands for the American Standard Code for Information Interchange.) This is how the computer works with characters. To see what these values are, look at Table 7.1 in Part 7 of this book. Look up the letter 'A'. Note that it has a decimal code value of 65. Let's use BASIC's CHR$ function to print the character that has a value of 65. Type:

```
PRINT CHR$(65) (ENTER)
```

The computer prints:

```
A
```

The syntax for CHR$ is:

`CHR$` *n*
 Returns the character corresponding to character code *n*

As you might have already guessed, BASIC has a function for going the other direction as well. ASC converts a string such as "A" to an ASCII value, in this case 65. Type:

```
PRINT ASC("A") (ENTER)
```

The computer prints:

```
65
```

The syntax for ASC is:

`ASC (`*string*`)`
 Returns the ASCII code of the first character in *string*

Now let's combine ASC and CHR$ to create a secret message encoder. Our secret message encoder will take each letter and move it up to the next leader. 'A' becomes 'B', 'B' becomes 'C', on through 'Y' becoming 'Z'. What happens to 'Z'? 'Z' becomes 'A'! With this code, the word CAT becomes DBU, and DOG becomes EPH. Type in:

```
10  A$=INKEY$:IF A$="" THEN 10:REM WAIT FOR KEY
20  IF A$ < "A" OR A$ > "Z" THEN 10:REM ONLY A-Z
30  B=ASC(A$):REM GET ASCII CODE OF KEY PRESSED
40  B = B + 1:REM MOVE UP TO NEXT LETTER
50  IF A$ = "Z" THEN B = 65:REM IF 'Z', THE MAKE 'A'
60  PRINT CHR$ (B) ;
70  GOTO 10
```

RUN the program, and type in your secret message. Just for fun, can you figure out how to make the program print spaces when you press the space bar? Can you write a program to decode the messages you write?

DO-IT-YOURSELF PROGRAM 37-4

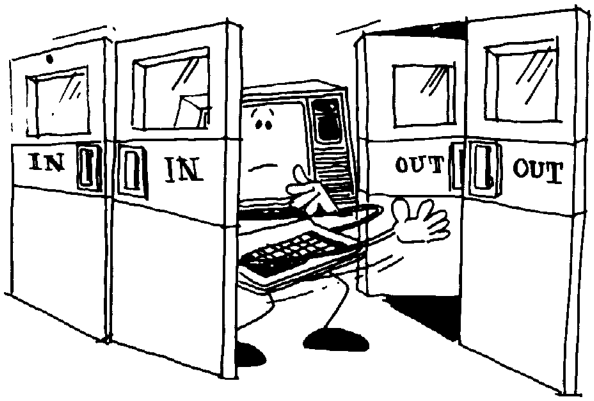
Pretend you worked at a telephone company in the days when telephone exchanges were being switched from alpha-characters to numeric-characters. Write a program that uses MID$ to replace all alpha-exchanges with numbers. Be sure to clear enough string space or you get a ?OS ERROR.

Learned in Chapter 37

COMMAND	FUNCTIONS
MID$	STRING$
	INSTR
	STR$
	ASC
	CHR$

38 / IN AND OUT

Input/output statements let you send data from the keyboard to the computer, from the computer to the TV, and from the computer to the printer. These functions are primarily used inside programs to send out data, results and messages.



Another Way Of Inputting (The LINE INPUT Command)

The LINE INPUT command is similar to INPUT, except for these differences:

- When the statement executes, the computer does not display a question mark while awaiting keyboard input.
- Each LINE INPUT statement can assign a value to only one variable.
- The computer accepts commas and quotation marks as part of the string input.
- Leading blanks, rather than being ignored, become part of the string variable.

Its syntax is:

LINE INPUT *"prompt" string variable*

prompt is the prompting message.

string variable is the name assigned to the line that is input from the keyboard.

With LINE INPUT, you can enter string data without worrying about accidentally including delimiters such as commas, quotation marks, and colons. The computer accepts everything. In fact, some situations require that you enter commas, quotation marks, and leading blanks as part of the data.

Examples:

```
LINE INPUT X$ (ENTER)
```

This command lets you enter X$ without displaying any prompt.

```
LINE INPUT "LAST NAME, FIRST NAME?";N$(ENTER)
```

This command displays the prompt "LAST NAME, FIRST NAME?" and enters data. Commas do not terminate the input string. Notice that the *prompt* includes the question mark and the following space.

To understand LINE INPUT better, enter and run the following program:

```
10  CLEAR 300: CLS
20  PRINT "LINE INPUT STATEMENT": PRINT
30  PRINT:PRINT "*** ENTER TEXT ***"
40  '*** GET STRING, THEN PRINT IT ***
50  A$= "" 'SET A$ TO NULL STRING
60  LINE INPUT "==>";A$
70  IF A$="" THEN END 'IF ENTER PRESSED WITHOUT TYPING ANYTHING,
    STOP!
80  PRINT A$
90  GOTO 50
```

Customized Printing (The PRINT USING Command)

The PRINT USING command lets you display strings and numbers in a "customized" format. This can be especially useful for accounting reports, checks, tables, graphs, or other output that requires a specific print format.

Here is PRINT USING's syntax:

PRINT USING *format;item-list*

format is a string expression that tells the computer the format to use in printing each item in *item-list*. It consists of *field specifiers* and other characters and is one (or one set).

item-list is the data to be formatted.

Note: PRINT USING does not automatically print leading and trailing blanks around numbers. It prints them only as you indicate in *format*.

You can use the following field specifiers as part of *format*.

#	$$	-
,	**$	↑↑↑↑
**	+	!
$	%	.

The following explains each field specifier, and includes examples of its use.

\# A number sign specifies the position of each digit in the number you enter. The number signs establish the length of the numeric field.

If the field is larger than the number, the computer displays the unused positions to the left of the number as spaces and those to the right as zeros.

```
PRINT USING "#####";66.2(ENTER)
   66
```

If the field is too small for the number, the computer displays the number with a leading % sign.

```
PRINT USING "#";66.2(ENTER)
%66
```

. A period specifies the position of the decimal point in the number you enter. You can place the decimal point at any field location that you established with the number sign. The computer automatically rounds off any digits to the right of the decimal point that don't fit into the field.

```
PRINT USING "#.#";66.25(ENTER)
%66.3
```

```
PRINT USING "##.#";58.76(ENTER)
58.8
```

```
PRINT USING "##.##    "; 1Ø.2,5.3,
66.789,.234(ENTER)
1Ø.2Ø    5.3Ø    66.79    Ø.23
```

Note: In the last example, *format* contains three spaces after the final number sign. These spaces separate the numbers when the computer displays them.

, The comma, when placed in any position between the first digit and the decimal point, displays a comma to the left of every third digit. The comma establishes an additional position in your numeric field. To avoid an overflow (indicated by a leading percent sign), place a comma at every third position in the numeric field. Overflows occur when the field isn't large enough.

```
PRINT USING "#########,"; 12345678
12,345,678
```

```
PRINT USING "#########,"; 123456789
%123,456,789
```

```
PRINT USING "###,###,###"; 123456789
123,456,789
```

** When you place two asterisks at the beginning of the numeric field, the computer fills all unused positions to the left of the decimal with asterisks. The two asterisks establish two more positions in the numeric field.

```
PRINT USING "**####"; 44.0
****44
```

$ When you place a dollar sign ahead of the numeric field, the computer places a dollar sign ahead of the number when displaying it. This, of course, is handy when you are working with money.

```
PRINT USING "$###.##"; 18.6735
$ 18.67
```

$$ When you place two dollar signs at the beginning of the field, the computer displays a floating dollar sign immediately preceding the first digit.

```
PRINT USING "$$##.##"; 18.6735
$18.67
```

****$** When you place this combination of symbols at the beginning of the field, the computer fills the vacant positions to the left of the number with asterisks and places a dollar sign in the position immediately preceding the first digit.

```
PRINT USING "**$.##"; 8.333
*$8.33
```

+ When you place a plus sign at the beginning or end of the field, the computer precedes all positive numbers with a plus sign and all negative numbers with a minus sign.

```
PRINT USING "+**####"; 75200
*+75200

PRINT USING "+###"; -216
-216
```

– When you place a minus sign at the end of the field, the computer follows all positive numbers with a space and precedes all negative numbers with a minus sign.

```
PRINT USING "####.#-"; -8124.420
8124.4-
```

% A percent sign defines a string of spaces or text. The first percent sign starts the string, and the second percent sign defines the end of the string.

```
PRINT USING "%5/3=%##.####";5/3
%5/3=% 1.6667
```

↑↑↑↑ Four up-arrows specifies scientific notation printout. Use the number sign to control the mantissa printout. PRINT USING will add the power of ten exponent prefaced by the letter "E".

```
PRINT USING "##.###↑↑↑↑";55555/3
 1.852E+04
```

! An exclamation point tells PRINT USING to print the first character in a string.

```
PRINT USING "!";"YESTERDAY"
Y
```

To see PRINT USING in use, run the following program:

```
5  CLS
10  A$= "**$##,######.## DOLLARS"
20  INPUT "WHAT'S YOUR FIRST NAME";F$
30  INPUT "WHAT'S YOUR MIDDLE NAME";M$
40  INPUT "WHAT'S YOUR LAST NAME";L$
50  INPUT "ENTER THE AMOUNT PAYABLE";P
60  CLS
70  PRINT "PAY TO THE ORDER OF ";
80  PRINT USING "!";F$;".";M$;".";
90  PRINT L$
100  PRINT:PRINT USING A$;P
110  GOTO 110
```

Line 10 defines the format, using **$ to fill the leading spaces with asterisks and placing a dollar sign directly before the first number. This format is sometimes used to protect checks from being altered.

Line 10 also sets up the numeric field using the # sign. So, whenever you enter a number that is smaller than the numeric field, the computer precedes the number with asterisks to fill the unused spaces. Included in Line 10 are two more field specifiers, the decimal point and the comma.

The computer displays the decimal point at only those positions specified. Because you tell the computer to include two places to the right of the decimal (for cents), the computer rounds all numbers of more than two digits to two digits. If you enter a number that has one or no digits to the right of the decimal point, the computer inserts zeros.

The exclamation marks in Line 80 tell the computer to use only the first character (the initial) of F$ (your first name) and of M$ (your middle name).

DO-IT-YOURSELF PROGRAM 38-1

Change the program so that no leading asterisks appear on the check.

DO-IT-YOURSELF PROGRAM 38-2

Write a program that creates a table showing your income and expenses on a monthly basis. Don't bother to itemize your expenses; simply calculate the totals and the net result (plus or minus).

Use STRING$ to organize the table, making it flexible enough so you can use it month after month without changing the entire program.

Finding Your Position (The POS Function)

The POS function calculates the current cursor position on the low-resolution text screen or the carriage position on the printer. Its syntax is:

POS (*device number*)

device number is 0 (low-resolution text screen) or -2 (printer)

As an example of POS, type:

```
PRINT TAB (8); POS(Ø)
```

The screen displays the number 8 at Column 8 in the current line.

Note: The leading space before 8 causes it to appear in Column 9.

One way to use POS is to disable the *wrap-around* feature on the screen or the printer. Doing this prevents words from being broken in the middle. On the other hand, it necessarily shortens the line length. Run the following program to see POS at work:

```
5   CLS
10  A$=INKEY$
20  IF A$="" THEN 10
30  IF POS (0)>22 THEN IF A$=CHR$(32) THEN A$=CHR$(13)
40  PRINT A$;
50  GOTO 10
```

This program lets you use the keyboard as a typewriter POS watches the end of the line so no word is divided.

In Line 30, the computer checks to see if the "current" cursor position is greater than Column 22. (The screen is 32 columns wide.) If the cursor passes Column 22, the computer begins a new line the next time you press the space bar (CHR$(32)). When the computer decides to begin a new line, it does so by printing a carriage return (CHR$(13)). In effect, the computer presses (ENTER).

DO-IT-YOURSELF PROGRAM 38-3

Write a program that uses POS to space words evenly on a single line.

Finding Your High-Resolution Position (The HSTAT Command)

POS does not work on the high-resolution screen, but its counterpart, HSTAT, returns even more information than POS does.

The HSTAT command returns the x and y coordinates of the current cursor position. It also returns the character at that position, as well as the character's *attributes*, which we discuss shortly.

Unlike POS, which is a function, HSTAT is a command. Its syntax is.

HSTAT *dummy-variable1*, *dummy-variable2*, *dummy-variable3*, *dummy-variable4*

dummy-variable1 is any string variable. After HSTAT is executed, it contains the character at the current position.

dummy-variable2 is any numeric variable. After HSTAT is executed, it contains the attributes of the current position.

dummy-variable3 is any numeric variable. After HSTAT is executed, it contains the x coordinate of the current position.

dummy-variable4 is any numeric variable. After HSTAT is executed, it contains the y coordinate of the current position.

As an example of using HSTAT, type:

```
HSTAT A$, B, C, D
```

After this command is executed, A$ contains the character at the current position, B contains the character at the current position's attributes, C contains the current position's x coordinate, and D contains the current position's y coordinate.

The following program prints an "A" on the screen and then uses HSTAT to check if the "A" is really in the right location.

```
10  WIDTH 40
20  LOCATE 19,11
30  PRINT "A";
40  LOCATE 19,11
50  HSTAT A$,A,X,Y
60  LOCATE 0,14
70  PRINT A$,X,Y
```

To interpret the information regarding a character's attribute, you need to understand machine language. The high-resolution screen uses two bytes to store each character. One byte stores the character code; the other stores the character's attributes.

The attribute byte contains the following information:

Bits 0-2	Background color (Palette 0-7)
Bits 3-5	Foreground color (Palette 8-15)
Bit 6	Underline
Bit 7	Blink at 1/2 second rate

When you execute HSTAT, it returns the actual attribute byte. To be able to interpret this byte, you need to convert it to its binary representation and then figure out which bits are set.

Checking the Time (The TIMER Function)

Your computer has a built-in *timer* that measures time in sixtieths of a second (approximately). The moment you power-up the computer, the timer begins counting at zero. When it counts to 65535 (approximately 18 minutes later), the timer starts over at zero. It pauses during cassette and printer operations.

At any instant, you can see the count of the timer by using the TIMER function. Type:

PRINT TIMER (ENTER)

The TIMER function displays a value in the range 0–65535.

You can also reset the timer to any specified time by typing:

TIMER = *number* (ENTER)

number is in the range 0–65535.

To see TIMER (and PRINT @ USING, another "new" function), run the following program called "Math Quiz." It presents you with a math problem. When you press A, B, C, or D, the computer tells you whether the answer is right or wrong. Then, the computer uses the timer to tell you the time you took to answer (using TIMER).

```
5    WIDTH 32
10   DIM CH(3), L$(3) 'CH(#)=CHOICES, L$=ANSWER FORMATS
20   LL=10:UL=20:  'LOWER LIMIT AND UPPER LIMIT FOR H AND V
30   NV=UL-LL+1
40   P$="WHAT'S ### + ### ?":  'QUESTION FORMAT
50   FOR I=0 TO 3 'INITIALIZE CH( )
60   L$(I)=CHR$(I+65)+") ###"
70   NEXT I
80   CLS
90   X=INT(RND(NV)+LL-.5):  'GET RANDOM X BETWEEN LL
     AND UL
100   Y=INT(RND(NV)+LL-.5):  'GET RANDOM Y BETWEEN LL
      AND UL
110   R=INT(X+Y+.5)  'CORRECT ANSWER
130   FOR I=0  TO 3  'GET MULT. CHOICES
140   CH(I)=INT(RND(NV)+LL-.5)
150   NEXT I
160   RC=RND(4)-1  'MAKE 1 CHOICE RIGHT
170   CH(RC)=R
180   PRINT @ 32, USING P$;X,Y  'DISPLAY PROBLEM
190   FOR LN=3 TO 6
200   PRINT @ LN*32+10, USING L$(LN-3);CH (LN-3)
210   NEXT LN
220   TIMER =0
230   A$=" "  'CLEAR KEYBOARD
240   A$=INKEY$: IF A$=" " THEN 240
250   SV=TIMER 'IF KEY PRESSED, SAVE TIMER CONTENTS
260   IF A$<"A" OR A$>"D" THEN 240  'INVALID KEY -
      GO BACK
265   PRINT @ 8*32+10, A$
270   K=ASC(A$)-65
280   IF CH(K)=R THEN PRINT "RIGHT!":GOTO300
290   PRINT "WRONG! ANSWER IS ";R
300   PRINT "YOU TOOK"; SV/60; "SECONDS"
310   INPUT "PRESS <ENTER> FOR NEXT PROBLEM"; EN
320   GOTO 80
```

Through trial and error, change the upper and lower limits (Line 20) for *h* and *v*. Make the program perform a mathematical operation other than addition, or have the computer keep score, based on your time. Add five seconds for each incorrect answer.

Changing Devices (Using Device Numbers)

Did you ever think of your video display as an "output" device and your keyboard as an "input" device?

With PRINT, PRINT USING, LINE INPUT, and POS, you can use device numbers to direct input or output. For instance, suppose you type:

```
PRINT #-2, USING "###.###";123.45678(ENTER)
```

The screen remains "silent" while the printer prints:

```
123.457
```

You can use any of the available field specifiers with PRINT #-2, USING.

POS(-2) returns the printer's current print position as it changes. Note that the position is figured internally, not mechanically.

LINE INPUT # works similarly, with the difference that it lets you read a "line of data" from a cassette file.

LINE INPUT # reads everything from the first character up to whichever of the following comes first:

- A carriage-return character that is not preceded by a line-feed character
- The 249th data character
- The end-of-file

Other characters encountered (quotes, commas, leading blanks, and line feed/carriage return sequences) are included in the string. For instance:

```
LINE INPUT #-1,A$
```

sends a line of cassette file data into A$.

The following program uses LINE INPUT # to count the number of lines in any cassette-stored program that is CSAVED in ASCII format (using the A option):

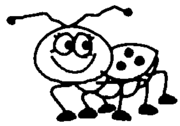
```
10  CLEAR 500
20  LINE INPUT "NAME OF DATA FILE? ";F$
30  K=0 'K IS THE COUNTER
40  OPEN "I",-1,F$
50  IF EOF (-1) THEN 100
60  LINE INPUT #-1, A$
70  K=K+1
80  PRINT A$
90  GOTO 50
100  CLOSE #-1
110  PRINT "FILE CONTAINED";K;"LINES"
```

Learned in Chapter 38

COMMANDS	FUNCTIONS
LINE INPUT	POS
PRINT USING	TIMER
HSTAT	

39 / BUGS

In this chapter, we deal with the inevitable subject of bugs (errors in your program). Although we can't prevent bugs from occurring, we can certainly give you ways to track, trace, and trap bugs so they no longer cause a serious problem.



Tracking Bugs (The STOP and CONT Commands)

BASIC lets you track an error with two commands, STOP and CONT. Their syntaxes are:

STOP Temporarily stops the execution of a BASIC program.

CONT Continues the execution of a BASIC program.

As an example of using these commands, assume you have entered the following BASIC program that does various computations on a number and stores the result in Variable A:

```
10  A = 1
15  FOR N = 1 TO 10
20  A = A + 1
30  A = A * 2
40  NEXT N
```

For some reason, Variable A does not end up containing the number you think it contains. To see what is causing the problem, you could add these lines:

```
25  STOP
35  STOP
```

After adding these lines, you could run the program again. When the computer gets to the first STOP command it prints:

```
BREAK IN 25
```

This indicates that the computer has temporarily stopped program execution, but it has not ended the program. You could then type:

PRINT A (ENTER)

The computer prints the value of A at this point of program execution. To continue program execution, you could type:

CONT (ENTER)

When the computer gets to the next STOP command it prints:

```
BREAK IN 35
```

You could then ask the computer to PRINT A, and the computer prints the value of A at this point of the program execution.

By continuing in this fashion, you could eventually track what is causing the problem and, after correcting the problem, delete the STOP commands from your program.

For long Programs... (MEM)

Clear memory and type:

```
PRINT MEM (ENTER)
```

The computer shows how much memory is currently available for BASIC programming. If you do this right after you turn on the computer, BASIC will print:

```
24872
OK
```

When you are typing in a long program, you may want to PRINT MEM from time to time to make sure that the computer is not running out of memory. If the program is a little on the large side, you can save memory by omitting extra spaces in your program. You can omit spaces before and after punctuation marks, operators, and BASIC words.

Tracing Bugs (The TRON and TROFF Commands)

The TRON/TROFF command lets you trace program execution to see where a bug is occurring. Its syntax is:

```
TRON or TROFF
```

Turns the program tracer on or off.

As an example of using this command, trace the execution of the Lines program. Type:

```
TRON (ENTER)
```

Then, run the program:

```
5   PCLS
10  PMODE 3,1
20  SCREEN 1,1
30  LINE(0,0)-(255,191), PSET
```

The computer displays:

```
[5] [10] [20] [30]
OK
```

This tells you that the computer executed Line 5, 10, 20, and 30, in that order.

To turn off the tracer, type:

```
TROFF (ENTER)
```

Trapping Bugs (The ON ERR GOTO Command)

No matter how smoothly your program runs, it is incomplete if it doesn't anticipate operator errors. ON ERR GOTO is a command that lets you anticipate and "trap" an error—before it causes your program to crash.

The syntax of ON ERR GOTO is:

`ON ERR GOTO` *line number* Goes to line number if an error occurs during program execution.

As an example of using ON ERR GOTO, assume you have written this program:

```
10  PRINT "THIS PROGRAM DIVIDES X BY Y":PRINT
20  INPUT "INPUT A NUMBER AS X"; X
30  INPUT "INPUT A NUMBER AS Y"; Y
40  ANS=X/Y
50  PRINT ANS
60  GOTO 10
```

This program works like a dream, but only if the user knows that the computer can't divide a number by 0. If the user types `0` (ENTER), or simply (ENTER), the program ends and the user is confronted with this message:

```
/0 ERROR IN 40.
```

To trap this error, you could add these lines to the program:

```
5   ON ERR GOTO 100
100  REM ERROR HANDLING ROUTINE
110  PRINT "THE COMPUTER CAN'T DIVIDE A NUMBER BY 0"
120  PRINT "PLEASE TRY AGAIN"
130  GOTO 20
```

Now, when the user enters `0` (ENTER) or (ENTER), these messages appear:

```
THE COMPUTER CAN'T DIVIDE A NUMBER BY 0
PLEASE TRY AGAIN
INPUT A NUMBER AS Y?
```

Trapping the Right Bug (The ERNO Function)

When using an error-trapping routine, be sure you are trapping the right error. For example, assume that Line 50 has a Syntax Error (the word PRINT misspelled):

```
50  PIRNT ANS
```

When the computer executes Line 50, it goes to the same error-trapping routine. This time, it does not go to the error-trapping routine because of a Division by 0 Error. It goes to the routine because of a Syntax Error.

To anticipate this kind of problem, you can use the ERNO function. Its syntax is:

`ERNO` Returns an error number that corresponds to the error that occurred.

ERNO returns an error number, rather than an error message. You can find out the error number that corresponds to each BASIC error message by referring to "BASIC Error Messages" in the back of this book.

Using ERNO, you can change the error-trapping routine in this way:

```
100  REM ERROR HANDLING ROUTINE
105  IF ERNO <> 10 THEN PRINT "FATAL ERROR": END
110  PRINT "THE COMPUTER CAN'T DIVIDE A NUMBER BY 0"
120  PRINT "PLEASE TRY AGAIN"
130  GOTO 20
```

Now, if any error occurs other than Error 10 (Division by 0 Error), the program prints "FATAL ERROR" and ends.

Returning to the Right Trap (The ERLIN Function)

In addition to being sure you trap the right error, you need to be sure that, after you trap the error, you return to the right place in the program.

For example, assume a program includes two lines—Lines 20 and 70—where a Division by 0 Error could occur.

To determine which of these lines causes an error, BASIC has an ERLIN function. Its syntax is:

`ERLIN` Returns the line number where the error occurred.

Using ERLIN, you can change the error-trapping routine in this way:

```
100  REM ERROR HANDLING ROUTINE
105  IF ERNO <> 10 THEN PRINT "FATAL ERROR": END
110  PRINT "THE COMPUTER CAN'T DIVIDE A NUMBER BY 0"
120  PRINT "PLEASE TRY AGAIN"
130  IF ERLIN = 20 THEN 20
140  IF ERLIN = 70 THEN 70
```

Trapping a Break (The ON BRK GOTO Command)

Closely related to the ON ERR GOTO command is the ON BRK GOTO command. This command goes to a *break-trapping routine* if the operator presses the (BREAK) key.

Its syntax is:

`ON BRK GOTO` *line number* Goes to line number if the (BREAK) key is pressed during program execution.

This is a short example of ON BRK GOTO:

```
10  ON BRK GOTO 100
20  PRINT "HELLO"
30  GOTO 20
100  PRINT "IT'S ABOUT TIME YOU STOPPED THAT!"
```

This program prints **HELLO** over and over again until the operator presses the BREAK key. Then, instead of displaying the **BREAK IN 20** or **BREAK IN 30** message, the computer displays:

```
IT'S ABOUT TIME YOU STOPPED THAT!
```

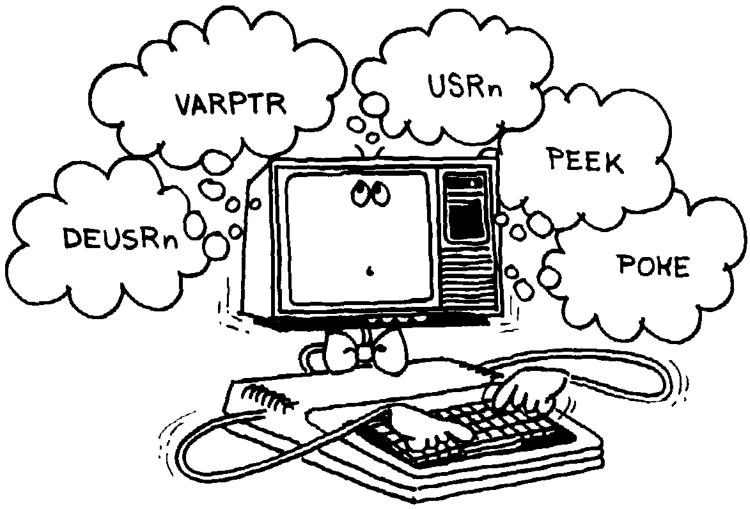
Break-trapping does not have a counterpart to the ERNO and ERLIN functions, but it is still a better alternative than having the program simply stop when the operator presses BREAK.

Learned in Chapter 39

COMMANDS	FUNCTIONS
STOP	ERLIN
CONT	ERNO
MEM	
TRON/TROFF	
ON ERR GOTO	
ON BRK GOTO	

40 / MACHINE-LANGUAGE SUBROUTINES

This chapter assumes you know how to write *machine language (ML) subroutines* (subroutines that contain 6809 instructions). It shows how to call an ML subroutine from a BASIC program.



How to Call an ML Subroutine CLEAR, CLOADM, PEEK, POKE, DEFUSR, and USR

(Working With Machine Language)

To call an ML subroutine from a BASIC program, you need to follow six steps.

1. Assemble the ML Subroutine into Object Code

You can do this yourself, by looking up the code for each instruction, or by using an *assembler* such as EDTASM or Disk EDTASM.

Step 2. Reserve Memory for the ML Subroutine

You can do this using the second parameter of the CLEAR command. The syntax of CLEAR is:

CLEAR *n1*, *n2* Clears *n1* bytes of string space and sets *n2* as the highest address that BASIC can use. (By setting *n2* as the highest address that BASIC can use, you are reserving this area for your ML subroutine.)

For example, this command reserves an area of memory from Addresses 21000 to the top of memory:

```
5  CLEAR 25, 21000
```

BASIC cannot overwrite this area of memory.

3. Store the ML Subroutine in Memory

You can do this by loading it from tape, with the CLOADM command, or by poking it into memory, with the POKE command.

The syntax for CLOADM is:

CLOADM *filename, offset address* loads an ML program or subroutine from cassette tape.

filename is the ML program or subroutine.
offset address is optional; it specifies an offset to add to the ML program or subroutine's loading address.

The syntax for POKE is:

POKE *address*, *n* stores a number in memory.

n	The number you want to store (0-&HFF).
address	The memory address where you want to store the number (0-&HFFFF).

4. Tell BASIC Where the ML Subroutine Is

You can do this using the DEFUSR command. The syntax for DEFUSR is:

DEFUSR*n* = *address* tells where, in memory, an ML subroutine starts

n is the number of the ML subroutine (0-9). *address* is the first address in memory where the ML subroutine is stored.

For example, this command tells BASIC that a routine is stored at address 21000:

```
10  DEFUSR1 21000
```

Notice that we have labeled the routine as subroutine 1.

5. Call the ML Subroutine

You can do this with the USR command or the EXEC command. The syntax of USR is:

dummy variable = USR*n*(*value*) calls an ML subroutine

n is the number of the ML subroutine (0-9).
value is a value you want to pass to the ML subroutine.
dummy variable is a variable you can use to store the data returned by USR.

For example, this command calls the routine at 21000 labeled as subroutine 1 above:

```
110  A=USR1(0)
```

The EXEC command is simpler than the USR command. Use EXEC when you do not need to pass variables to and from the machine-language program. EXEC jumps to the program, and when it is done, returns. The syntax for EXEC is:

EXEC (*address*)
Transfers control to a machine-language program at *address*. If *address* is omitted, control is transferred to the address set by the last CLOADM.

An example of EXEC is:

```
EXEC 21000
```

6. Return from the ML Subroutine

You can do this by using the RTS instruction. This causes the ML subroutine to return without passing any values to USR's *dummy variable.*

Using Stack Space

An ML subroutine, called by USR, that requires more than 30 bytes of stack storage must provide its own stack area. Save BASIC's stack pointer upon entry to the USR function, setting up a new stack pointer and restoring BASIC's stack pointer prior to returning to BASIC. The values of the A, B, X, and CC registers need not be preserved by USR.

Reading and Saving Memory Using PEEK and CSAVEM

BASIC makes it easy to look at your computer's memory, and save its contents. PEEK lets you look inside your computer, and CSAVEM lets you save machine-language programs on tape. Their syntaxes are:

PEEK *(memory location)*
Returns the contents of a memory location (0-65535 decimal, or 0-&HFFFF hexadecimal).

An example of PEEK is:

```
A = PEEK(&H2155)
```

CSAVEM *"filename",l,h,e*
Saves machine-language program *filename* on cassette.

filename	Name of machine-language program being saved. Name can have as many as 8 characters.
l	Lowest address of machine-language program.
h	Highest address of machine-language program.
e	Exec address of machine-language program.

An example of CSAVEM is:

```
CSAVEM "GRAPHICS",&HD000,&HDFE0,&HD003
```

Helpful BASIC Functions (The &H and &O Operators; the HEX$, LPEEK, and LPOKE Functions)

BASIC offers several functions that are helpful when dealing with ML subroutines.

The first are the "&H" operator, which permits using hexadecimal notation and the "&O" operator which permits using octal notation. For example, this command stores the hexadecimal number 2D into Memory Address 21000:

```
POKE 21000, &H2D
```

This command would store the octal number 377 into Memory Address 21000:

```
POKE 21000, &O377
```

The second is HEX$, which converts a decimal number into a hexadecimal string. The syntax of HEX$ is:

HEX$(*n*) returns the hexadecimal value of n as a string.

For example, this command stores "10," the hexadecimal value of 16, into A$:

```
A$=HEX$(16)
```

If you have been using PEEK and POKE, you may have wondered how you can access **all** the memory in your Color Computer 3. PEEK and POKE can only access 0-&HFFFF. Your computer has a good deal more memory than that, and two new functions to let you access it. These new functions are LPEEK and LPOKE. Their syntaxes are:

LPEEK (*virtual memory location*)
Returns the contents of a virtual memory location (0-524287 decimal or 0-&H7FFFF hexadecimal).

LPOKE *virtual memory location,value*
Stores a value (0-255) in a virtual memory location (0-524287 decimal or 0-&H7FFFF hexadecimal).

As you can see, virtual memory addresses have a much wider range than those used for CLOADM, CSAVEM, PEEK, POKE, DEF USR, and EXEC. The memory map in Section 7 lists many important *virtual addresses* in your computer. You can access all of them with LPEEK and LPOKE. To use 0-&H5FFFF, you must have 512K of memory in your computer.

The relationship between regular memory addresses, and virtual memory addresses is easy to understand. **Regular memory addresses 0-&HFFFF correspond to virtual memory addresses &H70000-&H7FFFF.** For example, the regular memory address &H128F is the same as the virtual memory address &H7128F.

Here is an example of LPEEK:

```
PRINT LPEEK(&H60000)
```

Here is an example of LPOKE:

```
LPOKE &H60000,&H255
```

Learned in Chapter 40

COMMANDS	FUNCTIONS	OPERATOR
CLEAR	PEEK	&H
DEFUSR	LPEEK	&O
USR	HEX$	
POKE		
LPOKE		

PART 7 / ODDS AND ENDS

SUGGESTED ANSWERS TO DO-IT-YOURSELF PROGRAMS

Do-It-Yourself Program 4-4

Sounding tones from bottom of range to top and back to bottom:

```
10  FOR X=1 TO 255
20  SOUND X,1
30  NEXT X
40  FOR X=255 TO 1 STEP -1
50  SOUND X,1
60  NEXT X
```

Do-It-Yourself Program 5-1

```
10  PRINT "HOW MANY SECONDS"
20  INPUT S
30  FOR Z = 1 TO 460 * S
40  NEXT Z
50  PRINT S "SECONDS ARE UP!!!"
60  FOR T = 120 TO 180
70  SOUND T,1
80  NEXT T
90  FOR T = 150 TO 140 STEP -1
100  SOUND T,1
110  NEXT T
120  GOTO 50
```

Do-It-Yourself Program 5-2

Lines added to clock program:

```
92  FOR T=200 TO 210 STEP 5
94  SOUND T,1
95  NEXT T
97  FOR T=210 TO 200 STEP -5
98  SOUND T,1
99  NEXT T
```

Do-It-Yourself Program 5-3

```
10  FOR C=0 TO 8
20  CLS C
30  FOR X = 1 TO 460
40  NEXT X
50  NEXT C
```

Do-It-Yourself Program 9-1

```
10   T = RND(255)
14   C = RND(8)
16   CLSC
20   SOUND T,1
30   GOTO 10
```

Do-It-Yourself Program 9-2

```
10   CLS
20   A=RND(6)
30   B=RND(6)
40   R=A+B
50   PRINT @ 200, A
60   PRINT @ 214, B
70   PRINT @ 394, "YOU ROLLED A" R
80   IF R=2 THEN 600
90   IF R=3 THEN 600
100  IF R=12 THEN 600
110  IF R = 7 THEN 500
120  IF R = 11 THEN 500
130  FOR X = 1 TO 800
140  NEXT X
150  CLS
160  PRINT @ 195, "ROLL ANOTHER" R "AND YOU WIN"
170  PRINT @ 262, "ROLL A 7 AND YOU LOSE"
180  PRINT @ 420, "PRESS <ENTER> WHEN READY"
185  PRINT @ 456, "FOR YOUR NEXT ROLL"
190  IPUT A$
200  N=RND(6)
210  Y=RND(6)
220  Z=X+Y
225  CLS
230  PRINT @ 200, X
240  PRINT @ 214, Y
250  PRINT @ 394, "YOU ROLLED A" Z
260  IF Z=R THEN 500
270  IF Z=7 THEN 600
280  GOTO 180
500  FOR X=1 TO 1000
510  NEXT X
515  CLS
520  PRINT @ 230, "YOU'RE THE WINNER"
530  PRINT @ 294, "CONGRATULATIONS!!!"
540  GOTO 630
600  FOR X =1TO1000
610  NEXT X
615  CLS
620  PRINT @ 264, "SORRY, YOU LOSE"
630  PRINT @ 458, "GAMES OVER"
```

Do-It-Yourself Program 10-1

```
5   CLEAR 500
10  DATA TACITURN, HABITUALLY UNTALKATIVE
20  DATA LOQUACIOUS, VERY TALKATIVE
30  DATA VOCIFEROUS, LOUD AND VEHEMENT
40  DATA TERSE, CONCISE
50  DATA EFFUSIVE, DEMONSTRATIVE OR GUSHY
60  N = RND(10)
65  IF INT(N/2) = N/2 THEN N = N - 1
70  FOR X = 1 TO N
80  READ A$
90  NEXT X
110  READ B$
120  PRINT "WHAT WORD MEANS :" B$
130  RESTORE
140  INPUT R$
150  IF R$ = A$ THEN 190
160  PRINT "WRONG"
170  PRINT "THE CORRECT WORD IS :" A$
180  GOTO 60
190  PRINT "CORRECT"
200  GOTO 60
```

Do-It-Yourself Program 12-1

```
10  A$ = "CHANGE A SENTENCE."
20  B$ = "IT'S EASY TO"
30  C$ = B$ + " " + A$
40  PRINT C$
```

Do-It-Yourself Challenger Program (Chap. 12)

```
10  PRINT "TYPE A SENTENCE :"
15  INPUT S$
20  PRINT "TYPE A PHRASE TO DELETE"
23  INPUT D$
25  L=LEN(D$)
30  PRINT "TYPE A REPLACEMENT PHRASE"
35  INPUT R$
40  FOR X=1 TO LEN(S$)
50  IF MID$(S$,X,L)=D$ THEN 100
60  NEXT X
70  PRINT D$ "-- IS NOT IN YOUR SENTENCE"
80  GOTO 20
100  E = X-1+LEN(D$)
110  N$=LEFT$(S$,X-1) + R$ + RIGHT$(S$,LEN(S$) - E)
120  PRINT "NEW SENTENCE IS:"
130  PRINT NS$
```

Do-It-Yourself Program 16-1

```
5  CLS
1Ø  FOR N=12 TO 1 STEP -1
15  PRINT "NOTE";N
2Ø  PLAY STR$(N)
25  FOR I=1 TO 5ØØ: NEXT I
3Ø  NEXT N
```

Do-It-Yourself Program 16-2

Change the following lines:

```
1ØØ  Ø A$ = "T5;C;E;F;L1;G;P4;L4;C;E;F;L1;G"
1Ø5  B$ = "P4;L4;C;E;F;L2;G;E;C;E;L1;D"
11Ø  C$ = "P4;L4;D+;L8;E;G;E;P8;L4;C;L8;D;D+"
115  D$ = "L4;E;C;L2;O3;C;L8;O3;D;L8;O2;B-"
12Ø  E$ = "G;E;L4;G;L1;F;P4;L8;G;F;E;F"
125  F$ = "L2;G;E;L4;C;L8;D;D+;E;G;L4;A;L1;O3;C"
13Ø  X$ = "XA$;XB$;XC$;XD$;XE$;XF$;"
```

Add Line 140:

```
14Ø  PLAY X$
```

Do-It-Yourself Program 20-2

```
5  PMODE 1,1
1Ø  PCLS
2Ø  SCREEN 1,1
3Ø  X=RND(256)-1
4Ø  Y=RND(192)-1
5Ø  C=RND(9)-1
6Ø  PSET (X,Y,C)
7Ø  GOTO 3Ø
```

Do-It-Yourself Program 21-1

```
5  PMODE 1,1
1Ø  PCLS
2Ø  SCREEN 1,1
25  LINE (Ø,Ø)-(255,191), PSET
3Ø  LINE (Ø,191)-(255,Ø), PSET
35  LINE (Ø,Ø)-(255,191), PSET,B
4Ø  GOTO 4Ø
```

Do-It-Yourself Program 21-2

```
5  PMODE 1,1
10  PCLS
20  SCREEN 1,1
30  LINE (72,168)-(200,72), PSET,B 'FRAME
40  LINE (72,72)-(136,36), PSET 'ROOF
45  LINE (200,72)-(136,36), PSET 'ROOF
50  LINE (120,168)-(152,100), PSET,B 'DOOR
55  LINE (152,60)-(168,36), PSET, BF 'CHIMNEY
60  LINE (165,128)-(191,100), PSET, B 'WINDOW
65  LINE (178,128)-(178,100), PSET 'WINDOW PART
70  LINE (165,114)-(191,114), PSET 'WINDOW PART
75  LINE (85,128)-(111,100), PSET, B 'WINDOW
80  LINE (85,114)-(111,114), PSET 'WINDOW PART
85  LINE (98,100)-(98,128), PSET 'WINDOW PART
90  GOTO 90
```

Do-It-Yourself Program 21-3

```
5  PMODE 1,1
10  PCLS
20  SCREEN 1,1
30  Y=0
40  FOR X=0 TO 200 STEP 10
50  OY=Y
60  Y=30-OY
70  LINE (X,100-Y)-(X+10,100-OY), PSET
80  NEXT
90  GOTO 90
```

Do-It-Yourself Program 22-1

```
1  Y=-1
5  CLS
10  PRINT @ 193, "DO YOU WANT TO SEE A SQUARE?"
20  FOR X=1 TO 1000: NEXT X
30  PMODE 1,1
35  PCLS
40  SCREEN 1, Y+1
60  LINE (75,150)-(150,75), PSET, B
70  FOR X=1 TO 1000: NEXT X
75  Y=-Y
80  GOTO 5
```

Do-It-Yourself Program 24-1

```
2   WIDTH 32
5   PMODE 4,1
8   PALETTE 11, Color code
9   PALETTE 1Ø, Ø
1Ø  COLOR 7,6
2Ø  SCREEN 1,1
25  LINE(Ø,Ø)-(255,191), PSET
3Ø  LINE(Ø,191)-(255,Ø), PSET
4Ø  GOTO 4Ø
```

Do-It-Yourself Program 25-2

Make the following changes:

```
22  PCOPY 4 TO 3
32  PCOPY 3 TO 2
42  PCOPY 2 TO 1
```

Delete Lines 11, 21, and 31.

Do-It-Yourself Program 25-3

```
2Ø  PMODE Ø,1
3Ø  SCREEN 1,1:PCLS
4Ø  LINE (RND(255), RND(191))-(RND(255), RND(191)), PSET
5Ø  PMODE Ø,2
55  SCREEN 1,1:PCLS
6Ø  LINE (RND(255), RND(191))-(RND(255), RND(191)), PSET
7Ø  PMODE Ø,3
75  SCREEN 1,1:PCLS
8Ø  LINE (RND)(255), RND(191))-(RND(255), RND(191)), PSET
9Ø  PMODE Ø,4
95  SCREEN 1,1:PCLS
96  PCLS
1ØØ  LINE (RND(255), RND(191))-(RND(255), RND(191)), PSET
11Ø  FOR Z = 1 TO 4
12Ø  PMODE Ø,Z:SCREEN 1,1
13Ø  FOR R = 1 TO 2Ø: NEXT R: NEXT Z
14Ø  GOTO 11Ø
```

Do-It-Yourself Program 26-1

```
10  PMODE 4,1
20  PCLS
30  SCREEN 1,0
40  FOR RADIUS = 1 TO 100 STEP 10
50  CIRCLE (128,96), RADIUS
60  NEXT RADIUS
70  GOTO 70
```

Do-It-Yourself Program 26-3

```
5  PMODE 4,1
10  PCLS
20  SCREEN 1,0
30  CIRCLE (200,40),30,,1,.13,.63
40  CIRCLE (230,10),52,,1,.29,.48
50  GOTO 50
```

Do-It-Yourself Program 26-4

```
5  PMODE 1,1
10  SCREEN 1,0
15  PCLS 3
20  COLOR 1,0
25  CIRCLE (200,40),30,,1,.13,.63 'MOON
30  CIRCLE (230,10),52,,1,.29,.48 'MOON
35  LINE (100,185)-(180,125), PSET,B 'HOUSE FRAME
40  LINE -(140,85), PSET 'ROOF
45  LINE -(100,125), PSET 'ROOF
55  LINE (110,160)-(125,130),PSET,B 'WINDOW
60  LINE (155,160)-(170,130), PSET,B 'WINDOW
70  LINE (130,130)-(149,185), PSET,B 'DOOR
75  PSET (134,157,1) 'DOOR KNOB
80  LINE (160,105)-(160,90), PSET 'CHIMNEY
85  LINE -(175,90), PSET 'CHIMNEY
90  LINE -(175,115),PSET 'CHIMNEY
100  'SMOKE STARTS HERE
105  X=167:Y=89 'CIRCLE CENTERPOINT
110  SP=0: EP=0 'CIRCLE START AND END POINT
115  FOR R=1 TO 50 STEP .05 'CIRCLE RADIUS
120  EP=EP+.02: IF EP>1 THEN EP=0
125  CIRCLE (X+R, Y-R),R,4,1,SP,EP 'SMOKE
130  NEXT R
200  GOTO 200
```

Do-It-Yourself Program 27-1

Delete Line 40 and add Line 65:

```
65  PAINT (150,100),8,8
```

Do-It-Yourself Program 27-3

```
5  PMODE 1,1
1Ø  PCLS
15  SCREEN 1,Ø
2Ø  PCLS 3
25  COLOR 1,Ø
3Ø  CIRCLE (2ØØ,3Ø),15
35  PAINT (2ØØ,3Ø),2,1
4Ø  LINE (1ØØ,185)-(18Ø,125),PSET,B
45  LINE -(14Ø,9Ø), PSET
5Ø  LINE -(1ØØ,125), PSET
55  PAINT (135,115),4,1
6Ø  LINE (11Ø,16Ø)-(125,13Ø), PSET,B
65  LINE (155,16Ø)-(17Ø,13Ø), PSET,B
7Ø  LINE (134,157)-(41,185), PSET,B
75  PAINT (12Ø,18Ø),Ø,1
8Ø  LINE (13Ø,13Ø)-(149,185), PSET,B
85  LINE (1Ø1,135)-(41,185), PSET,B
9Ø  LINE (91,14Ø)-(51,185), PSET,B
95  PAINT (55,138),Ø,1
1ØØ  PAINT (89,183),4,1
1Ø5  FOR X=1 TO 5ØØ: NEXT X
11Ø  PAINT (89,183),2,1
115  FOR X=1 TO 5ØØ: NEXT X
12Ø  PAINT (89,155),4,1
14Ø  GOTO 11Ø
```

Do-It-Yourself Program 28-1

```
5  PMODE 4,1
1Ø  PCLS
2Ø  SCREEN 1,Ø
3Ø  DRAW "BM68,116;E2Ø;BE2Ø;E2Ø;F2Ø;BF2Ø;F2Ø;L4Ø;BL4Ø;
    L4Ø;BU4Ø;R4Ø;BR4Ø;R4Ø;G2Ø;BG2Ø;G2Ø;H2Ø;BH2Ø;H2Ø;
    BM128,96;NU4Ø;ND4Ø;NE2Ø;NF2Ø;NG2Ø;NH2Ø;NL4Ø;R4Ø"
4Ø  GOTO 4Ø
```

The star you created probably isn't as fancy as this one because you haven't been introduced to B or N yet. But don't worry; you will be before the end of the chapter.

Do-It-Yourself Program 28-2

```
5  PMODE 4,1
1Ø  PCLS
2Ø  SCREEN 1,1
25  DRAW "BM4Ø,8Ø;U4Ø;R4Ø;D4Ø;L4Ø"
3Ø  DRAW "BM+2Ø,2Ø;U4Ø;R4Ø;D4Ø;L4Ø"
4Ø  LINE (6Ø,1ØØ)-(4Ø,8Ø), PSET
5Ø  LINE (6Ø,6Ø)-(4Ø,4Ø), PSET
6Ø  LINE (1ØØ,6Ø)-8Ø,4Ø), PSET
7Ø  LINE (1ØØ,1ØØ)-(8Ø,8Ø), PSET
8Ø  GOTO 8Ø
```

Do-It-Yourself Program 28-3

```
5  PMODE 4,1
10  PCLS
20  SCREEN 1,1
25  DRAW "BM50,50L30D30R30D30L30"
30  DRAW "BM90,50D60R30U60"
40  DRAW "BM160,50D60R30BU60L30D30R30"
50  GOTO 50
```

Do-It-Yourself Program 28-4

```
5  PMODE 4,1
10  PCLS
20  SCREEN 1,0
30  DRAW "BM98,96;NU80;NE56;NR80;NF56;ND80;NG56; NL80;NH56"
40  CIRCLE (98,96),80,1,1,.125,1
50  CIRCLE (135,110),80,1,1,1,.125
60  LINE (135,110)-(190,167), PSET
70  LINE (135,110)-(213,110), PSET
80  GOTO 80
```

Do-It-Yourself Program 28-5

```
1  CLEAR 2500
5  DIM AZ$(25)
6  FOR LE=0 TO 25
10  READ AZ$(LE)
15  NEXT LE
20  NC$="BR4BU7"  'NEXT CHARACTER
25  NL$="BD4"  'NEXT LINE
30  BS$="BL9"  'BACKSPACE
35  HM$="BM5,10"  'HOME POSITION
100  CW=6: CH=8 'SIZE OF CELL
110  R1=7: R24=191 'ROW POSITION
120  C1=8: C42=247 'COLUMN POSITION
125  CC=1: CL=1 'CURRENT ROW/COL
200  PMODE 4,1
210  PCLS
220  SCREEN 1,0
230  DRAW HM$
250  A$=INKEY$: IF A$="" THEN 250
260  IF "A" > A$ OR "Z" < A$ THEN 250
262  CC=CC+1
265  IF CC>27 THEN DRAW NL$: FOR I=1 TO 27:
     DRAW BS$: NEXT I:CC=1: GOTO 270
269  DRAW NC$
270  DRAW AZ$ (ASC(A$)-65)
290  GOTO 250
```

```
1000  'A
1010  DATA BD1D6U4NR5U2E1R3F1D6
1020  'B
1030  DATA ND7R4F1D1G1NL4F1D2G1NL4BR1
1040  'C
1050  DATA BD1D5F1R3E1U1BU3U1H1L3G1BD6BR5
1060  'D
1070  DATA D7R4E1U5H1L4BD7BR5
1080  'E
1090  DATA NR5D3NR4D4R5
1100  'F
1110  DATA NR5D3NR4D4BR5
1120  'G
1130  DATA BD1D5F1R3E1U2NL2BU2U1H1L3G1BD6BR5
1140  'H
1150  DATA D7U4R5NU3D4
1160  'I
1170  DATA R4L2D7L2R4BR1
1180  'J
1190  DATA BD5D1F1R3E1U6BD7
1200  'K
1210  DATA D7U4R3E2NU1G2F2D2
1220  'L
1230  DATA D7R5
1240  'M
1250  DATA ND7F2ND7E2D7BR1
1260  'N
1270  DATA ND7D1F5ND1U6BD7
1280  'O
1290  DATA BD1D5F1R3E1U5H1L3G1BD6BR5
1300  'P
1310  DATA ND7R4F1D2G1L4BD3BR5
1320  'Q
1330  DATA BD1D5F1R3E1U5H1L3G1D4BR3F2
1340  'R
1350  DATA ND7R4F1D1G1NL4F1D3
1360  'S
1370  DATA BD1D1F1R3F1D2G1L3H1BU5E1R3F1BD6
1380  'T
1390  DATA R4L2D7BR3
1400  'U
1410  DATA D6F1R3E1U6BD7
1420  'V
1430  DATA D5F2E2U5BD7BR1
1440  'W
1450  DATA D7E2NU5F2U7BD7BR1
1460  'X
1470  DATA D1F5D1BL5U1E5U1BD7
1480  'Y
1490  DATA D2F2ND3E2U2BD7BR1
1500  'Z
1510  DATA R5D1G5D1R5
```

Do-It-Yourself Program 28-6

```
5   PMODE 3,1
10  PCLS
15  SCREEN 1,0
20  DRAW "BM50,170;U80;NG30;E80;F80;NF30;D80;U70;
    L50;U60;L60;D60;L50"
25  LINE (50,170)-(170,170), PSET
30  LINE (160,170)-(160,100), PSET
31  LINE (100,170)-(100,100), PSET
32  LINE -(160,100), PSET
33  LINE (110,145)-(120,145), PSET
35  FOR X=1 TO 500: NEXT X
40  LINE (100,170)-(160,170), PRESET
45  LINE (120,180)-(120,110), PSET
50  LINE (160,100)-(120,110), PSET
55  LINE (160,170)-(120,180), PSET
56  LINE (110,145)-(120,145), PRESET
59  FOR X=1 TO 300: NEXT X
60  LINE (120,180)-(120,110), PRESET
65  LINE (160,100)-(120,110), PRESET
70  LINE (160,170)-(120,180), PRESET
75  DRAW "BM110,170;BU70;BR50;G25;D70;E25;BU35;BG15;G5"
80  CIRCLE (130,125),10,,1,.135,.9
85  DRAW "BM130,130;D15;D15;G10;E10;U15;L10"
90  LINE (120,145)-(120,135), PSET
91  FOR X=1 TO 60: NEXT X
95  LINE (120,145)-(120,135), PRESET
96  FOR X=1 TO 120: NEXT X
100  LINE (120,145)-(110,145), PSET
101  FOR X=1 TO 60: NEXT X
110  LINE (120,145)-(120,135), PSET
120  FOR X=1 TO 120: NEXT X
121  CIRCLE (130,125),10,1
122  DRAW BM130,130;C1;D30;G10;E10;U15;L10"
125  DRAW "BM110,170;BU70;BR50;C1;G25;D70;E25;BU35;BG15;G5"
130  COLOR 4,1
135  LINE (120,180)-(120,110), PSET
140  LINE (160,100)-(120,110), PSET
141  LINE (120,145)-(110,145), PRESET
145  LINE (160,170)-(120,180), PSET
146  FOR X=1 TO 300: NEXT X
150  LINE (120,180)-(120,110), PRESET
155  LINE (160,100)-(120,110), PRESET
160  LINE (120,180)-(160,170), PRESET
170  LINE (100,170)-(160,170), PSET
175  GOTO 20
```

Do-It-Yourself Program 29-1

```
5  PCLEAR 4
10  PMODE 4,1
15  PCLS
20  SCREEN 1,1
25  DIM V(35,35)
30  X=10: Y=10
35  DRAW "BM10,10;S2;H10;R15;F10;R20;F10;G10;L20;G10;
    L15;E10;U20;D4;NL8'D4'NL12'D4NL16;D4;NL12;D4;NL8"
40  GET (X-X,Y-Y)-(X*3.5,Y*3.5),V,G
45  A$=INKEY$: IF A$=" " THEN 45 'PRESS ANY KEY TO START
50  PCLS
55  FOR A=10 TO 200 STEP 5
60  PUT (X+A,Y)-(X+A+35,Y+35),V,PSET
65  NEXT A
70  PCLS
75  GOTO 55
```

Notice that we've used the options for both GET and PUT. If you want this rocket to go faster, delete the options and switch to Mode 3.

Do-It-Yourself Program 32-1

```
5  CLS: PRINT "POSITION TAPE - PRESS PLAY AND RECORD:"
7  INPUT "PRESS <ENTER> WHEN READY";R$
10  OPEN "O", #-1, "CHECKS"
15  CLS: PRINT "INPUT CHECKS - PRESS <XX> WHEN FINISHED"
20  INPUT "NUMBER:"; N$
25  IF N$ = "XX" THEN 90
30  INPUT "DATE :"; D$
40  INPUT "PAYABLE TO :"; P$
50  INPUT "ACCOUNT :"; S$
60  INPUT "AMOUNT :$"; A
70  PRINT #-1, N$, D$, P$, S$, A
80  GOTO 15
90  CLOSE #-1
92  CLS: T=0
95  INPUT "WHICH ACCOUNT:"; B$
100  PRINT "REWIND TAPE - PRESS PLAY"
110  INPUT "PRESS <ENTER> WHEN READY"; R$
120  OPEN "I", #-1, "CHECKS"
130  IF EOF (-1) THEN 170
140  INPUT #-1, N$, D$, P$, S$, A
150  IF B$ = S$ THEN T = T+A
160  GOTO 130
170  CLOSE #-1
180  PRINT "TOTAL SPENT ON-" B$, "IS $" T
```

Do-It-Yourself Program 33-1

```
10  DATA 33, 12, 42, 13, 15, 23
20  DATA 25, 30, 33, 27, 14, 8
30  DIM I(12)
40  FOR X=1 TO 12
50  READ I(X)
60  NEXT X
70  INPUT "ITEM NO."; N
75  IF N>12 THEN 70
80  PRINT "INVENTORY FOR ITEM" N "IS" I(N)
90  GOTO 70
```

Do-It-Yourself Program 33-2

```
5  DIM T(52)
7  DIM D(52)
10  FOR X=1 TO 52
20  T(X)=X
30  NEXT X
34  CLS
36  PRINT @ 101, "... DEALING THE CARDS"
40  FOR X=1 TO 52
50  C=RND(52)
60  IF T(C)=0 THEN 50
70  D(X)=C
75  SOUND 128,1
80  T(C) = 0
100  NEXT X
110  CLS
120  PRINT @ 107, "YOUR HAND"
130  PRINT @ 167, " "
140  FOR X = 1 TO 5
150  PRINT D(X);
160  NEXT X
```

Do-It-Yourself Program 34-1

Lines that change items:

```
110  INPUT "WHICH ITEM NO. DO YOU WANT TO CHANGE"; N
115  IF N>12 THEN 110
120  INPUT "WHAT IS THE REPLACEMENT ITEM"; S$(N)
130  GOTO 80
```

The appendix has a sample program that adds and deletes items from this list.

Do-It-Yourself Program 34-2

Lines that change the song lyrics:

```
110  PRINT
120  INPUT "WHICH LINE DO YOU WANT TO REVISE"; L
130  PRINT "TYPE THE REPLACEMENT LINE"
140  INPUT A$(L)
150  GOTO 50
```

Do-It-Yourself Program 34-3

```
1 CLEAR 1000
5  DIM A$(50)
7  CLS
10  PRINT "TYPE A PARAGRAPH"
20  PRINT "PRESS </> WHEN FINISHED"
30  X=1
40  A$ = INKEY$
50  IF A$ = "" THEN 40
60  PRINT A$;
70  IF A$ = "/" THEN 105
80  A$(X) = A$(X) + A$
90  IF A$ = "." OR A$ = "?" OR A$ = "!" THEN X = X+1
100  GOTO 40
105  PRINT: PRINT
110  INPUT "(1) PRINT OR (2) REVISE"; R
120  CLS
130  ON R GOSUB 1000, 2000
140  GOTO 105
1000  REM PRINT PARAGRAPH
1010  FOR Y = 1 TO X-1
1020  PRINT A$(Y);
1030  NEXT Y
1040  RETURN
2000  REM REVISE PARAGRAPH
2010  FOR Y=1 TO X-1
2020  PRINT Y "-" A$(Y)
2030  NEXT Y
2040  INPUT "SENTENCE NUMBER TO REVISE"; S
2045  IF S> X-1 OR S<1 THEN 2040
2050  PRINT A$(S)
2060  PRINT "TYPE PHRASE TO DELETE"
2070  INPUT D$
2080  L=LEN(D$)
2090  PRINT "TYPE A REPLACEMENT PHRASE"
2100  INPUT R$
2110  FOR Z = 1 TO LEN(A$(S))
2120  IF MID$(A$(S),Z,L) = D$ THEN 2160
2130  NEXT Z
2140  PRINT D$ "- IS NOT IN YOUR SENTENCE"
2150  GOTO 2060
2160  E = Z-1+LEN(D$)
2170  A$(S) = LEFT$(A$(S),Z-1) + R$ + RIGHT$ (A$(S), LEN(A$(S))-E)
2180  RETURN
```

Do-It-Yourself Program 34-4

Change this line to print on the printer:

```
150  PRINT #-2, A$(Y);
```

Do-It-Yourself Program 35-1

```
10   DIM S$(4), N$(13), T(4,13)
20   DATA SPADES, HEARTS, DIAMONDS, CLUBS
30   FOR X = 1 TO 4
40   READ S$(X)
50   NEXT X
60   DATA ACE, 2, 3, 4, 5, 6, 7, 8, 9, 10, JACK, QUEEN, KING
70   FOR X=1 TO 13
80   READ N$(X)
90   NEXT X
100   FOR S=1 TO 4
110   FOR N = 1 TO 13
120   T(S,N) = (S-1)*13 + N
130   NEXT N,S
140   FOR X=1 TO 52
150   S=RND(4): N=RND(13)
160   IF T(S,N) = 0 THEN 150
170   T(S,N) = 0
180   PRINT N$(N) "-" S$(S),
190   NEXT X
```

Do-It-Yourself Program 36-1

```
5   CLS
10   DEF FNC(X) = X^3
20   INPUT "WHAT NUMBER DO YOU WANT TO CUBE";X
30   X=FNC(X)
40   PRINT X
50   FOR A = 1 TO 75
55   NEXT A
60   GOTO 20
```

Do-It-Yourself Program 37-1

```
5   CLS
10  X$ = STRING$(30,"-")
20  FOR X = 64 TO 352 STEP 64
30  PRINT @ X, X$
40  PRINT @ 97, "BILL"
41  PRINT @ 161, "SUE"
42  PRINT @ 225, "JON"
43  PRINT @ 289, "MARY"
50  PRINT @ 38, "MATH"
51  PRINT @ 45, "SPELL"
52  PRINT @ 53, "READ"
60  PRINT @ 103, "X"
61  PRINT @ 175, "X"
62  PRINT @ 231, "X"
63  PRINT @ 311, "X"
70  NEXT X
80  GOTO 80
```

Do-It-Yourself Program 37-2

```
5   CLS
10  X$ = "ABCDEB"
20  Y$ = "B"
30  PRINT INSTR (X$,Y$); INSTR(4,X$,Y$)
```

Do-It-Yourself Program 37-3

```
15  X=1
20  X$ = "JAMES SMITH,6550HARISON,DALLASTX*75002:SUE
    SIM,RT3,GRAVIOSMO*65084: LYDIA LONG,3445SMITHST,
    ASBURYNJ*32004:BOB STRONG,BOX60,EDMONTONALBERTACA:
    TIMMY DUNTON, PIERMONTMO*65078"
50  P = INSTR(X,X$,A$): PRINT P
60  IF P<> 0 THEN X=P+1: GOTO 50
```

Do-It-Yourself Program 37-4

```
10   DIM TBL$(26)
20   FOR I=0 TO 25
30   READ TBL$(I): NEXT I
40   PRINT "ENTER OLD-STYLE PHONE NUMBER"
50   INPUT N$
60   IF N$=" " THEN 40
70   FOR I=1 TO LEN(N$)
80   C$=MID$(N$,I,1)
90   IF C$<"A" OR C$>"Z" THEN 120
100   C$=TBL$(ASC(C$)-65)
110   MID$(N$,I)-C$
120   NEXT I
130   PRINT "NEW-STYLE = ";N$
140   REM A B C D E F
150   DATA "2","2","2","3","3","3"
160   REM G H I J K L
170   DATA "4","4","4","5","5","5"
180   REM M N O P Q R
190   DATA "6","6","6","7","7","7"
200   REM S T U V W X
210   DATA "7","8","8","8","9","9"
220   REM Y Z
230   DATA "9","9"
```

Do-It-Yourself Program 38-1

```
10   A$ = "$$##,######.## DOLLARS"
```

Do-It-Yourself Program 38-2

```
5   CLS
10   INPUT "INCOME"; I
15   INPUT "EXPENSES"; E
20   N=I-E 'NET GAIN OR LOSS
25   A$ = "$$####.##"
30   B$ = "$$####.##"
35   C$ = "+$$####.##"
40   CLS: PRINT @ 33, "MONTHLY ECONOMIC STATUS REPORT"
45   PRINT @ 96, STRING$ (32,"-")
50   PRINT @ 160, "INCOME"
55   PRINT @ 256, "EXPENSES"
60   PRINT @ 352, "TOTAL (+) OR (-)"
65   PRINT @ 340, STRING$(10,"-")
70   PRINT @ 180, USING A$; I
75   PRINT @ 276, USING B$; E
80   PRINT @ 371, USING C$; N
90   GOTO 90
```

Try modifying this program to keep track of your electricity bills and to store the information on a yearly basis.

Do-It-Yourself Program 38-3

```
5   CLS
10   PRINT "THIS" TAB(POS(0)+4) "IS";
20   PRINT TAB(POS(0)+4)"EVENLY" TAB(POS(0)+4) "SPACED"
```

SAMPLE PROGRAMS

Sample Program #1

Type this program and save it on cassette, but don't open it (or run it) until Christmas!

```
5  CLS
10  PRINT @ 64, STRING$ (32,"*")
15  PRINT @ 352, STRING$ (32,"*")
20  PRINT @ 199, "JOY TO THE WORLD"
25  FOR X=1 TO 1000: NEXT X
30  CLS
35  PRINT @ 64, "JOY TO THE WORLD"
40  PRINT @ 96, "THE LORD IS COME"
45  PRINT @ 128, "LET EARTH RECEIVE HER KING"
50  PRINT @ 160, "LET EVERY HEART"
55  PRINT @ 192, "PREPARE HIM ROOM"
60  PRINT @ 224, "AND HEAVEN AND NATURE SING"
65  PRINT @ 256, "AND HEAVEN AND NATURE SING"
70  PRINT @ 288, "AND HEAVEN AND HEAVEN AND NATURE SING"
100  A$="T4;O3,L2;C;L4;O2;B;L8;A;L4,;G;L4;F;L2;E;D;"
105  B$="L2.;C;P32;L4;G;L2;A;L4;P32;A;L2.;B;P32;L4;B;O3;
     L1.;C"
110  C$="L4;C;C;O2;L4;B;A;G;L4.;G;L8;F;L4;E;O3;C
115  D$="O3;L4;C;O2;B;A;G;P32;L4.;G;L8;F;L4;E;P32;E;
     P32;E;P32;E;P32;E;P32;E;P32;L8;E;F
120  E$="L2.;G;L8;F;E;L4;D;P32;D;P32;D;P32;L8;D;E;
     L2.;F;L8;E;D"
125  F$="O2;L4;C;O3;L2;C;O2;L4;A;L4.;G;L8;F;L4;E;F;L2;
     E;D;L1;C"
130  X$ = "XA$;XB$;XC$;XD$;XE$;XF$;"
135  PLAY X$
200  PMODE 3,1
205  PCLS 4
210  SCREEN 1,0
215  COLOR 1,4
220  LINE (90,96)-(118,26), PSET
230  LINE (90,96)-(146,96), PSET
235  DRAW "BM112,96;D15;R10;U15"
240  LINE (0,112)-(255,96), PSET
245  PAINT (238,85),1,1
250  X = RND(255)
255  Y = RND(114)
260  A = RND(4)
265  PSET (X,Y,A): GOTO 250
```

Sample Program #2

```
1   '*** BACK TO BACH ***
2   '
5   CLS
10   PRINT @ 96, STRING$(32,"*")
20   PRINT @ 320, STRING$(32,"*")
25   PRINT @ 201, "BACK TO BACH"
40   FOR X = 1 TO 1000: NEXT X
55   A$ = "T6;O2;L2;G;L4;C;D;E;F;L2;G;C;P16;C"
60   B$ = "L2;A;L4;F;G;A;B;O3;L2;C;O2;C;P16;C;F;L4;G; F;E;D"
65   C$ = "L2;E;L4;F;E;D;C;L2;O1;B;O2;L4;C;D;E;C"
70   D$ = "L2;E;L1;D;L2;G;L4;C;D;E;F;L2;G;C;P16;C"
75   E$ = "L2;A;L4;F;G;A;B;O3;L2;C;O2;C;P16;C;F;L4;G; F;E;D"
80   F$ = "L2;E;L4;F;E;D;C;D;E;L2;F;O1;B;L1;O2;C"
85   X$ = "XA$;XB$;XC$;XD$;XE$;XF$;"
90   PLAY X$
```

Sample Program #3

```
1   ' *** MEXICAN HAT DANCE ***
2   '
5   CLS
10   PRINT @ 96, STRING$(32,"*")
20   PRINT @ 320,STRING$(32,"*")
30   PRINT @ 199, "MEXICAN HAT DANCE"
40   FOR X=1 TO 500: NEXT X
125   REM START TUNE
130   O$="V15;T3;O2;"
135   P$="L8CFP8CFP8CFP4P8"
140   Q$="CFGFEP8FGP4P8"
145   X$="XO$;XP$;XQ$;"
150   PLAY X$
155   R$="CEP8CEP8CEP4P8"
160   S$="CEFEDP8EFP4P8"
165   Y$="XO$;XR$;XS$;"
170   PLAY Y$
180   REM 2ND TIME
185   O$="V25;T3;O1"
190   PLAY X$
195   O$="T3;O4"
197   S$="CEFEDP8EFO4CO3AF"
200   PLAY Y$
210   A$="O3CO2BO3CO2AA-AFEFCP4"
220   B$="CO1BO2CDEFGAB-O3C3G"
225   O$="V15;T4;"
230   Z$="XO$;XA$;XB$;"
235   PLAY Q$
240   C$="0EB-AB-GF+FEG-ECEG"
245   D$="O4L16CP16CP16CP16L8DCO3B-AGFP4"
250   E$="XO$;XC$;XD$;"
255   PLAY E$
```

```
260  F$="O2L16GP16GP16GP16DP16DP16DP16EP16FP16L8EL16GP
     16O1GP1GL8G
265  G$="V15O2L16GP16GP16GP16DP16DP16DP1GEP1GFP1GL8
     ECO1GC"
270  H$="XF$;XG$;"
280  PLAY H$
285  I$=XF$;"
290  PLAY I$
295  J$="O2L16GP16GP16GP16AP16GP16GP16AP16BP1GO3L4CP8"
300  PLAY "XJ$;"
310  K$="O4L1DL4DEDEL8DEDEL16DEDEDEDEL32DEDEDEDEDEDED
     EDEL64DEDEDE DEDEDEDEDEDEDEDEDEL32DD-CO3BB-AA-GF
     +FEE-DDD-L4DD-"
320  PLAY "XK$;"
330  M$="T5L8DO2BB-BGF+GL4DP8"
340  N$="L8DC+DEF+GABO3CO2L4AP8"
350  AA$="O3L8CO2BO2CO2AG+AF+FF+L4DP8"
370  BB$="O3L8DDDEDCO2BAO3DEDCO2BA"
380  CC$="O2DEDCO1BAO4DEDDEDDEDDEF+GDO3BGT4DO2BGT3DO1
     T2BL4P2V30L1G"
400  PLAY "XM$;XN$;XAA$;XBB$;XCC$;"
500  PMODE 4,1
505  FOR Y=1 TO 5
510  SCREEN 1,0
520  PCLS
550  CIRCLE (128,96),50,1,.2,.85,.67
560  CIRCLE (128,96),25,1,2,.5,1
570  LINE (105,96)-(151,96),PSET
600  PMODE 4,1
610  SCREEN 1,0
620  PCLS
630  CIRCLE (128,75),50,1,.2,.85,.67
660  CIRCLE (128,75),1,2,.5,1
670  LINE (105,75)-(151,75), PSET
675  NEXT Y
680  IF Y>5 THEN 690
685  GOTO 500
690  CLS
700  PRINT @ 227, "NOT THAT'S A HOT TAMALE"
710  FOR X=1 TO 600: NEXT X
720  GOTO 5
```

Sample Program #4

```
1   ' *** BUFFALO GALS ***
2   '
5   CLS
10  PRINT @ 64, STRING$(32,"*")
15  PRINT @ 384, STRING$(32,"*")
20  PRINT @ 201, "BUFFALO GALS"
25  FOR X=1 TO 1000: NEXT X: CLS
30  PRINT @ 32, "AS I WAS WALKING DOWN THE STREET"
35  PRINT @ 64, "DOWN THE STREET, DOWN THE STREET"
40  PRINT @ 96, "A PRETTY GAL I HAPPENED"
45  PRINT @ 133, "TO MEET"
50  PRINT @ 160, "JUST AS LOVELY AS"
55  PRINT @ 197, "THE MORNING DEW"
60  PRINT @ 224, "BUFFALO GALS WON'T YOU"
65  PRINT @ 261, "COME OUT TONIGHT"
70  PRINT @ 288, "COME OUT TONIGHT,"
75  PRINT @ 320, "COME OUT TONIGHT,"
80  PRINT @ 352, "BUFFALO GALS WON'T YOU"
85  PRINT @ 391, "COME OUT TONIGHT"
90  PRINT @ 416, "AND DANCE IN THE"
95  PRINT @ 453, "LIGHT OF THE MOON."
100  A$="T4;C;E;P32;E;F;P32;F;A;G;L2;E;"
105  B$="L4;G;F;L2;D;L4;A;G;E;C;"
110  C$="L4;E;P32;E;F;P32;F;L8;A;P32;A;L4;G;E;O3;L8;C;
     P32;C;"
115  D$="O2;B;P32;B;G;P32;G;L4;F;O1;B;O2;L1;C;P16;"
120  E$="L8;G;P32;G;L4;F;L2;D;L4;A;L8;G;P32;G;L2;E"
130  G$="L8;C;P64;C;P64;L4;C;E;L8;G;P32;G;L4;A;L8;G;
     P32;G;L4;E;O3;C;"
135  H$="O2;B;L8;G;P32;G;F;P32;F;L4;D;L2.;C;"
140  X$="XA$;XB$;XC$;XD$;XE$;XF$;XG$;XH$;"
145  PLAY X$
150  CLS
155  PRINT @ 230, "THAT'S ALL FOLKS"
```

Sample Program #5

```
1   '*** IN-OUT ***
2   '
5   PMODE 3,1
10  PCLS3
15  SCREEN 1,0
20  FOR I=3 TO 7
25  FOR J=2 TO 6
30  FOR S=0 TO 3
35  FOR R=0 TO 3
40  COLOR R,S
45  A=0: B=255: C=0: D=191
50  LINE (A,C)-(B,D), PSET,B
55  A=A+J:B=B-J:C=C+I:D=D-I
60  IF A<255 AND C<191 THEN 50
65  NEXT R
70  NEXT S
75  NEXT J
76  NEXT I
80  GOTO 80
```

Sample Program #6

```
1   '*** DRAWING TRIANGLES ***
10  CLS: CLEAR
75  PRINT @ 96, STRING$(32,"*")
80  PRINT @ 288, STRING$(32,"*")
100  PRINT @ 160, "THIS PROGRAM DRAWS THE TRIANGLE YOU
     SPECIFY AND THEN CALCULATES ITS AREA"
110  FOR X=1 TO 2200: NEXT: CLS
120  CLS:PRINT "FOR 3 SIDES TYPE, SSS (0-100)"
125  PRINT "FOR 2 SIDES (1-100) AND 1 ANGLE (0-90) TYPE,
     SAS"
130  PRINT "FOR 1 SIDE (0-60) AND 2 ANGLES (0-90) TYPE,
     ASA"
140  INPUT A$: IF A$="SAS" GOTO 300
150  IF A$="ASA" GOTO 400
200  'SSS
210  PRINT "ENTER 3 SIDES, (LONGEST SIDE FIRST)"
220  INPUT L1,L2,L3
225  IF L2>L1 OR L3>L1 THEN PRINT "*** LONGEST FIRST
     PLEASE ...": PRINT: GOTO 210
230  S=(L1+L2+L3)/2
235  IF S<L1 THEN PRINT "***NOT A TRIANGLE***": PRINT:
     GOTO 210
240  Y3=2*SQR(S*(S-L2)*(S-L1)*(S-L3))/L1
250  A=Y3/L2: A=ATN(A/SQR(-A*A+1))
260  X3=COS(A)*L2
270  AR=(L1*Y3)/2
280  GOTO 490
300  'SAS
310  PRINT "ENTER 2 SIDES AND 1 ANGLE: AB,AC,THETA
     (LARGEST SIDE FIRST)"
320  INPUT L1,L2,T
325  T=(T*3.14159)/180
330  Y3=L2*SIN(T)
340  X3=COS(T)*L2
350  AR=(L1*Y3)/2: GOTO 490
400  'ASA
410  PRINT "ENTER 2 ANGLES AND 1 SIDE: THETA1,THETA2,AB"
420  INPUT T1,T2,L2
425  T1=(T1*3.14159)/180: T2=(T2*3.14159)/180
430  Y3=L2*SIN(T1)
440  B1=COS(T1)*L2
450  B2=Y3/TAN(T2)
460  L1=B1+B2: XE=B1: IF LX>L1 THEN X=L1: L1=L2: L2=X
470  AR=(L2*Y3)/2
490  CLS: PMODE4,1: PCLS:SCREEN 1,1
500  F=1
510  VC=(3.14159 * (L1*F-X3*F)*(Y3*F)↑1)/3
520  VS=(3.14159 * (X3*F)*(Y3*F)↑2)/3: VT=VC+VS
530  S1=Y3/X3: S2=Y3/(X3-L1)
532  IF INT(X3) = 0 THEN 1100
533  IF INT(X3) = INT(L1) THEN 1000
535  IF X3>L1 THEN 1100
```

```
537  IF X3=L2 THEN 1000
540  FOR Y=20 TO L1*2+20 STEP 2:PSET(Y,Y3+5,5): NEXT
550  FOR X=0 TO X3
551  PSET(X*2+20,S1*(X3-X)+5,5): NEXT
560  FOR X=X3 TO L1: PSET(X*2+20,Y3+(S2*(L1-X)+5),5):
     NEXT
580  FOR X=1 TO 600: NEXT X
610  PRINT @ 130, "AREA=";AR;" SQ. UNITS";
630  PRINT @352, "*";: INPUT "TO RUN AGAIN, PRESS <1>
     <ENTER>"; B6: IF B6=1 THEN 120
640  STOP: GOTO 10
1000  FOR Y=5 TO Y3+5: PSET(X3*2+20,Y,4): NEXT: GOTO 540
1100  FOR Y=5 TO Y3+5: PSET(20,Y,5): NEXT: GOTO 540
1200  FOR X=L1 TO X3: PSET (X*2+20,Y3+(S2*(L1-X)+5),5):
      NEXT: GOTO 540
```

Sample Program #7

```
1  ' *** PROJECTION STUDIES ***
2  '
5  PMODE 4,1
10  PCLS
15  SCREEN 1,0
20  DRAW "BM50,50R60D10NL20D20L20NU20L20NU20L20U20NR
    20U10"  'TOP VIEW
25  DRAW "BM50,100R20ND20R20ND20R20D20NL20D10L60U10NR
    20U10"  'FRONT VIEW
30  DRAW "BM150,100R30D30L30U10NE20U20"  'SIDE VIEW
35  'OBLIQUE VIEW - LINES 40-60
40  DRAW "BM150,50U5E15R10BF20BD30NR5L20H25U10"
45  DRAW "BM150,50U5F8U15R15H8F8L15F8NR15D15F8ND10E
    15NR10H8"
50  LINE (175,30)-(200,55), PSET
55  LINE -(200,80), PSET
60  LINE (167,60)-(183,46), PSET
65  GOTO 65
```

Sample Program #8

```
1   ' *** UNFOLDING BOX ***
2   '
5   PCLEAR 8
10   PMODE 3,1
15   PCLS
20   COLOR 6,5
25   DRAW "BM100,100U30NR30E15R30NG15D30G16NU30L30"
30   PAINT (105,95),8,6
35   PAINT (135,80),8,6
40   PAINT (110,65),8,6
45   SCREEN 1,1
50   FOR X=1 TO 600: NEXT X
110   PMODE 3,5
112   PCLS
115   COLOR 6,5
120   DRAW "BM100,100U30NR30E20R30G20D30NL30F20L30H20"
125   LINE (100,100)-(70,95), PSET
130   LINE - (70,65), PSET
135   LINE - (100,70), PSET
140   LINE (70,95)-(40,65), PSET, B
145   LINE (130,100)-(160,95), PSET
150   LINE - (160,65), PSET
155   LINE - (130,70), PSET
160   PAINT (95,95),8,6
165   PAINT (105,95),8,6
170   PAINT (135,85),8,6
175   PAINT (45,85),8,6
180   PAINT (115,65),8,6
185   PAINT (125,114),8,6
190   SCREEN 1,1
195   FOR X=1 TO 600: NEXT X
200   GOTO 10
```

Sample Program #9

```
1   ' *** SINE WAVE ***
2   '
5   PMODE 4,1
10   PCLS
15   SCREEN 1,1
20   LINE (0,96)-(255,96), PSET
25   PI=3.14159
30   A1=-4*PI
35   A2=4*PI
40   N=180
45   R=50
50   X=(A2-A1)/N
55   F=255/(A2-A1)
60   FOR I=A1 TO A2 STEP X
65   X=I*F
70   Y=R*SIN(I)
75   PSET ((X+140),(96+Y),1)
80   NEXT I
90   GOTO 90
```

Sample Program #10

```
1   ' *** SIN/COS ***
2   '
10  PMODE 4,1
20  PCLS
30  SCREEN 1,0
40  LINE (127,5)-(127,185), PSET
50  LINE (7,95)-(247,95), PSET
60  FOR XSCALE=7 TO 247 STEP 20
70  PRESET (XSCALE,95)
80  NEXT XSCALE
90  FOR YSCALE=5 TO 185 STEP 10
100  PRESET (127,YSCALE)
110  NEXT YSCALE
130  FOR X=-180 TO 180 STEP 1.5
140  AX=X/57.29578
145  XP=X/1.5+127
150  F1=-(SIN(AX)*90)+95
160  F2=-(COS(AX)*90)+95
170  PSET (XP,F1,1): PSET(XP,F2,1)
180  NEXT X
190  GOTO 190
```

Sample Program #11

```
1    ' *** RANDOM GRAPHICS ***
2   '
10  PMODE 3,1
15  PCLS
20  SCREEN 1,1
25  F=RND(4): B=RND(8): IF B=F OR (B-4=F) THEN 25
30  COLOR F,B:PCLS B: FOR L=0 TO 5
35  LINE -(RND(255), RND(191)), PSET
40  CIRCLE (RND(255), RND (191)), RND(100)
50  NEXT: FOR P0=0 TO 10
55  PAINT (RND(255), RND(191)), RND(4),F
60  NEXT: FOR H=1 TO 7
65  FOR T=0 TO 600: NEXT T: GOTO 10
```

Sample Program #12

```
1  ' *** NAVAHO BLANKET ***
2  '
5  PMODE 3,1
10  PCLS 4
15  SCREEN 1,0
20  COLOR 1,0
25  FOR X=0 TO 255 STEP 10
30  OY=Y
35  Y=30-OY
40  LINE (X,100-Y)-(X+10,100-OY), PSET
45  LINE (X,120+Y)-(X+10,120+OY), PSET
50  NEXT
60  FOR C=2 TO 4
65  PAINT (O,110),C,1
70  NEXT
80  GOTO 60
```

Sample Program #13

```
1  '*** PAINTED LACE ***
2  '
5  PMODE 3,1
10  PCLS
20  SCREEN 1,1
30  DRAW "BM50,180U60BU20U60R60BR20R60D60BD20D60L60
    BL20L60
40  DRAW "BM50,180U60R40BR20R80D20BL20L60BL20L20D20
    R20BR60R20U20
50  DRAW "BM50,180R60U80BU20U40L40BD20D20BD60D20R20
    U60BU20U20L20
60  DRAW "BM50,180U60BU40BR20R60BR20R20U20L20D60BD20
    D20R20
70  DRAW "BM50,180BR80U40BU20U80
80  DRAW "BM50,180BU80R80BR20R40
90  PAINT (85,128),6,8
95  PAINT (95,78),6,8
97  PAINT (155,95),6,8
98  PAINT (135,145),6,8
99  PAINT (128,185),7,8
100  PAINT (75,150),7,8
101  PAINT (160,150),7,8
102  PAINT (75,75),7,8
103  PAINT (160,75),7,8
104  PAINT (120,110),7,8
110  FOR X=1 TO 600: NEXT X
200  GOTO 5
```

Sample Program #14

```
1   '*** DRAWING BOARD ***
2   '
3   CLS
5   PRINT@ 128, STRING$(32,"*"): PRINT@ 288,
    STRING$(32,"*")
10   PRINT@ 200, "DRAWING BOARD"
15   FOR X=1 TO 600: NEXT X
20   CLS
25   PRINT@96, "PRESS <↑> FOR UP, <DOWN ARROW> FOR DOWN,
     <LEFT ARROW> FOR LEFT, <RIGHT ARROW> FOR RIGHT,
     <A> FOR SOUTHWEST, <S> FOR SOUTHEAST,
     <W> FOR NORTHEAST, <Q> FOR NORTHWEST"
30   PRINT@ 288, "PRESS <1> FOR INVISIBLE LINE,
     <2>, <3> OR <4> FOR DIFFERENT COLORED VISIBLE LINE,
     PRESS </> TO CHANGE COLOR-SET"
35   PRINT@ 488, "PRESS <SPACEBAR> TO PAUSE"
40   FOR X=1 TO 4800: NEXT X
45   CC=4: TG=0
50   PMODE 3,1
55   PCLS
60   SCREEN 1,TG
70   X=128: Y=96: XI=0: YI=0
80   U$="↑": D$=CHR$(10): W$=CHR$(8): E$=CHR$(9)
90   NW$="Q": NE$="W": SW$="A": SE$="S"
100   C1$="1": C2$="2": C3$="3": C4$="4"
110   A$=INKEY$
120   IF A$=U$ THEN YI=-1: XI=0: GOTO 240
130   IF A$=D$ THEN YI=1:XI=0: GOTO 240
140   IF A$=W$ THEN XI=-1:YI=0: GOTO 240
150   IF A$=E$ THEN XI=1:YI=0: GOTO 240
160   IF A$=NE$ THEN XI=1:YI=-1: GOTO 240
170   IF A$=NW$ THEN XI=-1:YI=-1: GOTO 240
180   IF A$=SE$ THEN XI=1:YI=1: GOTO 240
190   IF A$=SW$ THEN XI=-1:YI=1: GOTO 240
200   IF C1$<A$ AND A$<=C4$ THEN CC=ASC(A$)-48: GOTO 240
210   IF A$="/" THEN TG=(NOT TG AND 1) OR (TG AND NOT 1):
      GOTO 240
220   SCREEN 1, TG
230   IF A$=" " THEN XI=0: YI=0
240   X=X+XI: Y=Y+YI: IF X<0 THEN X=0
250   IF X>255 THEN X=255
260   IF Y<0 THEN Y=0
270   IF Y>191 THEN Y=191
280   PSET (X,Y,CC)
290   GOTO 110
```

Sample Program #15

```
1 '*** INTERACTING LINES ***
2 '
5  CLS
20  C=C+1
25  IF C>8 THEN C=5
30  COLOR C,1
50  PRINT "TYPE X0,Y0";
60  INPUT X0,Y0
70  PRINT "TYPE X1,Y1";
80  INPUT X1,Y1
90  PMODE 3,1
95  PCLS
100  SCREEN 1,1
110  LINE (X0,Y0)-(X1,Y1), PSET
115  FOR X=1 TO 2000: NEXT X
120  GOTO 20
```

Sample Program #16

```
1  '*** RANDOM LINES ***
2  '
20  PMODE 4,1
25  PCLS
30  SCREEN 1,1
35  X=RND(255): Y=RND(191)
40  LINE -(X,Y), PSET
45  FOR X=1 TO 200: NEXT X
50  GOTO 35
```

Sample Program #17

```
1  '*** 8-LEAF CLOVER ***
2  '
5  PCLEAR 8
10  PMODE 4,1
15  PCLS
20  SCREEN 1,0
25  PI=3.14159
30  A1=0:A2=2*PI
35  N=360:A=50
40  X=(A2-A1)/N
45  FOR I=A1 TO A2 STEP X
50  R=A*COS(4*I)
55  X=R*SIN(I)
60  Y=R*COS(I)
65  PSET (128+X,96+Y,5)
70  NEXT I
75  GOTO 75
```

Sample Program #18

```
1   '*** TIMEBOMB ***
2   '
10  PMODE 4,1
15  PCLS
20  SCREEN 1,1
25  CIRCLE (128,96),80
30  CIRCLE (128,96),90
35  PAINT (0,0),5
40  FOR T=30 TO -30 STEP -1
45  A=(2*3,1415)*T/60
50  LINE (128,96)-(75*SIN(A)+128,75*COS(A)+96), PSET
55  SOUND Q*2+1,20/(Q+1)+1
60  LINE (128,96)-(75*SIN(A)+128,75*COS(A)+96), PRESET
65  Q=60-2*T: FOR Y=Q TO 0 STEP -1:NEXT
70  NEXT
75  CLS
80  PCLS
85  PRINT@237, "BOOM!"
90  SOUND 1,30
95  PMODE 4,1
100  SCREEN 1,1
105  FOR I=2 TO 200 STEP 2
110  CIRCLE (128,96), I
115  NEXT I
120  SCREEN 1,1
125  FOR X=2 TO 200 STEP 2
130  CIRCLE (128,96), I,3,.5
135  NEXT X
140  FOR I=2 TO 200, STEP 2
145  CIRCLE (128,96),I,3,.5
150  NEXT I
155  GOTO 155
```

Sample Program #19

```
1   '*** ROTATING FAN ***
2   '
5   PCLEAR 8
50  GOTO 600
60  LINE ((255-X),(191-Y))-(X,Y), PSET
61  J=J+1: IF J>A THEN J=0: A=RND(50)
63  RETURN
600  REM ROTATING FAN
601  FOR I=1 TO 5 STEP 4
602  PMODE 3,I
603  PCLS
604  SCREEN 1,0
605  A=25: X=0: Y=0: J=0
610  FOR X=0 TO 254
612  COLOR X/32+1,5
```

```
615   GOSUB 60: NEXT X
620   FOR Y=0 TO 190
623   COLOR Y/24+1,5
625   GOSUB 60: NEXT Y
630   FOR X=255 TO 1 STEP -1
640   FOR Y=191 TO 1 STEP -1
643   COLOR 6/24+1,5
645   GOSUB 60: NEXT Y
650   NEXT I
660   FOR I=1 TO 5 STEP 4
670   PMODE 3,I
680   SCREEN 1,0
690   FOR T=1 TO 30: NEXT T
700   NEXT I
710   GOTO 660
```

Sample Program #20

```
1    '*** WALKING TRIANGLES ***
10   FOR A=90 TO 0 STEP -4
15   S1=A*9: S2=191
20   A3=A/57.29578
30   X1=0: Y1=191
40   X2=S1+X1: Y2=Y1
50   X3=X1+S2*COS(A3):Y3=Y1-S2*SIN(A3)
55   GOSUB 1000
90   NEXT A
99   GOTO 99
1000   PMODE 4,1
1005   PCLS
1010   SCREEN 1,0
1020   LINE (S1,Y1)-(X2,Y2), PSET
1030   LINE -(X3,Y3), PSET
1040   LINE -(X1,Y1), PSET
1060   RETURN
```

Sample Program #21

```
1    '*** COUNTING ***
2    '
10   CLS
20   CLEAR 1000
30   PRINT "WHERE DO YOU WANT TO START COUNTING?"
35   INPUT A$
40   P=LEN(A$)
50   PRINT: PRINT A$
60   C=VAL(MID$(A$,P,1)) + 1
70   MS$=A$: MR$=RIGHT$(STR$(C),1): PS=P: GOSUB 200:
     A$=MS$
80   IF C<10 THEN 40
90   P=P-1
100   IF P=0 THEN IF LEN(A$)=255 THEN PRINT "OVERFLOW":
      END: ELSE A$="1"+A$: GOTO 40
110   GOTO 60
200   LS=LEN(MS$)
210   IF LS<>LEN(MR$)+LS-1 OR PS<1 THEN STOP
220   MS$=LEFT$(MS$,PS-1)+MR$+RIGHT$(MS$,LS-PS)
230   RETURN
```

Sample programs 22-30 highlight BASIC version 2.

Sample Program #22

```
10   REM INTRO
20   ON BRK GOTO 400
30   PALETTE CMP: WIDTH 32
40   PRINT "THE COLOR COMPUTER 3 CAN DO ALL"
50   PRINT "THE THINGS THAT THE OTHER"
60   PRINT "MEMBERS OF THE COLOR COMPUTER"
70   PRINT "FAMILY CAN, PLUS A LOT MORE."
80   PRINT "YOU CAN CHOOSE FROM 32, 40, AND"
90   PRINT "80 COLUMN TEXT SCREENS."
100  PRINT: PRINT "THIS IS 16 LINES, 32 COLUMNS."
110  FOR T=1 TO 5000: NEXT
120  WIDTH 40:ATTR 2, 0: LOCATE 5, 11
130  PRINT "This is 24 lines, 40 columns."
140  LOCATE 0, 23
150  FOR T=1 TO 2000: NEXT
160  WIDTH 80: LOCATE 25, 11
170  PRINT "This is 24 lines, 80 columns."
180  LOCATE 0, 23
190  FOR T=1 TO 2000: NEXT
200  WIDTH 40
210  PRINT TAB(14);"NEW FEATURES": PRINT
220  PRINT "The Color Computer 3 offers:": PRINT
230  PRINT "  ADVANCED 40 AND 80 COLUMN TEXT"
240  PRINT "     -UPPER and lower case letters"
250  PRINT "     -Special characters:"
260  PRINT "      ";
270  FOR Z=128 TO 159: PRINT CHR$(Z);: NEXT: PRINT
280  PRINT "     -";: ATTR 2, 0, B: PRINT "BLINKING":
     ATTR 2, 0
290  PRINT"     -";: ATTR 2, 0, U: PRINT "UNDERLINING";:
     ATTR 2, 0: PRINT
300  PRINT "     -";: ATTR 2, 0, B, U: PRINT "BOTH";:
     ATTR 2, 0: PRINT
310  A$= "COLOR": PRINT"     -";: FOR Z=1 TO 5: ATTR Z-1, Z+2:
     PRINT MID$(A$,Z,1);: NEXT: ATTR 2, 0: PRINT
320  PRINT: PRINT "  POWERFUL NEW HIGH-RESOLUTION GRAPHICS"
330  PRINT "     -640V x 192H with 4 colors"
340  PRINT "     -320V x 192H with 16 colors"
350  PRINT "     -Shows 16 colors, with 64 to"
360  PRINT "      choose from."
370  PRINT "     -Programs can print text on"
380  PRINT "      the graphics screen."
390  LOCATE 0, 23: FOR T=1 TO 15000: NEXT
400  PALETTE CMP: ATTR 2, 0: WIDTH 40
410  END
```

Sample Program #23

```
10   REM HCOLORS
20   ON BRK GOTO 430
30   WIDTH 40: PALETTE CMP: ATTR 2, 0: LOCATE 0, 3
40   PRINT "SEE ALL 64 COLORS, 8 AT A TIME.": PRINT: PRINT
50   ATTR 2,0, B
60   PRINT "PRESS SPACE BAR TO BEGIN.": PRINT
70   PRINT "PRESS BREAK KEY TO STOP."
80   ATTR 1, 0: LOCATE 0, 23
90   A$=INKEY$: IF A$="" THEN 90
100  IF ASC(A$)=3 THEN 430
110  REM HCOLORS
120  REM SHOW ALL 64 COLORS, 8 AT A TIME
130  HSCREEN 2
140  FOR Z=0 TO 7
150  PALETTE CMP
160  HCLS
170  FOR Y=0 TO 7
180  C=Y+Z*8
190  PALETTE Y, C
200  PALETTE Y+8, (C+32) AND 63
210  HCOLOR Y, 0
220  HLINE (0,Y*24)-(319,Y*24+23), PSET, BF
230  HCOLOR (Y+1) AND 7, 0
240  HPRINT (4,Y*3+1), C
250  HCOLOR Y+8, 0
260  HPRINT (33,Y*3+1), C
270  NEXT
280  HCOLOR 3, 0
290  HPRINT (8,7), "PRESS"
300  HCOLOR 4, 0
310  HPRINT (8,10), "SPACE"
320  HCOLOR 5, 0
330  HPRINT (9,13), "BAR"
340  HCOLOR 10, 0
350  HPRINT (27,7), "PRESS"
360  HCOLOR 11, 0
370  HPRINT (27,10), "SPACE"
380  HCOLOR 12, 0
390  HPRINT (28,13), "BAR"
400  A$=INKEY$: IF A$="" THEN 400
410  IF ASC(A$)=3 THEN 430
420  NEXT
430  HSCREEN 0: PALETTE CMP: ATTR 2, 0: CLS
440  END
```

Sample Program #24

```
10   REM PALETTE
20   ON BRK GOTO 200
30   GOSUB 60
40   FOR Y=0 TO 7: PALETTE Y, RND(64)-1:
     PALETTE Y+8, RND(64)-1: NEXT: GOTO 40
50   GOTO 200
60   WIDTH 40: PALETTE CMP
70   LOCATE 1, 0: ATTR 0, 4, U
80   PRINT "COLOR COMPUTER 3 PALETTE DEMONSTRATION";
90   ATTR 0, 4: LOCATE 8, 6: ATTR 0, 4, B
100  PRINT "PRESS SPACE BAR TO BEGIN";
110  ATTR 0, 4: LOCATE 9, 11
120  PRINT "PRESS BREAK KEY TO STOP";
130  ATTR 2, 0: LOCATE 0, 23
140  K$=INKEY$: IF K$="" THEN 140
150  IF ASC(K$)=3 THEN 200
160  HSCREEN 2
170  FOR C=0 TO 15: FOR V=37 TO 157 STEP 30:
     D=((V-37)/15+C) AND 15: HCIRCLE((C*20)+10,V), 9, 2:
     HPAINT((C*20)+10,V), D, 2: NEXT V, C
180  FOR Z=0 TO 7: HCOLOR Z, 0:
     HLINE(0,Z*3)-(319,2+Z*3), PSET, BF: HCOLOR Z+8, 0:
     HLINE(0,168+Z*3)-(319,170+Z*3), PSET, BF: NEXT
190  RETURN
200  PALETTE CMP: HSCREEN 0: ATTR 2, 0: CLS
210  PRINT "THAT WAS FUN !": PRINT
220  END
```

Sample Program #25

```
10   REM COLORBOX
20   ON BRK GOTO 200
30   GOTO 50
40   FOR Z=0 TO 15: PALETTE Z, RND(64)-1: NEXT: GOTO 40
50   WIDTH 40: PALETTE CMP: ATTR 2, 0
60   PRINT "COLORBOX": LOCATE 0, 5
70   ATTR 2, 0, B: PRINT "PRESS SPACE BAR TO BEGIN.": PRINT:
     PRINT
80   PRINT "PRESS BREAK KEY TO STOP.": ATTR 2, 0: LOCATE 0, 23
90   A$=INKEY$: IF A$="" THEN 90
100  IF ASC(A$)=3 THEN 200
110  HSCREEN 2
120  FOR Z=0 TO 95: C=Z AND 15
130  PALETTE C, RND(64)-1: HCOLOR C, 0
140  HLINE(Z,Z)-(319-Z,Z), PSET
150  HLINE-(319-Z,191-Z), PSET
160  HLINE-(Z,191-Z), PSET
170  HLINE-(Z,Z), PSET
180  NEXT
190  GOTO 40
200  HSCREEN 0: PALETTE CMP: ATTR 2, 0: CLS
210  END
```

Sample Program #26

```
10   REM HPUT DEMONSTRATION
20   ON BRK GOTO 400
30   WIDTH 40: PALETTE CMP: CLS
40   PRINT "HPUT DEMONSTRATION"
50   LOCATE 0, 5: ATTR 2, 0, B
60   PRINT "PRESS SPACE BAR TO BEGIN": PRINT: PRINT
70   PRINT "PRESS BREAK KEY TO STOP"
80   ATTR 2, 0: LOCATE 0, 23
90   A$=INKEY$: IF A$="" THEN 90
100  IF ASC(A$)=3 THEN 400
110  HBUFF 1, 799
120  HSCREEN 2
130  PALETTE CMP
140  HCLS 0
150  HCIRCLE(20,20), 10, 2
160  HCIRCLE(20,20), 5, 3
170  HCIRCLE(10,10), 5, 4
180  HCIRCLE(10,30), 5, 5
190  HCIRCLE(30,10), 5, 6
200  HCIRCLE(30,30), 5, 7
210  HPAINT(20,20), 3, 2
220  HCOLOR 1, 0
230  HLINE(0,0)-(39,39), PSET
240  HCOLOR 6, 0
250  HLINE(10,20)-(30,20), PSET
260  HCOLOR 7, 0
270  HLINE(20,10)-(20,30), PSET
280  HCOLOR 4, 0
290  HLINE(0,0)-(0,39), PSET
300  HLINE(39,0)-(39,39), PSET
310  HCOLOR 5, 0
320  HLINE(0,0)-(39,0), PSET
330  HLINE(0,39)-(39,39), PSET
340  HGET(0,0)-(39,39), 1
350  PALETTE RND(8)-1, RND(64)-1
360  GOSUB 420: HPUT(X,Y)-(X+39,Y+39), 1, PSET
370  GOSUB 420: HPUT(X,Y)-(X+39,Y+39), 1, AND
380  GOSUB 420: HPUT(X,Y)-(X+39,Y+39), 1, OR
390  A$=INKEY$: IF A$="" THEN 350
400  HSCREEN 0: PALETTE CMP: CLS
410  END
420  X=2*INT(RND(278)/2)
430  Y=RND(150)
440  RETURN
```

Sample Program #27

```
10    REM LOOPS
20    ON BRK GOTO 190
30    HSCREEN 2: PALETTE CMP: PALETTE 0, 0
40    MIDDLE=RND(35)+5
50    SIZE=30+RND(70)
60    HCLS 0
70    HCOLOR 1, 0: HPRINT(8,0), "PRESS BREAK KEY TO STOP"
80    HCOLOR 2, 0: HPRINT(8,23), "PRESS BREAK KEY TO STOP"
90    FOR Z=0 TO 6.3 STEP .045
100   XOFFS=COS(Z)*MIDDLE+159.5
110   YOFFS=SIN(Z)*MIDDLE+95.5
120   PALETTE RND(15), RND(63)
130   CO=RND(16)-1
140   HCIRCLE(XOFFS,YOFFS), SIZE, CO, .45, .8, .2
150   HCIRCLE(XOFFS,YOFFS), SIZE, CO, .45, .3, .7
160   NEXT
170   FOR T=1 TO 4000: NEXT
180   GOTO 40
190   PALETTE CMP: HSCREEN 0: CLS: END
```

Sample Program #28

```
10    REM STRINGS
20    ON BRK GOTO 250
30    WIDTH 40: PALETTE CMP
40    PRINT "AUTOMATIC STRING ART": LOCATE 0, 5
50    ATTR 2, 0, B: PRINT"PRESS SPACE BAR TO BEGIN.":
      PRINT: PRINT
60    PRINT "PRESS BREAK KEY TO STOP": ATTR 2, 0: LOCATE 0, 23
70    A$=INKEY$: IF A$="" THEN 70
80    IF ASC(A$)=3 THEN 250
90    HSCREEN 2: PALETTE CMP: PALETTE 0, 0
100   X1=RND(150): Y1=RND(90): X2=RND(150)+168:
      Y2=RND(90)+100: LC=0
110   HCOLOR RND(7), 0
120   HLINE(X1,Y1)-(X2,Y2), PSET
130   LC=LC+1: IFLC=200 THEN HCLS: GOTO 100
140   X1=X1-4: X=X1: GOSUB 190: X1=X
150   Y1=Y1-4: Y=Y1: GOSUB 220: Y1=Y
160   X2=X2+3: X=X2: GOSUB 190: X2=X
170   Y2=Y2+2: Y=Y2: GOSUB 220: Y2=Y
180   GOTO 110
190   IF X<0 THEN X=319+X: RETURN
200   IF X>319 THEN X=X-319
210   RETURN
220   IF Y<0 THEN Y=191+Y: RETURN
230   IF Y>191 THEN Y=Y-191
240   RETURN
250   HSCREEN 0: PALETTE CMP: CLS: END
```

Sample Program #29

```
10    REM RANDOM
20    ON BRK GOTO 240
30    WIDTH 40: PALETTE CMP
40    PRINT "RANDOM GRAPHICS": LOCATE 0, 5
50    ATTR 2, 0, B: PRINT "PRESS SPACE BAR TO START.": PRINT:
      PRINT
60    PRINT "PRESS BREAK KEY TO STOP.": ATTR 2, 0:
      LOCATE 0, 23
70    A$=INKEY$: IF A$="" THEN 70
80    IF ASC(A$)=3 THEN 240
90    HSCREEN 2: PALETTE 0, 0
100   FOR Y=1 TO 5
110   FOR Z=1 TO 6
120   HCOLOR RND(15), 0:
      HLINE(RND(318),RND(190))-(RND(318),RND(190)), PSET
130   CX=RND(200)+60: CY=RND(70)+60: CC=RND(7)
140   HCIRCLE(CX,CY), RND(58), CC
150   HPAINT(CX,CY), (CC+1) AND 7, CC
160   NEXT
170   HCOLOR RND(7), 0
180   HX=RND(200): H1=RND(80)+200
190   HY=RND(70): H2=RND(30)+90
200   HLINE(HX,HY)-(H1,H2), PSET, BF
210   NEXT
220   HCLS 0
230   GOTO 100
240   PALETTE CMP: HSCREEN 0: CLS
250   END
```

Sample Program #30

```
10    REM DOODLE
20    ON BRK GOTO 750
30    WIDTH 40: PALETTE CMP
40    PRINT "WELCOME TO COLOR DOODLE. WITH COLOR"
50    PRINT "DOODLE, YOU CAN DRAW PICTURES ON THE"
60    PRINT "SCREEN BY PRESSING KEYS. THE KEYS ARE:": PRINT:
      PRINT
70    PRINT "(ARROW KEY).......MOVE ONE DOT"
80    PRINT "(SHIFT)(ARROW-KEY)MOVE 10 DOTS"
90    PRINT "(CLEAR)...........PAINT AN OBJECT"
100   PRINT "(SHIFT)(CLEAR)....CLEAR SCREEN"
110   PRINT "(F1)..............RAISE AND LOWER PEN"
120   PRINT "(F2)..............CHANGE INK COLOR"
130   PRINT "(LETTERS&NUMBERS).TYPE TEXT ON SCREEN"
140   PRINT "(ALT)OR(@)........STORE A POINT"
150   PRINT "(CTRL)............DRAW TO STORED POINT"
160   PRINT "(ENTER)...........FLICKER ON/OFF"
170   PRINT: PRINT: ATTR 2, 0, B:
      PRINT "PRESS SPACE BAR TO BEGIN.": PRINT
180   PRINT "PRESS BREAK KEY TO STOP.": ATTR 2, 0
190   LOCATE 0, 23
200   A$=INKEY$: IF A$="" THEN 200
210   IF ASC(A$)=3 THEN 750
220   HSCREEN 2: PALETTE 0, 0: HBUFF 1, 799: HBUFF 2, 307:
      HGET(72,184)-(159,190), 2
230   XP=160: YP=92: CV=1: PF=0: FF=0: MX=160: MY=92
240   HCOLOR 2, 0: HLINE(0,182)-(319,182), PSET: HCOLOR 1, 0
```

```
250   HPRINT(0,23), "PEN-UP": HPRINT(21,23), "COLOR->"
260   HCOLOR CV, 0: HLINE(224,183)-(317,191), PSET, BF
270   IF XP<2 THEN XP=317: GOTO 340
280   IF XP>317 THEN XP=2: GOTO 340
290   IF YP<2 THEN YP=179: GOTO 340
300   IF YP>179 THEN YP=2
310   HCOLOR CV, 0
320   IF PF=1 THEN HLINE(FX,FY)-(XP,YP), PSET
330   FX=XP: FY=YP
340   HCOLOR 1, 0: HPUT(72,184)-(159,190), 2, PSET
350   XV$=RIGHT$(" "+STR$(XP-2),3):
      YV$=RIGHT$(" "+STR$(179-YP),3)
360   PS$="X="+XV$+" Y="+YV$: HPRINT(9,23), PS$
370   HGET(0,YP-2)-(319,YP+2), 1
380   HDRAW "BM"+STR$(INT(XP))+","+STR$(INT(YP))+";C"+
      STR$(IC)+";D2;U2;L2;R4;L2;U2"
390   IC=(IC+1) AND 7 OR 8
400   IF FF=0 THEN 420
410   FOR PN=0 TO 15: PALETTE PN, RND(64)-1: NEXT
420   A$=INKEY$: HPUT(0,YP-2)-(319,YP+2), 1: IF A$="" THEN 370
430   KV=ASC(A$)
440   IF KV=3 THEN 750
450   IF KV=94 THEN YP=YP-1: GOTO 270
460   IF KV=10 THEN YP=YP+1: GOTO 270
470   IF KV=8 THEN XP=XP-1: GOTO 270
480   IF KV=9 THEN XP=XP+1: GOTO 270
490   IF KV=95 THEN YP=YP-10: GOTO 270
500   IF KV=91 THEN YP=YP+10: GOTO 270
510   IF KV=21 THEN XP=XP-10: GOTO 270
520   IF KV=93 THEN XP=XP+10: GOTO 270
530   IF KV=64 THEN MX=XP: MY=YP: HCOLOR CV, 0: HSET(XP,YP):
      GOTO 370
540   IF KV=189 THEN HCOLOR CV,0: HLINE(MX,MY)-(XP,YP), PSET:
      GOTO 370
550   IF KV<>13 THEN 590
560   FF=(FF+1) AND 1
570   IF FF=0 THEN PALETTE CMP: PALETTE 0, 0
580   GOTO 370
590   IF KV=4 THEN CV=(CV+1) AND 15: HCOLOR CV, 0:
      HLINE(224,183)-(317,191), PSET, BF: GOTO 370
600   IF KV<>12 THEN 670
610   TC=HPOINT(XP,YP): TX=XP
620   IF XP<160 THEN SV=1 ELSE SV=-1
630   TX=TX+SV: NC=HPOINT(TX,YP)
640   IF NC<>TC THEN HCOLOR NC, 0: HLINE(0,182)-(319,182),
      PSET: HPAINT(XP,YP), CV, NC: HCOLOR CV, 0: GOTO 370
650   IF TX=2 OR TX=317 THEN 370
660   GOTO 630
670   IF KV=103 THEN PF=(PF+1) AND 1 ELSE 710
680   HCOLOR 2, 0: HLINE(0,184)-(63,190), PRESET, BF:
      HCOLOR 1, 0
690   IF PF=0 THEN HPRINT(0,23), "PEN-UP"
      ELSE HPRINT(0,23), "PEN-DOWN": FX=XP: FY=YP: GOTO 310
700   GOTO 370
710   IF KV=92 THEN HCOLOR CV, 0:
      HLINE(0,0)-(319,181), PSET, BF: GOTO 370
720   IF KV<32 OR KV>90 THEN 370
730   TX=INT(XP/8): TY=INT(YP/8): HCOLOR CV, 0
740   HPRINT(TX,TY), A$: XP=XP+8: GOTO 270
750   PALETTE CMP: HSCREEN 0: ATTR 2, 0: CLS
760   END
```

Inventory Shopping List

```
5   CLEAR 2000: DIM S$(100)
10   REM INVENTORY/SHOPPING LIST
20   CLS
30   PRINT@ 71, "DO YOU WANT TO--"
40   PRINT @ 134, "(1) INPUT ITEMS"
50   PRINT @ 166, "(2) REPLACE ITEMS"
60   PRINT @ 198, "(3) ADD TO THE LIST"
70   PRINT @ 230, "(4) DELETE ITEMS"
80   PRINT @ 262, "(5) PRINT ALL ITEMS"
90   PRINT @ 294, "(6) SAVE ITEMS ON TAPE"
100   PRINT @ 326, "(7) LOAD ITEMS FROM TAPE"
110   PRINT @ 395, "(1-7)";
120   INPUT M
130   IF M<0 OR M>7 THEN 10
140   ON M GOSUB 1000, 2000, 1020, 3000, 4000, 5000, 6000
150   GOTO 10
900   REM
1000   REM      INPUT/ADD ITEMS
1010   Y=1
1020   CLS: PRINT@ 8, "INPUT/ADD ITEMS"
1030   PRINT@ 34, "PRESS <ENTER> WHEN FINISHED"
1040   PRINT: PRINT "ITEM" Y;
1045   INPUT S$(Y)
1050   IF S$(Y) = " " THEN RETURN
1060   Y=Y+1
1070   GOTO 1040
1900   REM
2000   REM     REPLACE ITEMS
2005   N=0
2010   CLS: PRINT@ 9, "REPLACE ITEMS"
2020   PRINT@ 34, "PRESS <ENTER> WHEN FINISHED"
2030   PRINT: INPUT "ITEM NO. TO REPLACE"; N
2040   IF N=0 THEN RETURN
2050   INPUT "REPLACEMENT ITEM"; S$(N)
2060   GOTO 2000
2900   REM
3000   REM      DELETE ITEMS
3005   N=0
3010   CLS: PRINT@ 9, "DELETE ITEMS"
3020   PRINT@ 34, "PRESS <ENTER> WHEN FINISHED"
3030   PRINT: INPUT "ITEM TO DELETE"; N
3035   IF N>Y-1 THEN 3030
3040   IF N=0 THEN RETURN
3050   FOR X=N TO Y-2
3060   S$(X) = S$(X+1)
3070   NEXT X
3080   S$(X) = " "
3090   Y=Y-1
3100   GOTO 3000
3900   REM
4000   REM      PRINT ITEMS
4010   FOR X=1 TO Y-1 STEP 15
```

```
4020  FOR Z=X TO X+14
4030  PRINT Z; S$(Z)
4040  NEXT Z
4050  INPUT "PRESS >ENTER> TO CONTINUE"; C$
4060  NEXT X
4070  RETURN
4900  REM
5000  REM    SAVE ITEMS ON TAPE
5010  CLS: PRINT@ 135, "SAVE ITEMS ON TAPE"
5020  PRINT @ 234, "POSITION TAPE"
5030  PRINT @ 294, "PRESS PLAY AND RECORD"
5040  PRINT @ 388, "PRESS <ENTER> WHEN READY"
5050  INPUT R$
5060  OPEN "O", #-1, "LIST"
5070  FOR X=1 TO Y-1
5080  PRINT #-1, S$(X)
5090  NEXT X
5100  CLOSE #-1: RETURN
5900  REM
6000  REM      LOAD ITEMS FROM TAPE
6010  CLS: PRINT "LOAD ITEMS FROM TAPE"
6020  PRINT "POSITION TAPE"
6030  PRINT "PRESS PLAY"
6040  PRINT "PRESS <ENTER> WHEN READY"
6050  INPUT R$
6060  OPEN "I", #-1, "LIST"
6070  FOR X=1 TO Y-1
6080  INPUT #-1, S$(X)
6090  NEXT X
6100  CLOSE #-1: RETURN
```

Speed Reading

```
10   REM SPEED READING
20   CLS: PRINT@ 32, "HOW MANY WORDS PER MINUTE"
30   INPUT "DO YOU READ"; WPM
40   FOR X=1 TO 23: NEXT Y
60   READ A$: PRINT@ 256, A$
70   FOR Y=1 TO (360/WPM)*460: NEXT Y
80   REM Y LOOP SETS LINES/MINUTE
90   NEXT X: END
100  DATA "SCARLETT OHARA WAS NOT BEAUTIFUL"
110  DATA "BUT MEN SELDOM REALIZED IT WHEN"
120  DATA "CAUGHT BY HER OWN CHARM AS THE"
130  DATA "TARLETON TWINS HERE. IN HER FACE"
140  DATA "WERE TOO SHARPLY BLENDED"
150  DATA "THE DELICATE FEATURES OF HER"
160  DATA "MOTHER, A COAST ARISTOCRAT OF"
170  DATA "FRENCH DESCENT, AND THE HEAVY"
180  DATA "ONES OF HER FLORID IRISH FATHER"
190  DATA "BUT IT WAS AN ARRESTING FACE,"
200  DATA "POINTED OF CHIN, SQUARE OF JAW"
210  DATA "HER EYES WERE PALE GREEN"
220  DATA "WITHOUT A TOUCH OF HAZEL,"
```

```
230  DATA "STARRED WITH BRISTLY BLACK"
240  DATA "LASHES AND SLIGHTLY TILTED"
250  DATA "THE ENDS, ABOVE THEM, HER THICK"
260  DATA "BLACK BROWS SLANTED UPWARDS,"
270  DATA "CUTTING A STARTLING OBLIQUE LINE"
280  DATA "IN HER MAGNOLIA-WHITE SKIN -- THAT "
290  DATA "SKIN SO PRIZED BY SOUTHERN WOMEN"
300  DATA "AND SO CAREFULLY GUARDED WITH"
310  DATA "BONNETS, VEILS, AND MITTENS"
320  DATA "AGAINST HOT GEORGIA SUNS"
```

Memory Test

This program uses an array to test both yours and your computer's memory.

```
5  DIM A(7)
10  PRINT "MEMORIZE THESE NUMBERS"
15  PRINT "YOU HAVE 10 SECONDS"
20  FOR X=1 TO 7
30  A(X) = RND(100)
40  PRINT A(X)
50  NEXT X
60  FOR X=1 TO 460*10: NEXT X
70  CLS
80  FOR X=1 TO 7
90  PRINT "WHAT WAS NUMBER" X
100  INPUT R
110  IF A(X)=R THEN PRINT "CORRECT" ELSE PRINT "WRONG -
     IT WAS" A(X)
120  NEXT X
```

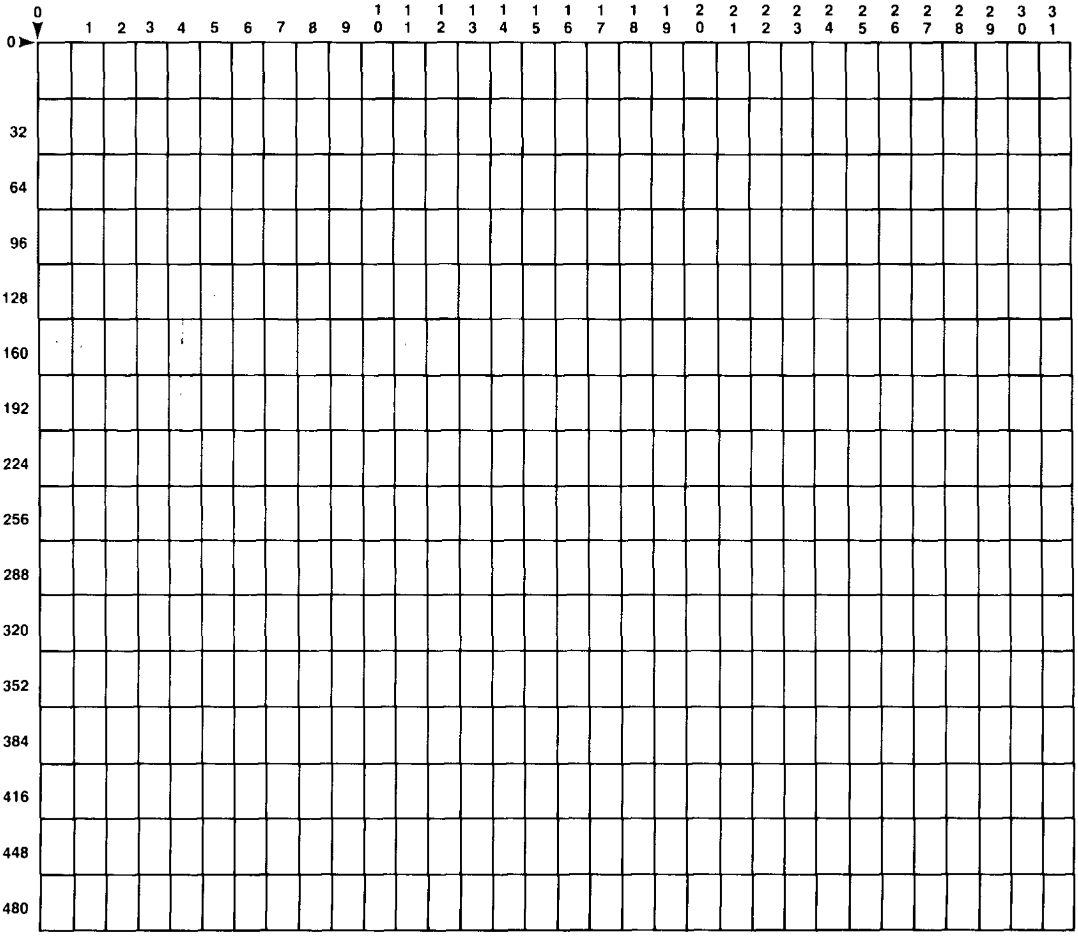
Sorting

```
1  CLS: CLEAR 1000: DIM T$(100), A$(100), S$(100),
   M$(100), Z(100)
2  PRINT "POSITION TAPE -- PRESS PLAY AND RECORD?
4  INPUT "PRESS <ENTER> WHEN READY"; R$
8  REM
9  REM OUTPUT TO TAPE
10  OPEN "O", #-1, "BOOKS"
15  CLS: PRINT "INPUT YOUR BOOKS -- TYPE <XX> WHEN
    FINISHED"
20  INPUT "TITLE"; T$
25  IF T$= "XX" THEN 50
26  INPUT "AUTHOR"; A$
28  INPUT "SUBJECT"; S$
30  PRINT #-1, T$, A$, S$
40  GOTO 15
50  CLOSE #-1
60  CLS: PRINT "REWIND THE RECORDER AND PRESS PLAY"
70  INPUT "PRESS <ENTER> WHEN READY"; R$
74  REM
76  REM INPUT FROM TAPE
78  B = 1
```

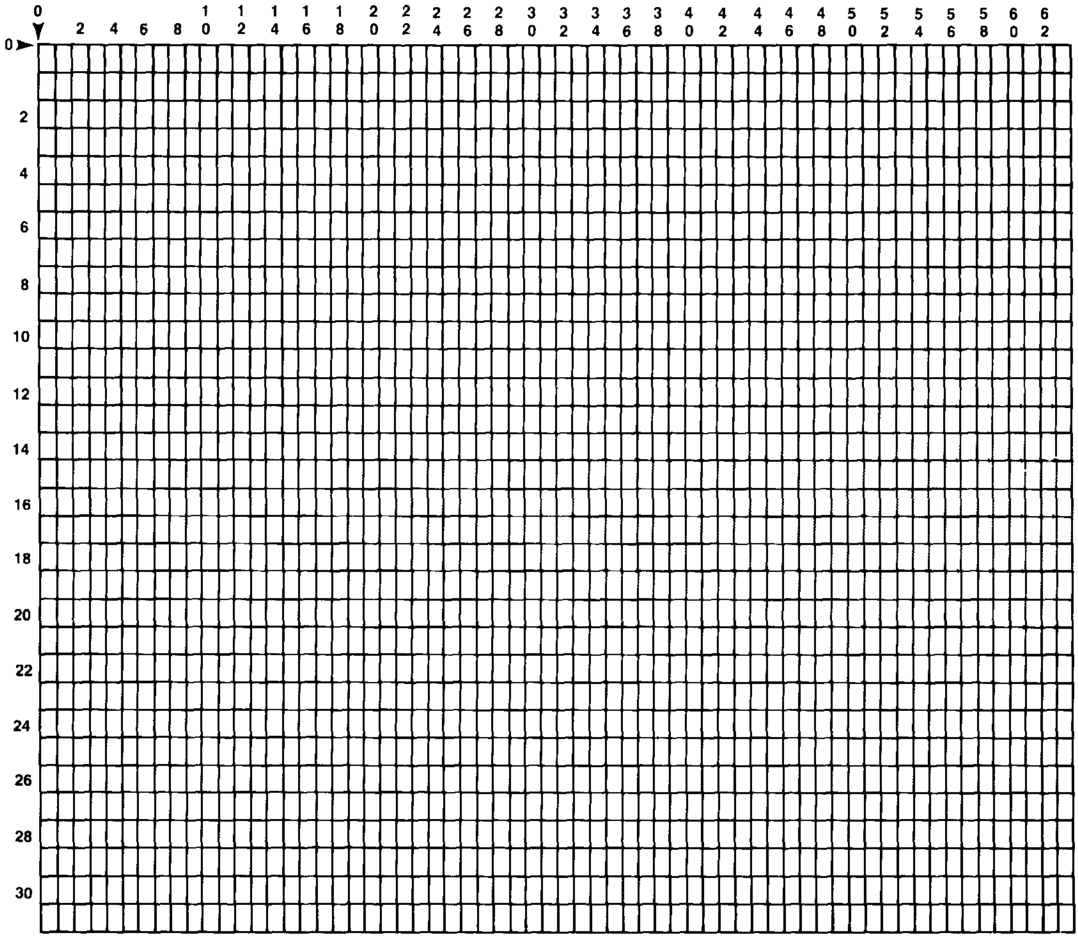
```
80  OPEN "I", #-1, "BOOKS"
85  IF EOF(-1) THEN 120
90  INPUT #-1, T$(B), A$(B), S$(B)
95  B = B+1
110  GOTO 85
120  CLOSE #-1
490  PRINT
500  INPUT "SORT BY (1) TITLE (2) AUTHOR OR (3)
     SUBJECT"; A
510  IF A>3 OR A<1 THEN 500
520  ON A GOSUB 1000, 2000, 3000
530  GOSUB 4000
540  PRINT
550  FOR X = 1 TO B-1
560  PRINT "TITLE :" T$(Z(X))
570  PRINT "AUTHOR: " A$(Z(X))
580  PRINT "SUBJECT: "S$(Z(X))
590  NEXT X
600  PRINT : GOTO 500
800  REM
900  REM BUILD M$ ARRAY
1000  FOR X=1 TO B-1
1010  M$(X) = T$(X)
1020  NEXT X
1030  RETURN
2000  FOR X = 1 TO B-1
2010  M$(X) = A$(X)
2020  NEXT X
2030  RETURN
3000  FOR X = 1 TO B-1
3010  M$(X) = S$(X)
3020  NEXT X
3030  RETURN
3900  REM
4000  REM SORT ROUTINE
4005  T = 1
4010  X = 0
4020  X = X+1
4030  IF X>B-1 THEN RETURN
4040  IF M$(X) = "ZZ" THEN 4020
4050  FOR Y=1 TO B-1
4060  IF M$(Y) < M$(X) THEN X=Y
4065  Z(T) = X
4080  NEXT Y
4090  M$(X) = "ZZ"
4100  GOTO 4010
```

WORKSHEETS

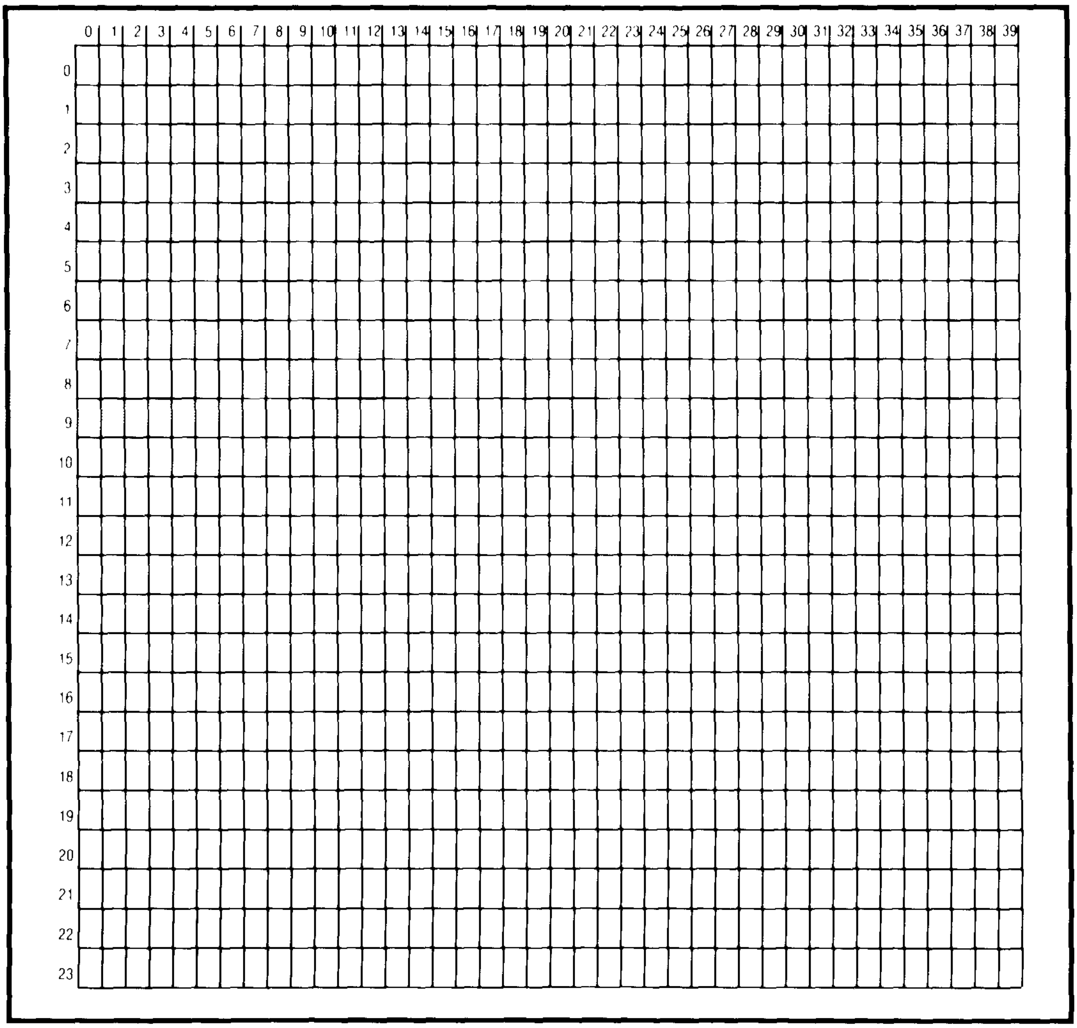
Low-Resolution Text Screen Worksheet (32 x 16)



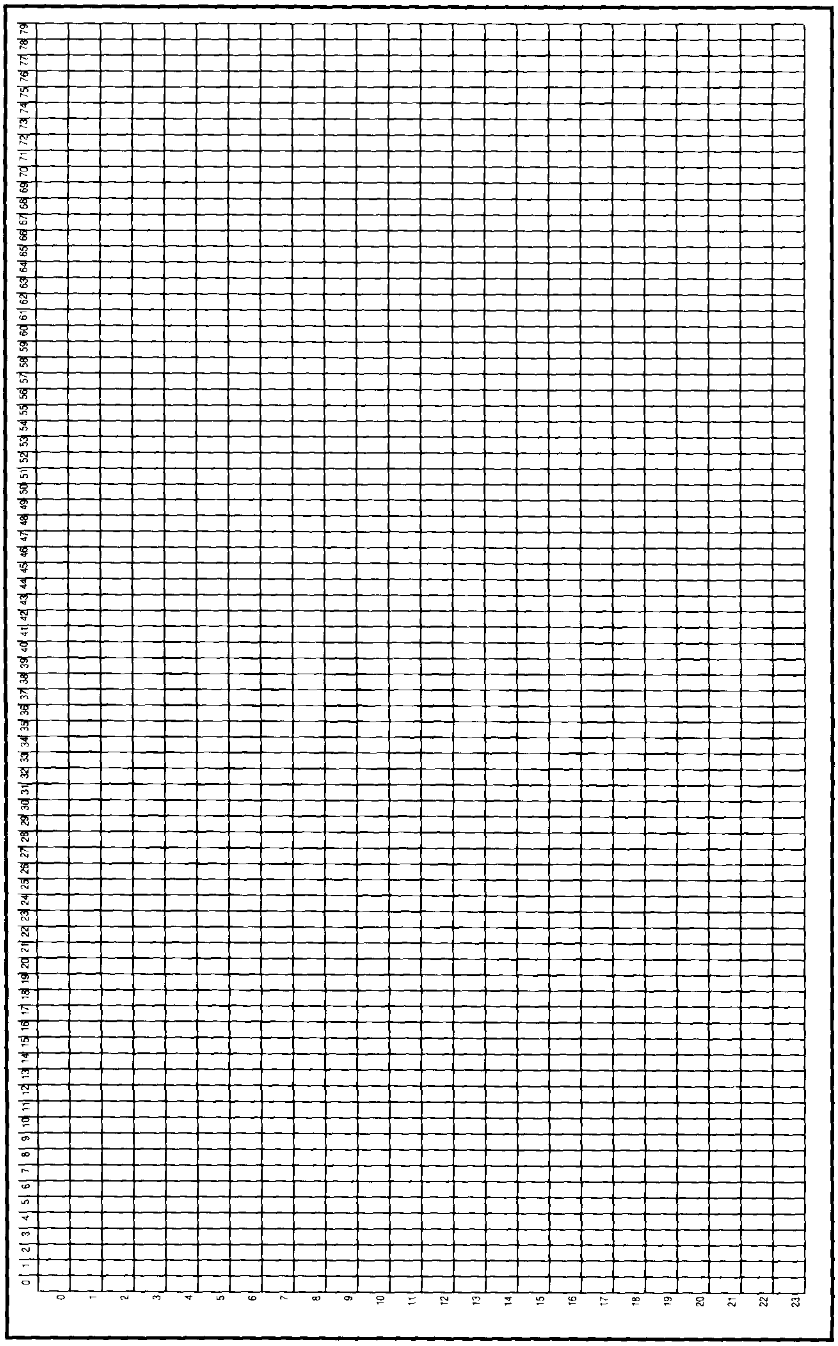
Low-Resolution Text Screen Worksheet (64 x 32)



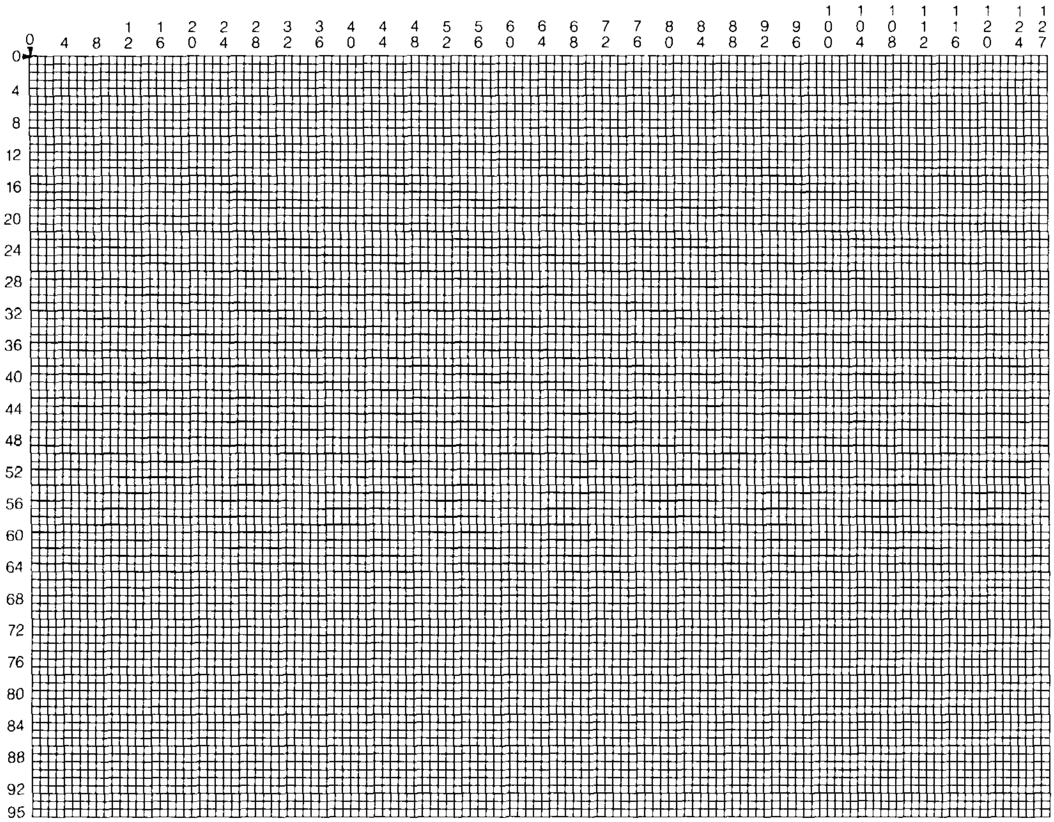
High-Resolution Text Screen Worksheet (40 x 24)



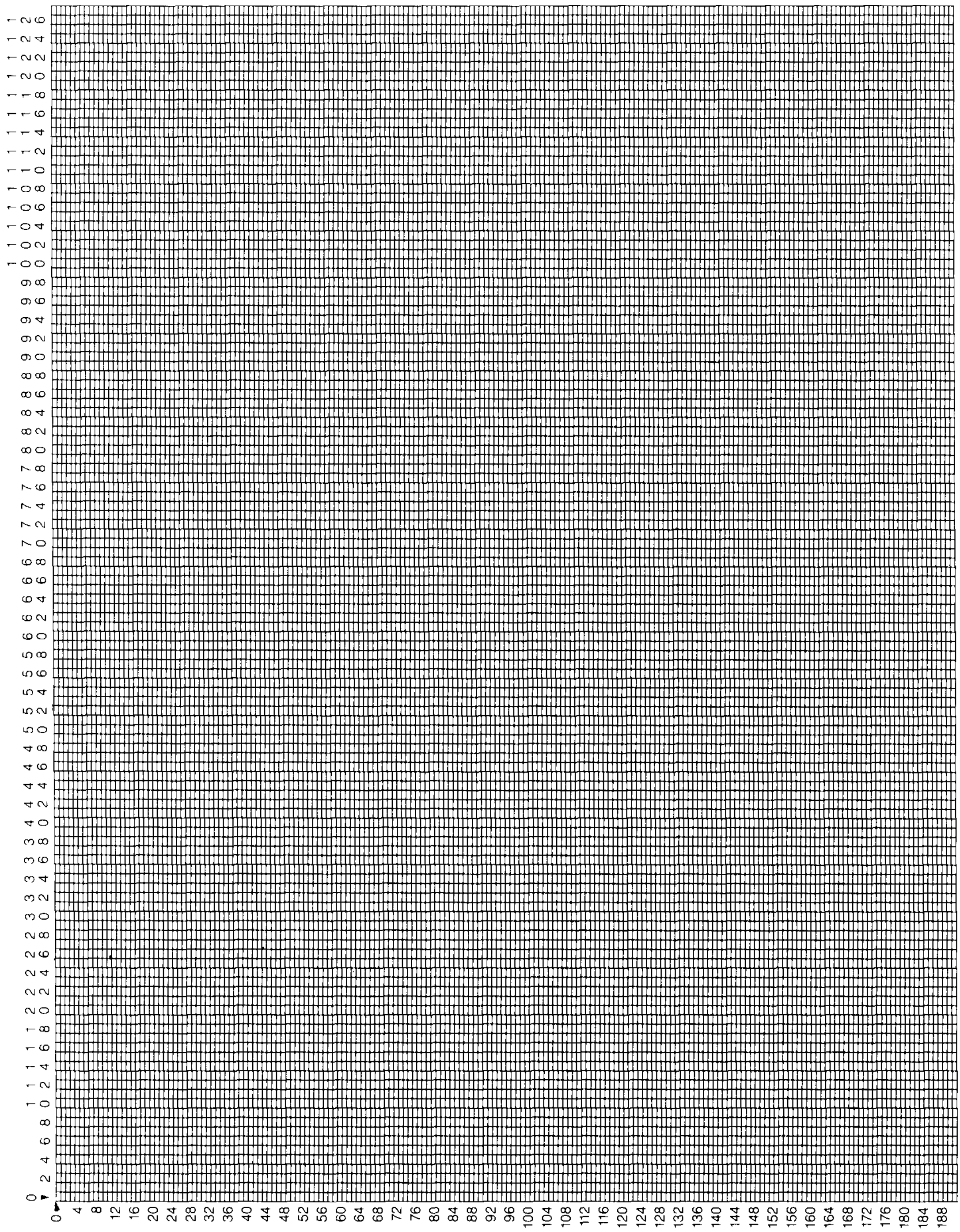
High-Resolution Text Screen Worksheet (80 x 24)



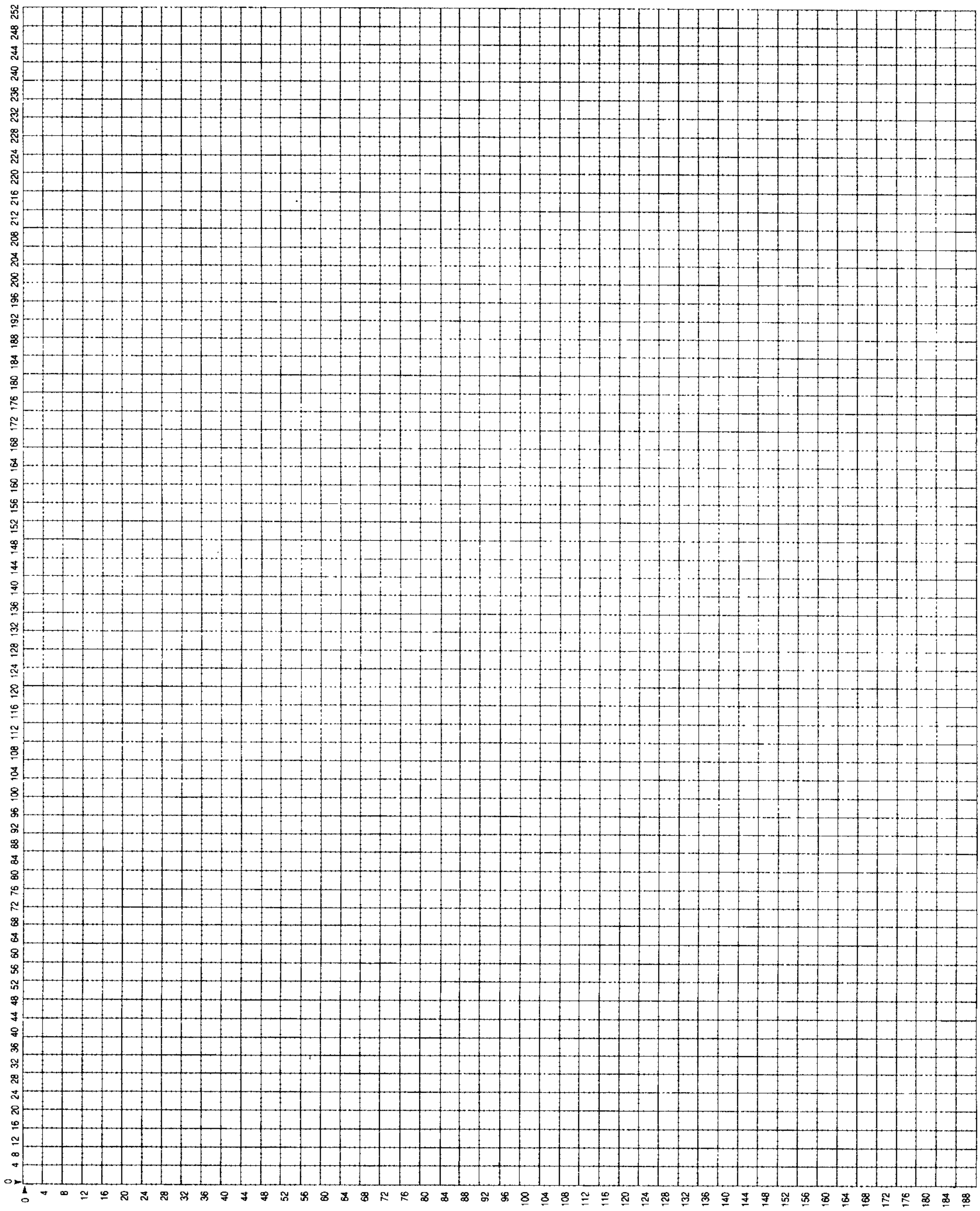
Low-Resolution Graphics Screen Worksheet (128 x 96)



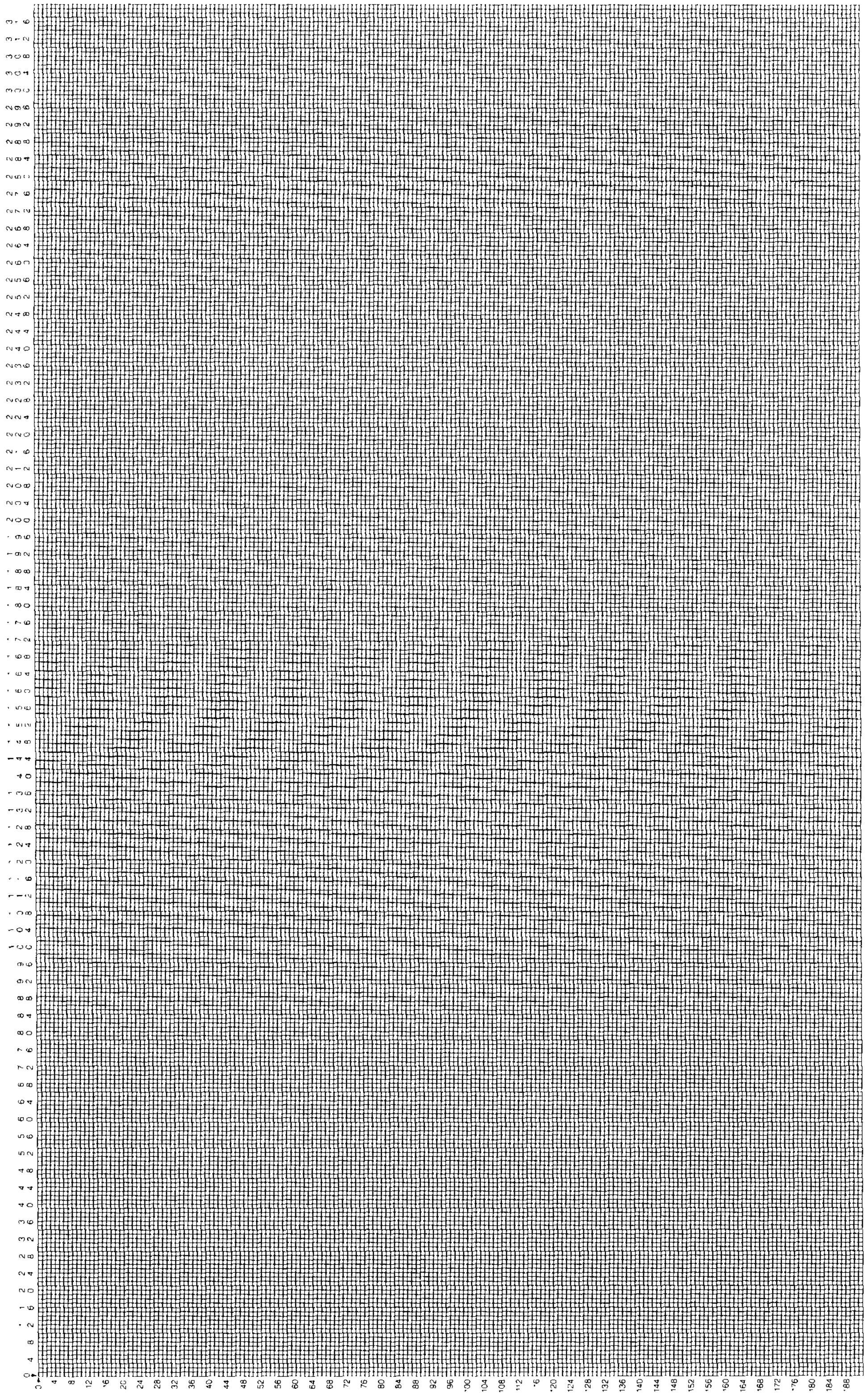
Low-Resolution Graphics Screen Worksheet (128 x 192)



Low-Resolution Graphics Screen Worksheet (256 x 192)



High-Resolution Graphics Screen Worksheet (320 x 192)



BASIC CHARACTER CODES

BASIC uses a code to represent each character that can be typed on the keyboard or displayed on the text screen. This section lists the codes that BASIC uses.

BASIC Codes 0-127/ Low- and High-Resolution Text Screens

For low- and high-resolution text screens, BASIC uses Codes 0-127 to represent the common alphanumeric characters. These codes adhere to a standard, called *ASCII codes*, that most small computers use.

You can generate these codes in two ways: by typing the keyboard character or by entering the CHR$ function. For example, you can generate the code for the letter A by typing the letter **A** or by typing **PRINT CHR$(65)** (ENTER). Both ways cause BASIC to display the letter A on your screen.

To type a lowercase character, you must first enter BASIC's upper/lowercase mode by pressing (SHIFT)(0). (Press (SHIFT) and, while holding down (SHIFT), press (0)). Then, type the keyboard character. For example, type an **A** to generate the code for the letter a.

BASIC displays a lowercase character differently depending on which kind of text screen you are using. If you use the low-resolution screen, BASIC displays the character in uppercase with reversed colors. (For example, |A|.) If you use the high-resolution screen, BASIC displays true lowercase. (For example, a.)

Table 7.1 / BASIC Codes 0-127/ Low and High-Resolution Text Screens

Character	Decimal Code	Hexadecimal Code
(BREAK)	03	03
(←)	8	O8
(→)	9	O9
(↓)	10	OA
(CLEAR)	12	0C
(ENTER)	13	0D
Space bar	32	20
!	33	21
"	34	22
#	35	23
$	36	24
%	37	25
&	38	26
'	39	27
(	40	28
)	41	29
*	42	2A
+	43	2B
,	44	2C
-	45	2D
.	46	2E
/	47	2F

Character	Decimal Code	Hexadecimal Code
0	48	30
1	49	31
2	50	32
3	51	33
4	52	34
5	53	35
6	54	36
7	55	37
8	56	38
9	57	39
:	58	3A
;	59	3B
<	60	3C
=	61	3D
>	62	3E
?	63	3F
@	64	40
A	65	41
B	66	42
C	67	43
D	68	44
E	69	45
F	70	46
G	71	47
H	72	48
I	73	49
J	74	4A
K	75	4B
L	76	4C
M	77	4D
N	78	4E
O	79	4F
P	80	50
Q	81	51
R	82	52
S	83	53
T	84	54
U	85	55
V	86	56
W	87	57
X	88	58
Y	89	59
Z	90	5A

Character	Decimal Code	Hexadecimal Code
[(SHIFT ↓)	91	5B
\ (SHIFT CLEAR)	92	5C
] (SHIFT →)	93	5D
↑	94	5E
← (SHIFT ↑)	95	5F
^	96	60
a	97	61
b	98	62
c	99	63
d	100	64
e	101	65
f	102	66
g	103	67
h	104	68
i	105	69
j	106	6A
k	107	6B
l	108	6C
m	109	6D
n	110	6E
o	111	6F
p	112	70
q	113	71
r	114	72
s	115	73
t	116	74
u	117	75
v	118	76
w	119	77
x	120	78
y	121	79
z	122	7A
{	123	7B
¦	124	7C
}	125	7D
~	126	7E
_	127	7F

BASIC Codes 128-255/ Low-Resolution Text Screen

For its low-resolution text screen, BASIC uses Codes 128-255 to represent its own, unique graphic characters. To generate a graphics character, use the CHR$(*code*) function. You can compute *code* with this formula:

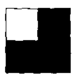
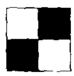
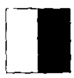
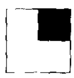
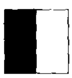
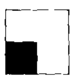
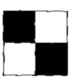
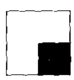
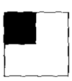
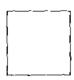
$$code = 128 + 16*(c\text{-}1) + pattern$$

pattern is the number of the graphics pattern (listed in the table below) you want to display. *c* is the color you want to use.

For example, assume that you want to display a large blue box. As shown in the table below, the pattern for a large box is 15. And, assuming your computer's palette is set to the standard colors, you can specify the color blue on a low-resolution text screen as Color 3. To display a large blue box, type:

```
PRINT CHR$(128+16*(3-1)+15)
```

Table 7.2 BASIC Codes 128-255/ Low-Resolution Text Screen

Character	Pattern	Character	Pattern
	0		8
	1		9
	2		10
	3		11
	4		12
	5		13
	6		14
	7		15

Codes 128-159/ High-Resolution Text Screen

For its high-resolution text screen, BASIC uses Codes 128-159 to represent its own, unique foreign and special characters. To generate a foreign or special character, use the CHR$(code) function. For example, you can type **PRINT CHR$(13Ø)** (ENTER) to display an "e" with an accent on the high-resolution text screen.

Table 7.3 BASIC Codes 128-159/ High-Resolution Text Screen

Character	Code	Character	Code
Ç	128	ó	144
ü	129	æ	145
é	130	Æ	146
â	131	ô	147
ä	132	ö	148
à	133	ø	149
å	134	û	150
ç	135	ù	151
ê	136	Ø	152
ë	137	Ö	153
è	138	Ü	154
ï	139	§	155
î	140	£	156
ß	141	±	157
Ä	142	°	158
Å	143	ƒ	159

COLOR CODES

The color computer has 64 colors. Use this table to record names for each of the colors as they appear on your display. Type in and run Sample program #23. Eight colors will appear on the display at a time. First 0-7, then 8-15 on through 56-63. As each screen full of colors appears, write the name of each color on the chart below.

00	________	16	________	32	________	48	________
01	________	17	________	33	________	49	________
02	________	18	________	34	________	50	________
03	________	19	________	35	________	51	________
04	________	20	________	36	________	52	________
05	________	21	________	37	________	53	________
06	________	22	________	38	________	54	________
07	________	23	________	39	________	55	________
08	________	24	________	40	________	56	________
09	________	25	________	41	________	57	________
10	________	26	________	42	________	58	________
11	________	27	________	43	________	59	________
12	________	28	________	44	________	60	________
13	________	29	________	45	________	61	________
14	________	30	________	46	________	62	________
15	________	31	________	47	________	63	________

PALETTE SLOTS

The color computer has a palette containing 16 slots. This palette can hold 16 colors at a time, and you can choose among 64 codes, listed in the previous section, to store in the palette.

When you enter a PALETTE CMP or PALETTE RGB command or when you first start BASIC (in which case, BASIC automatically enters a PALETTE CMP command), BASIC creates a *standard palette.* It does this by storing 16 standard codes in the palette.

BASIC's standard palette is listed in the table below. By using the standard palette, you can run programs written in earlier versions of BASIC.

You can change the standard palette by using the PALETTE command. For example PALETTE 2, 30 stores Code 30 in Slot 2.

Table 7.5 BASIC's Standard Palette (PALETTE CMP or PALETTE RGB)

Slot	Color	CMP Code	RGB Code
0	Green	18	18
1	Yellow	36	54
2	Blue	11	9
3	Red	7	36
4	Buff	63	63
5	Cyan	31	27
6	Magenta	9	45
7	Orange	38	38
8	Black	0	0
9	Green	18	18
10	Black	0	0
11	Buff	63	63
12	Black	0	0
13	Green	18	18
14	Black	0	0
15	Orange	38	38

BASIC COLORS

Each screen uses different palette slots to produce:

- The foreground and background colors.
- The colors you specify, such as the "3" in CLS3.

For example, the low-resolution text screen uses Slot 2 to produce Color 3. The high-resolution text screen uses Slot 2, 3, or 11 to produce Color 3.

This section shows the palette slots that each screen uses to produce the foreground and background colors and the colors you specify.

Low-Resolution Text Screen

Foreground and Background Colors: The slots that produce the foreground and background colors depend on which character the screen is displaying and what that character's code is. (See "Character Codes" to find out each character's code.)

For example, if the screen is displaying a common alphanumeric character (Code 0-127), Slot 12 produces the foreground color and Slot 13 produces the background color. If the screen is displaying Graphics Character 129, Slot 0 produces the foreground color and Slot 8 produces the background color.

The Colors You Specify: The slot that produces your color specification is one less than the specification. For example, Slot 2 produces Color 3.

Table 7.6 BASIC's Use of the Palette
Low-Resolution Text Screen

Color 0	Slot 8
Color 1	Slot 0
Color 2	Slot 1
Color 3	Slot 2
Color 4	Slot 3
Color 5	Slot 4
Color 6	Slot 5
Color 7	Slot 6
Color 8	Slot 7
FOREGROUND:	
Codes 0-127	Slot 12
Codes 128-143	Slot 0
Codes 144-159	Slot 1
Codes 160-175	Slot 2
Codes 176-191	Slot 3
Codes 192-207	Slot 4
Codes 208-223	Slot 5
Codes 224-239	Slot 6
Codes 240-255	Slot 7
BACKGROUND:	
Codes 0-127	Slot 13
Codes 128-255	Slot 8

High-Resolution Text Screen

Foreground and Background Colors: Slot 8 produces the foreground color and Slot 0 produces the background color. Both slots are default slots; you can change them with the ATTR or CLS commands.

The Colors You Specify: The slot that produces the color you specify depends on whether you specify the color with CLS, ATTR as the foreground, or ATTR as the background.

For example, the slot that produces Color 3 could be:

- Slot 2, when specified with CLS.
- Slot 11, when specified with ATTR as the foreground.
- Slot 3, when specified with ATTR as the background.

Table 7.7 BASIC's Use of the Palette High-Resolution Text Screen

	CLS	ATTR FOREGROUND	ATTR BACKGROUND
Color 0	------	Slot 8	Slot 0
Color 1	Slot 0	Slot 9	Slot 1
Color 2	Slot 1	Slot 10	Slot 2
Color 3	Slot 2	Slot 11	Slot 3
Color 4	Slot 3	Slot 12	Slot 4
Color 5	Slot 4	Slot 13	Slot 5
Color 6	Slot 5	Slot 14	Slot 6
Color 7	Slot 6	Slot 15	Slot 7
Color 8	Slot 7	-------	------

FOREGROUND: Slot 8
BACKGROUND: Slot 0

Low-Resolution Graphics Screen

Foreground and Background Colors: The slots that produce the foreground and background colors depend on which PMODE and color set you are using. For example, if you are using a 2-color PMODE with Color Set 0, Slot 9 produces the foreground color and Slot 8 produces the background color.

Regardless of which PMODE and color set you are using, all the slots that produce the foreground and background colors are default slots; you can change them with the COLOR command.

The Colors You Specify: The slot that produces the color you specify depends on which PMODE and color set you are using. For example, the slot that produces Color 3 could be:

- Slot 9, in a 2-color PMODE with Color Set 0
- Slot 11, in a 2-color PMODE with Color Set 1
- Slot 6, in a 4-color PMODE with Color Set 0
- Slot 2, in a 4-color PMODE with Color Set 1

Table 7.8 Color Specifications
Low-Resolution Graphics Screen

	2-CLR PMODE		4-CLR PMODE	
	CLR SET 0	CLR SET 1	CLR SET 0	CLR SET 1
Color 1	Slot 9	Slot 11	Slot 0	Slot 4
Color 2	Slot 8	Slot 10	Slot 1	Slot 5
Color 3	Slot 9	Slot 11	Slot 2	Slot 6
Color 4	Slot 8	Slot 10	Slot 3	Slot 7
Color 5	Slot 9	Slot 11	Slot 0	Slot 4
Color 6	Slot 8	Slot 10	Slot 1	Slot 5
Color 7	Slot 9	Slot 11	Slot 2	Slot 6
Color 8	Slot 8	Slot 10	Slot 3	Slot 7
FOREGROUND	Slot 9	Slot 11	Slot 3	Slot 7
BACKGROUND	Slot 8	Slot 10	Slot 0	Slot 4

High-Resolution Graphics Screen

Foreground and Background Colors: Slot 1 produces the foreground color and Slot 0 produces the background color. Both slots are defaults; you can change them with the HCOLOR command.

The Colors You Specify: The slot that produces the color you specify depends on which HSCREEN setting you are using. For example, the slot that produces Color 3 could be:

- Slot 1, in a 2-color HSCREEN.
- Slot 3, in a 4-color or 16-color HSCREEN.

Table 7.9 BASIC Use of the Palette
High-Resolution Graphics Screen

	2-CLR HSCREEN	4-CLR HSCREEN	16-CLR HSCREEN
Color 1	Slot 1	Slot 1	------
Color 2	Slot 0	Slot 2	Slot 2
Color 3	Slot 1	Slot 3	Slot 3
Color 4	Slot 0	Slot 0	Slot 4
Color 5	------	------	Slot 5
Color 6	------	------	Slot 6
Color 7	------	------	Slot 7
Color 8	------	------	Slot 8
Color 9	------	------	Slot 9
Color 10	------	------	Slot 10
Color 11	------	------	Slot 11
Color 12	------	------	Slot 12
Color 13	------	------	Slot 13
Color 14	------	------	Slot 14
Color 15	------	------	Slot 15

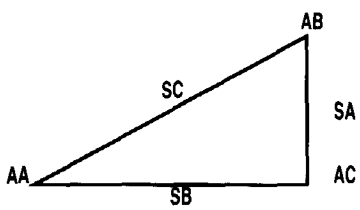
FOREGROUND: Slot 1
BACKGROUND: Slot 0

BASIC MUSICAL TONES

Number	Note
1	C
2	C#/D –
3	D
4	E – /D#
5	E/F –
6	F/E#
7	F#/G –
8	G
9	G#/A –
10	A
11	A#/B –
12	B

MATHEMATICAL FORMULAS WORTH TWO IN THE BOOK . . .

Quantity	Standard Formulas	BASIC Statement
Total Degrees of a Triangle	$180^\circ = A + B + C$	TTL = AA + AB + AC
Solve for Area Given Side a, Angles B and C	$A = 180 - (B + C)$ $Area = \frac{a^2 \sin B \cdot \sin C}{2 \sin A}$	AA = 180 – (AB + AC) [then convert AA, AB and AC to radians] AREA = SA ↑ 2*SIN(AB)*SIN(AC)/(2*SIN(AA))
Given Sides a, b and c	$s = ½(a + b + c)$ $Area = \sqrt{s(s - a)(s - b)(s - c)}$	S = (SA + SB + SC)/2 AREA = SQR(S*(S – SA)*(S – SB)*(S – SC))
Law of Sines	$\frac{a}{b} = \frac{\sin A}{\sin B}$ or $a = \frac{\sin A}{\sin B} \cdot b$	SA = (SIN(AA)/SIN(AB))*SB
Law of Consines	$a^2 = b^2 + c^2 - 2bc \cdot \cos A$ or $a = \sqrt{b^2 +}$ $c^2 - 2bc \cdot \cos A$	SA = SQR(SB ↑ 2 – SC ↑ – 2*SB*SC*COS(AA))
Law of Tangents	$\frac{a - c}{a + c} = \frac{\tan ½(A - C)}{\tan ½(A + C)}$ or $\tan ½(A - C) = \frac{a - c}{a + c} \cdot \tan ½(A \pm C)$	REM Y = TAN((AA – AC)/2) Y = (SA – SC)/(SA + SC)*TAN((AA + AC)/2)
Given Three Sides, Solve for an Angle	$s = ½(a + \ + c$ $r = \uparrow \underline{(s - a)(s - b)(s - c)}$ $A = 2 \arctan (\frac{r}{s - a})$	S = (SA + SB + SC)/2 R = SQR((S – SA)*(S – SB)*(S – SC)/S) AA = 2*ATN(R/(S – SA))
Quadratic Equations	$ax^2 + bx + c = 0$ $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$	REM A*X ↑ 2 + B*Y + C = 0 Z = B ↑ 2 – 4*A*C X1 = (– B + SQR(Z))/(2*A) 'IF Z> = 0 X 2 = (– B – SQR(Z))/(2*A) 'IF Z> = 0
Algebraic	$(a^x)^y = a^{xy}$ $a^x = \frac{1}{a^x}$ $\log x^y = y \cdot \log x$ $\log xy = \log x + \log y$ $\log \frac{x}{y} = \log x - \log y$	Z = (A ↑ X) ↑ Y or Z = A ↑ (X*Y) Z = A ↑ (– X) or Z = 1/(A ↑ X) Z = LOG(X ↑ Y) or Z = Y*LOG(X) Z = LOG(X*Y) or Z = LOG(X) + LOG(Y) Z = LOG(X/Y) or Z = LOG(X) – LOG(Y)



Side a = SA (side opposite Angle A)
Side b = SB (side adjacent to Angle Z)
Side c = SC (hypotenuse)
Angle A = AA
Angle B = AB
Angle C = AC

DERIVED FUNCTIONS

Function	Function Expressed in Terms of Extended Color BASIC Functions. x is in radians.
SECANT	SEC(X) = 1/COX(X)
COSECANT	CSC(X) = 1/SIN(X)
COTANGENT	COT(X) = 1/TAN(X)
INVERSE SINE	ARCSIN(X) = ATN(X/SQR(-X*X+1))
INVERSE COSINE	ARCCOS(X) = -ATN(X/SQR(-X*X+1))+1.5708
INVERSE COSECANT	ARCCSC(X) = ATN(1/SQR(X*X-1))+(SGN(X)-1)*1.5708
INVERSE COTANGENT	ARCCOT(X) = -ATN(X)+1.5708
HYPERBOLIC SINE	SINH(x) = (EXP(X)-EXP(-X))/2
HYPERBOLIC COSINE	COSH(X) = (EXP(X)+EXP(-X))/2
HYPERBOLIC TANGENT	TANH(X) = -EXP(-X)/(EXP(X)+EXP(-X))*2+1
HYPERBOLIC SECANT	SECH(X) = 2/(EXP(X)+EXP(-X))
HYPERBOLIC COSECANT	CSCH(X) = 2/(EXP(X)-EXP(-X))
HYPERBOLIC COTANGENT	COTH(X) = EXP(-X)/(EXP(X)-EXP(-X))*2+1
INVERSE HYPERBOLIC SINE	ARCSINH(X) = LOG(X+SQR(X*X+1))
INVERSE HYPERBOLIC COSINE	ARCCOSH(X) = LOG(X+SQR(X*X-1))
INVERSE HYPERBOLIC TANGENT	ARCTANH(X) = LOG((1+X)/(1-X))/2
INVERSE HYPERBOLIC SECANT	ARCSECH(X) = LOG((SQR(-X*X+1)+1)/X)
INVERSE HYPERBOLIC COSECANT	ARCCSCH(X) = LOG((SGN(X)*SQR(X*X+1)+1)/X)
INVERSE HYPERBOLIC COTANGENT	ARCCOTH(X) = LOG((X+1)/X+1))/2

VALID INPUT RANGES

Inverse sine	$-1 < X < 1$
Inverse cosine	$-1 < X < 1$
Inverse secant	$X < -1$ or $X > 1$
Inverse cosecant	$X < -1$ or $X > 1$
Inverse hyperbolic cosine	$X > 1$
Inverse hyperbolic tangent	X * X < 1
Inverse hyperbolic secant	$0 < X < 1$
Inverse hyperbolic cosecant	X <> 0
Inverse hyperbolic cotangent	X*X > 1

Certain special values are mathematically undefined, but our functions may provide Invalid values:

TAN and SEC of 90 and 270 degrees
COT and SCS of 0 and 180 degrees

For example, TAN(1.5708) returns a value but TAN(90*.01745329) returns a DIVISION BY ZERO error. 90*.01745329 = 1.5708

Other values that are not available from these functions are:

ARCSIN(-1) = -PI/2
ARCSIN(1) = PI/2
ARCOS(-1) = PI
ARCCOS(1) = 0
ARCSEC(-1) = -PI
ARCSEC(1) = 0
ARCCSC(-1) = -PI/2
ARCCSC(1) = PI/2

Please note that the above information may not be exhaustive.

MEMORY MAP

Decimal Address	Contents	Hex Address
0-393215	Unused by BASIC	0-5FFFF
393216-425983	Hires graphics screen memory	60000-67FFF
425984-434175	Hires GET/PUT buffer memory	68000-69FFF
434176-442367	Secondary stack area	6A000-6BFFF
442368-450559	Hires text screen Screen memory	6C000-6DFFF
450560-458751	Unused by BASIC	6E000-6FFFF
458752-459775	System use	70000-703FF
459776-460287	Standard text screen memory	70400-705FF

Standard graphics screen memory

Decimal Address	Contents	Hex Address
460288-461823	Page 1	70600-70BFF
461824-463359	Page 2	70C00-711FF
463360-463359	Page 3	71200-717FF
464896-466431	Page 4	71800-71DFF
466432-467967	Page 5	71E00-723FF
467968-469503	Page 6	72400-729FF
469504-471039	Page 7	72A00-72FFF
471040-472575	Page 8	73000-735FF
472576-491519	Program and variable storage	73600-77FFF
491520-499711	Extended Color BASIC	78000-79FFF
499712-507903	Color BASIC	7A000-7BFFF
507904-516095	Cartridge ROM	7C000-7DFFF
516096-523775	Super Extended BASIC	7E000-7FDFF
523776-524031	Secondary vectors	7FE00-7FFFF
524032-524287	Input/output	7FF00-7FFFF

COLOR COMPUTER LINE PRINTER VARIABLES

Hexadecimal Address	Decimal Address	Initial Hex	Value Dec
LPTBTD Baud			
MSB 0095	149	00	0
LSB 0096	150	57	87
LPTLND Line Delay			
MSB 0097	151	00	0
LSB 0098	152	01	1
LPTCFW Comma Field Width			
0099	153	10	16
LPTWID Line Printer Width			
009B	155	84	132
LPTPOS			
009C	156	00	00

Your computer's software uses the following initial conditions:

- The baud rate is 600
- The printer width is 132 columns
- The printer generates a busy output when not ready
- The printer automatically executes a carriage return at 132 columns.

The RS-232 Interface uses a four-pin DIN connector. A diagram of the Pin out is shown in your introduction manual.

Pin 4 is the computer output to the printer. Pin 3 is ground. Pin 1 is not used for a printer. Pin 2 should be connected to the busy output (or status line) of the printer. If your printer does not provide a status indication, then this line must be connected to a positive voltage of greater than 3 volts. This tells the computer that the printer is ready at all times. In addition, the line delay variable should be set to the proper value.

The following list of alternate values for the line printer variables is provided as an aid in interfacing nonstandard printers.

Baud Rate	Decimal Value	(msb,lsb)	Hexadecimal Value
120 baud	458	(1,202)	01CA
300 baud	180	(0,180)	00BE
600 baud	87	(0,87)	0057
1200 baud	41	(0,41)	0029
2400 baud	18	(0,18)	0012

Line Delay (seconds)	Decimal Value	(",")	Hexadecimal Value
.288	64	(0,64)	4000
.576	128	(0,128)	8000
1.15	65535	(255,255)	FFFF

Line Width (characters/line)	Decimal Value	Hexadecimal Value
16	16	10
32	32	20
64	64	40
255	255	FF

The last comma field variable should be set to the width value — the comma field width. (The comma field width normally stays at 16.)

In Color BASIC, the output format to the printer is 1 start bit, 7 data bits (LSB first), and 2 stop bits with no parity.

ROM ROUTINES

The Color BASIC ROM contains many subroutines that can be called by a machine-language program. Each subroutine will be described in the following format:

NAME — *Entry address*
Operation Performed
Entry Condition
Exit Condition

Note: The subroutine **NAME** is only for reference. It is not recognized by the Color Computer. The **entry address** is given in hexadecimal form; you must use an indirect jump to this address. **Entry** and **Exit Conditions** are given for machine-language programs.

BLKIN = [A006]
Reads a Block from Cassette

Entry Conditions
Cassette must be on and in bit sync (see CSRDON). CBUFAD contains the buffer address.

Exit Conditions
BLKTYP, which is located at 7C, contains the block type:
0 = File Header
1 = Data
FF = End of File
BLKLEN, located at 7D, contains the number of data bytes in the block (0-255).
Z* = 1,A = CSRERR = 0 (if no errors).
Z = 0,A = CSRERR = 1 (if a checksum error occurs).
Z = 0,A = CSREER = 2 (if a memory error occurs).

Note: CSRERR = 81
Unless a memory error occurs, X = CBUFAD + BLKEN. If a memory error occurs, X points to beyond the bad address. Interrupts are masked. U and Y are preserved, all other modified.

**Z is a flag in the Condition Code (CC) register.*

BLKOUT = [A008]
Writes a Block to Cassette

Entry Conditions
The tape should be up to speed and a leader of hex 55s should have been written if this is the first block to be written after a motor-on.
CBUFAD, located at 7E, contains the buffer address.
BLKTYP, located at 7C, contains teh block type.
BLKLEN, located at 7D, contains the number of data bytes.

Exit Conditions
Interrupts are masked. X = CBUFAD + BLKLEN. All registers are modified.

WRITLDR = [A00C]
Turns the Cassette On and Writes a Leader

Entry Conditions
None

Exit Conditions
None

***CHROUT* = [A002]**
Outputs a Character to Device

CHROUT outputs a character to the device specified by the contents of 6F (DEVNUM).
DEVNUM = -2 (printer)
DEVNUM = 0 (screen)

Entry Conditions
On entry, the character to be output is in A.

Exit Conditions
All registers except CC are preserved.

***CSRDON* = [A004]**
Starts Cassette

CSRDON starts the cassette and gets into bit sync for reading.

Entry Conditions
None

Exit Conditions
FIRQ and IRO are masked. U and Y are preserved. All others are modified.

JOYIN = [A00A]
Samples Joystick Pots

JOYIN samples all four joystick pots and stores their values in POTVAL through POTVAL + 3.

Left Joystick

Up/Down	15A
Right/Left	15B

Right Joystick

Up/Down	15C
Right/Left	15D

For Up/Down, the minimum value = UP.
For Right/Left, the minimum value = LEFT.

Entry Conditions
None

Exit Conditions
Y is preserved. All others are modified.

POLCAT = [A000]
Polls Keyboard for a Character

Entry Conditions
None

Exit Conditions
Z = 1, A = 0 (if no key seen).
Z = 0, A = key code, (if key is seen).
B and X are preserved. All others are modified.

ERROR MESSAGES

Abbreviation / Explanation

AO Attempt to Open a file that is already open. If you press RESET during cassette I/O, you'll get this message. Turn the computer off and try again.

BS Bad Subscript. The subscripts in an array are out of range. For example, if you have A(12) in your program without a preceding DIM line that dimensions array A for 12 or more elements, you'll get this error. Use DIM to dimension the array.

CN Can't Continue. If you use the CONT command and you're at the END of program or in other non-continue situations, you'll get this error.

DD Attempt to Redimension an Array. You can dimension an array only once. For example, you can't have DIM A(12) and DIM A(50) in the same program.

DN Device Number Error. You may use only three device numbers with OPEN, CLOSE, PRINT, or INPUT — 0, -1, or -2. If you use another number, you'll get this error.

DS Direct Statement. The data file contains a direct statement. This error can be caused by attempting to CLOAD a data file.

/0 Division by Zero. It's impossible to divide by zero, even for computers.

FC Illegal Function Call. This error occurs when you use a parameter (number or variable) with a BASIC word that is out of range. For example, PLAY":" causes this error.

FD Bad File Data. This error occurs when you PRINT data to a file or INPUT data from the file, using the wrong type of variable for the corresponding data. For example, INPUT #-1,A, when the data in the file is a string, causes this error.

FM Bad File Mode.This error occurs when you try to INPUT data from a file OPEN for OUTPUT(O), or PRINT data into a file OPEN for INPUT(I).

ID Illegal Direct Statement. For example, you can use INPUT only as a line in program, not as a command line.

HP High-resolution Print Error. Attempt to execute a high-resolution text function on a low resolution text screen or to execute a low resolution text function on a high-resolution text screen.

HR High-resolution Graphics Error. Attempt to execute a high-resolution graphic's statement without having first setting up a high-resolution screen with the HSCREEN statement.

IE Input Past End of File. Use EOF to check to see when you've reached the end of the file.1. When you have, CLOSE the file.

IO Input/Output Error. This error is often caused by trying to input a program or data file from a bad tape.

LS String Too Long. A string may contain only 255 characters.

NF NEXT Without FOR. NEXT is being used without a FOR statement. This error also occurs when you have the NEXT lines reversed in a nested loop.

NO File Not Open. You can't input or output data to a file until you have OPENed it.

OD Out of Data. A READ was executed with insufficient DATA for it to READ. A DATA statement may have been left out of the program.

OM Out of Memory. All available memory has been used or reserved.

OS Out of String Space. There is not enough space in memory to do your string operations. You may be able to CLEAR more space.

OV Overflow. The number is too large for the computer to handle. (ABS(x)>IE38)

RG RETURN Without GOSUB. A RETURN line was encountered without a prior GOSUB.

SN Syntax Error. This could result from a misspelled command, incorrect punctuation, open parentheses, or an illegal character. Retype the program line or command.

ST String Formula Too Complex. A string operation was too complex to handle. Break it into shorter steps.

TM Type Mismatch. This occurs when you try to assign numeric data to a string variable1 (A$=3) or string data to a numeric variable (A="DATA").

UL Undefined Line. The program contains a GOTO, GOSUB, or other branching line that asks the computer to go to a nonexisting line number.

ERROR CODES

Number	Code	Comment
0	NF	Next Without For
1	SN	Syntax Error
2	RG	RETURN Without GOSUB
3	OD	Out of Data
4	FC	Illegal Function Call
5	OV	Overflow
6	OM	Out of Memory
7	UL	Undefined Line
8	BS	Bad Subscript
9	DD	Attempt to Redimension Array
10	/0	Division by Zero
11	ID	Illegal Direct Statement
12	TM	Type Mismatch
13	OS	Out of String Space
14	LS	String Too Long
15	ST	String Formula Too Complex
16	CN	Cannot Continue
17	FD	Bad File Data
18	AO	Already Open
19	DN	Device Number Error
20	IO	Input/Output Error
21	FM	Bad File Mode
22	NO	File Not Open
23	IE	Input Past End of File
24	DS	Direct Statement
38	HR	Hires Graphics Error
39	HP	Hires Print Error

BASIC COMMANDS, FUNCTIONS, AND OPERATORS

This section gives a summary of each BASIC command, function, and operator. Please note that the colors you specify with a command or function have different meanings for each screen mode. (See "BASIC Colors.")

Commands

BASIC commands tell your computer to do some action, such as drawing a circle on the screen.

ATTR *c1,c2,*B,U

Sets display attributes of a high-resolution text screen.

- *c1* Foreground color
- *c2* Background color
- *B* Character blink on
- *U* Underline on

```
ATTR 3,2,U
```

AUDIO *switch*

Connects or disconnects cassette output to the display speaker.

- ON Switches ON sound from cassette player to display speaker.
- OFF Switches OFF sound from cassette player to display speaker.

```
AUDIO OFF
```

CIRCLE *(x,y),r,c,h,s,e*

Draws a circle on the current low-resolution graphics screen.

- *x,y* Center point
- *r* Radius
- *c* Color
- *h* Height/width ratio
- *s* Starting point
- *e* Ending point

```
CIRCLE (65,43),20,1,,.5,.8
```

CLEAR *n,h*

Erases variables, reserves string workspace, and reserves high memory for machine language programs.

- *n* String workspace size
- *h* Highest BASIC memory address

```
CLEAR 200,20000
```

CLOAD *"filename"*

Loads program *filename* from cassette. If *filename* is not specified, BASIC loads the first program file found.

- *filename* Name of desired program. Name can have as many as 8 characters.

```
CLOAD "PUPPIES"
```

CLOADM *"filename",o*

Loads machine-language program *filename* from cassette. If *filename* is not specified, BASIC loads the first machine-language program found.

filename	Name of desired machine-language program. Name can have as many as 8 characters.
o	Memory address offset. If specified, BASIC loads the machine-language program o bytes higher in memory than normal.

```
CLOADM "GRAPHICS",2730
```

CLOSE# *d*

Closes access to specified device or file. If *d* is not specified, BASIC closes all open devices and files.

d	Number of device or file

```
CLOSE #-1
```

CLS *c*

Clears the text screen to a specified color. When in high-resolution text mode, BASIC also sets the background color. If *c* is not specified, BASIC uses the current background color.

c	Color code (0-8)

```
CLS 2
```

COLOR *c1, c2*

Sets foreground and background colors of the current low-resolution graphics screen.

c1	Foreground color code (0-8)
c2	Background color code (0-8)

```
COLOR 2,3
```

CONT

Continues program execution after a program halt from the (BREAK) key or a STOP instruction.

```
CONT
```

CSAVE *"filename",A*

Saves program *filename* on cassette.

filename	Name of program to save. Name can have as many as 8 characters.
A	Selects ASCII format.

```
CSAVE "NEWFILE",A
```

CSAVEM *"filename",l,h,e*

Saves machine-language program *filename* on cassette.

filename	Name of machine-language program being saved. Name can have as many as 8 characters.
l	Lowest address of machine-language program.
h	Highest address of machine-language program.
e	Exec address of machine-language program.

```
CSAVEM "GRAPHICS",28000,29000,28032
```

DATA *constant,constant,...*

Stores numeric and string constants for use with READ statement.

constant	String or numeric constant(s), such as: 127.2985 or "Beagle".

```
DATA 45,CAT,98.DOG,24.3,1000
```

DEF FN *name (variables)* = *formula*

Defines a numeric function.

name — Name of function. Must be a valid variable name.
variables — List of dummy variables used in *formula*.
formula — Defines the operation.

```
DEF FNA(B)=B*(B+(1/B))
```

DEFUSR *n* = *addr*

Defines the starting address of a machine-language subroutine.

n — Number of machine-language routine. (0-9)
addr — Starting address of machine-language routine. (0-65535)

```
DEFUSR0=28032
```

DEL *L1-L2*

Deletes program lines.

L1 — Lowest line number to delete.
L2 — Highest line number to delete

- *L1* — Deletes 1 line.
- *-L2* — Deletes from beginning of program up to and including *L2*.
- *L1-* — Deletes from and including *L1* to the end of the program.
- *L1-L2* — Deletes from and including *L1* to and including *L2*.

```
DEL 40-75
```

DIM *array(size),array(size),...*

Dimensions one or more arrays.

```
DIM A$(3,10),R4(22)
```

DRAW *string*

Draws a line on the current low-resolution graphics screen as specified by *string*. The *string* commands are:

A — Angle
BM — Blank move
C — Color
D — Down
E — 45 degree angle
F — 135 degree angle
G — 225 degree angle
H — 315 degree angle
L — Left
M — Move draw position
N — No update
R — Right
S — Scale
U — Up
X — Execute substring

```
DRAW "BM128,96;U25;R25;D25;L25"
```

EDIT *line number*

Edits a program line. After fetching specified *line number*, EDIT recognizes several commands:

C	Changes characters
D	Deletes characters
H	Hacks off rest of line and permits insertion
I	Inserts characters
K	Kills up to cursor
L	Lists line being edited
S	Searches for a specified character
X	Extends line
(SHIFT)(↑)	Returns to line mode

```
EDIT40
```

END

Marks the end of a BASIC program.

```
END
```

EXEC *(address)*

Transfers control to a machine-language program at *address*. If *address* is omitted, control is transferred to the address set by the last CLOADM.

```
EXEC 28032
```

FOR *variable* = *n1* TO *n2* STEP *n3*

Defines the beginning of a loop. The end is specified by NEXT.

variable	Loop counter variable
n1	Starting value of counter
n2	Ending value of counter
n3	Increment or decrement value of counter

```
FOR Z=35 TO 125 STEP 5
```

GET *(sx,sy)-(ex,ey),array,G*

Stores a rectangle that is on the low-resolution graphics screen in an *array*, for future use by the PUT command.

sx,sy	First corner of rectangle
ex,ey	Opposite corner of rectangle
array	Two dimensional array
G	Selects full graphic detail storage. Requires use of PSET, PRESET, AND, OR, or NOT when using PUT.

```
GET (22,34)-(47,38),M,G
```

GOSUB *line number*

Calls a subroutine beginning at the specified line number.

```
GOSUB 330
```

GOTO *line number*

Jumps to the specified line number.

```
GOTO 125
```

HBUFF *buffer,size*

Reserves an area in memory for high-resolution graphics.

buffer	Number of buffer selected
size	Defines buffer size. BASIC allows a *buffer* to have a maximum size of 7931.

```
HBUFF 1,65
```

HCIRCLE (*x*,*y*),*r*,*c*,*h*,*s*,*e*

Draws a circle on the high-resolution graphics screen.

x,y	Center point
r	Radius
c	Color
h	Height/width ratio
s	Starting point
e	Ending point

```
HCIRCLE (55,64),2Ø,2,3,.4,.7
```

HCLS *c*

Clears the high-resolution graphics screen to a specified color.

c	Color If unspecified, BASIC uses current background color.

```
HCLS 2
```

HCOLOR *c1*,*c2*

Sets foreground and background color on the high-resolution graphics screen.

c1	Foreground color (0-15)
c2	Background color (0-15)

```
HCOLOR 2,3
```

HDRAW *string*

Draws a line on the high-resolution graphics screen as specified by *string*. The *string* commands are:

A	Angle
BM	Blank move
C	Color
D	Down
E	45 degree angle
F	135 degree angle
G	225 degree angle
H	315 degree angle
L	Left
M	Move draw position
N	No update
R	Right
S	Scale
U	Up
X	Execute substring

```
HDRAW "BM128,96;U25;R25;D25;L25"
```

HGET (*sx*,*sy*)-(*ex*,*ey*),*buffer*

Stores a rectangle that is on the high-resolution graphics screen into a *buffer* previously set up by the HBUFF command for future use by the HPUT command.

sx,sy	First corner of rectangle
ex,ey	Opposite corner of rectangle
buffer	Number of *buffer*

```
HGET (21,32)-(28,37),1
```

HLINE (*x1,y1*)-(*x2,y2*),*c,a*

Draws a line on the high-resolution graphics screen.

(*x1,y1*)	Starting point of line. If omitted, the line starts at the last ending point, or the center of the screen.
-(*x2,y2*)	Ending point of HLINE.
c	Defines color (Required). PSET selects current foreground color. PRESET selects current background color.
a	Box action (Optional). If omitted, BASIC draws a line. If **B** is used, BASIC draws a box, using the starting and ending points as opposite corners of the box. If **BF** is used, BASIC draws a solid box.

```
HLINE (22,33)-(100,90),3,BF
```

HPAINT (*x,y*),*c1,c2*

Paints an area on the high-resolution graphics screen.

x,y	Starting point
c1	Paint color
c2	Border color

```
HPAINT (55,66),2,3
```

HPRINT (*x,y*),*message*

Prints *message* on the high-resolution graphics screen.

x,y	Starting position
message	String to print

```
HPRINT (20,12),"HELLO!"
```

HPUT (*sx,sy*)-(*ex,ey*),*b,a*

Copies graphics from a buffer to a a rectangle on the high-resolution graphics screen.

sx,sy	First corner of rectangle
ex,ey	Opposite corner of rectangle
b	Buffer number
a	Action used. Actions include: PSET, PRESET, AND, OR, NOT

```
HPUT (22,33)-(28,37),1,PSET
```

HRESET (*x,y*)

Resets a point on the high-resolution graphics screen to the background color.

```
HRESET (22,33)
```

HSCREEN *mode*

Selects a high-resolution graphics screen *mode*. Modes 1-4 also clear high-resolution graphics screen.

mode	Mode number. Mode numbers are: 0 — Low resolution 1 — 320 X 192, 4-color 2 — 320 X 192, 16-color 3 — 640 X 192, 2-color 4 — 640 X 192, 4-color

```
HSCREEN 4
```

HSET (*x,y,c*)

Sets point *x,y* on the high-resolution graphics screen to Color *c*. If you omit *c*, BASIC uses the foreground color.

```
HSET (22,33,2)
```

HSTAT *v1,v2,v3,v4*

Returns information regarding the high-resolution text screen cursor to variables *v1*,*v2*,*v3*, and *v4*.

- *v1* Character code
- *v2* Character attribute
- *v3* Cursor X coordinate
- *v4* Cursor Y coordinate

```
HSTAT C,A,X,Y
```

IF *test* THEN *#1* ELSE *#2*

Performs a test. If the results are true, the computer executes the first instruction (*#1*). If the results are false, the computer executes the second instruction (*#2*).

```
IF A<N THEN PRINT "A<N" ELSE PRINT "A>=N"
```

INPUT *var1,var2,...*

Reads data from the keyboard, and saves it in one or more variables.

```
INPUT K3
```

INPUT #-1 *var1,var2,...*

Reads data from a cassette, and saves it in one or more variables.

```
INPUT #-1,C$
```

LET

Assigns a value to a variable (optional).

```
LET A3=27
```

LINE *(x1,y1)-(x2,y2),c,a*

Draws a line on the current low-resolution graphics screen.

(x1,y1)	Starting point of line. If omitted, the line starts at the last ending point, or the center of the screen.
-*(x2,y2)*	Ending point of line.
c	Defines color (Required). PSET selects current foreground color. PRESET selects current background color.
a	Box action (Optional). If omitted, BASIC draws a line. If **B** is used, BASIC draws a box using the starting and ending points as opposite corners of the box. If **BF** is used, BASIC draws a solid box.

```
LINE (22,33)-(27,39),PSET,BF
```

LINE INPUT

Reads data from the keyboard, and saves it in a variable. Commas are characters, and not delimiters.

```
LINEINPUT A$
```

LIST *L1-L2*

Lists specified program line(s) or the entire program on the screen.

L1 Lowest line number to list.
L2 Highest line number to list.
L1 Lists 1 line.
-L2 Lists from beginning of program up to and including *L2*.
L1- Lists from and including *L1* to the end of the program.
L1-L2 Lists from and including *L1* to and including *L2*.

```
LIST 2Ø-45
```

LLIST *L1-L2*

Lists specified program line(s) or the entire program on the printer.

L1 Lowest line number to list.
L2 Highest line number to list.
L1 Lists 1 line.
-L2 Lists from beginning of program up to and including *L2*.
L1- Lists from and including *L1* to the end of the program.
L1-L2 Lists from and including *L1* to and including *L2*.

```
LLIST -9Ø
```

LOCATE *x,y*

Moves the high-resolution text screen cursor to position *x,y*.

```
LOCATE 2Ø,12
```

LPOKE *location,value*

Stores a value (0-255) in a virtual memory location (0-524287 decimal or 0-$7FFFF hexadecimal).

```
LPOKE 48Ø126,241
```

MID$ (*s,p,l*)

Replaces a portion of the contents of string variable *s* with another string.

s String being modified
p Starting position in string
l Length of section being modified

```
MID$ (A$,4,3)="CAT"
```

MOTOR

Turns the cassette ON or OFF.

```
MOTOR ON
```

NEW

Erases everything in memory.

```
NEW
```

NEXT *v1,v2,...*

Defines the end of a FOR loop.

v1,v2 Optional variable names, used for nested loops. If used, list in reverse order of FOR variables. If omitted, only defines the end of the last loop declared.

```
NEXT X,Y,Z
```

ON BRK GOTO *line number*

Jumps to *line number* when the (BREAK) key is pressed.

```
ON BRK GOTO 12Ø
```

ON ERR GOTO *line number*
Jumps to *line number* when an error occurs.
```
ON ERR GOTO 120
```

ON...GOSUB
Multiway call to specified subroutines.
```
ON A GOSUB 100,230,500,1125
```

ON...GOTO
Multiway branch to specified lines.
```
ON A GOTO 100,230,500,1125
```

OPEN *m,#dev,f*
Opens specified file for data transmission.

m	Transmission mode
	I — Input
	O — Output
#dev	#-2 — Printer
	#-1 — Cassette
	#0 — Keyboard or screen
f	Filename

```
OPEN "O",#-1,"DATA"
```

PAINT *(x,y),c1,c2*
Paints an area on the current low-resolution graphics screen.

x,y	Starting point
c1	Paint color
c2	Border color

```
PAINT (44,55),2,3
```

PALETTE CMP or **RGB**
Resets the palette registers to the standard colors for a composite monitor or a television set (PALETTE CMP), or for an RGB monitor (PALETTE RGB).
```
PALETTE CMP
```

PALETTE *pr, cc*
Stores Color Code *cc* (0-63) into Palette Register *pr* (0-15).
```
PALETTE 1,13
```

PCLEAR *n*
Reserves *n* number of 1.5 K graphics memory pages.
```
PCLEAR 4
```

PCLS *c*
Clears current low-resolution graphics screen with Color *c*. If you omit *c*, BASIC uses the background color.
```
PCLS 0
```

PCOPY *s* TO *d*
Copies low-resolution graphics from source page to destination page.

s	Source page number
d	Destination page number

```
PCOPY 1 TO 2
```

PLAY *string*

Plays music as specified by *string*. The string commands are:

A-G	Notes
L	Length
O	Octave
P	Pause
T	Tempo
#or+	Sharp
–	Flat

```
PLAY "L1;A;A#;A-"
```

PMODE *mode,page*

Selects resolution and first memory page of a low-resolution graphics screen.

mode	0 — 128 x 96 x 2 color 1 — 128 x 96 x 4 color 2 — 128 x 192 x 2 color 3 — 128 x 192 x 4 color 4 — 256 x 192 x 2 color If omitted, BASIC uses the last value set. At power on, BASIC uses 2.
page	Start page. If omitted, BASIC uses the previously set page. At power on, BASIC uses 1.

```
PMODE 4,1
```

POKE *location,value*

Stores a value (0-255) in a memory location (0-65535 decimal or 0-$FFFF hexadecimal).

```
POKE 28000,241
```

PRESET (*x,y*)

Resets a point on the current low-resolution graphics screen to the background color.

```
PRESET (22,33)
```

PRINT *message*

Prints on the text screens.

```
PRINT "HELLO!"
```

PRINT #-1,*data*

Writes data to cassette.

```
PRINT #-1,A$
```

PRINT #-2,*data*

Prints on the printer.

```
PRINT #-2,"HELLO!"
```

PRINT TAB(*n*)

Moves the cursor to column *n* on the low and high-resolution text screens.

```
PRINT TAB(22);"HELLO!"
```

PRINT USING"*format*";*data*

Prints numbers in the specified format on the text screen. The *format* commands are:

#	Formats numbers.
.	Decimal point.
,	Prints comma to the left of every third character.
**	Fills leading spaces with asterisks.
$	Prints leading dollar sign.
$$	Floating dollar sign.
+	Leading or trailing sign.
↑↑↑↑	Exponential format.
–	Minus sign after negative numbers.
!	Prints first string character.
%spaces%	String field. Length of field is number of spaces plus 2.

```
PRINT USING "##.####";1/3
```

PRINT *@n,message*

Prints *message* on low-resolution text screen at position *n*.

```
PRINT @11,"HELLO!"
```

PSET (*x,y,c*)

Sets point *x,y* on the current low-resolution graphics screen to Color *c*. If *c* is omitted, BASIC uses the foreground color.

```
PSET (22,33,2)
```

PUT (*sx,sy*)-(*ex,ey*),*v,a*

Copies graphics from an array to a rectangle on the low-resolution graphics screen.

sx,sy	First corner of rectangle
ex,ey	Opposite corner of rectangle
v	Two dimensional array
a	Action used. Actions include: PSET, PRESET, AND, OR, NOT

```
PUT (22,33)-(27,39),A,PSET
```

READ *var1,var2,...*

Reads the next item(s) in a DATA line. Saves data in specified variable(s).

```
READ A1,B,C7
```

REM *comment*

Lets you insert comments in a program line. The computer ignores everything in the line, after the REM.

```
REM THIS IS A COMMENT LINE
```

RENUM *newline,startline,increment*

Renumbers program lines.

newline	New starting line
startline	Line where renumbering starts
increment	Step value for lines

```
RENUM 1,1,10
```

RESET (*x,y*)

Resets a point on the low-resolution text screen to the background color.

```
RESET (22,33)
```

RESTORE

Sets the computer's pointer back to the first item on the first DATA line.

```
RESTORE
```

RETURN

Returns the computer from a subroutine to the BASIC word following GOSUB.

```
RETURN
```

RUN

Executes a program.

```
RUN
```

SCREEN *type,colors*

Selects low-resolution screen modes and color sets.

type	0 — Text	
	1 — Graphics	
colors	0 — Color set 0	
	1 — Color set 1	

```
SCREEN 0,1
```

SET *(x,y,c)*

Sets point *x,y* on the low-resolution text screen to Color *c*. If you omit *c*, BASIC uses the foreground color.

```
SET (11,11,3)
```

SKIPF *filename*

Skips to next program on cassette tape or to the end of a specified program.

filename Optional name of program to skip over.

```
SKIPF "DATA"
```

SOUND *tone,duration*

Sounds a specified tone for a specified duration.

tone	1-255 sets pitch
duration	1-255 sets duration

```
SOUND 33,22
```

STOP

Stops execution of a program.

```
STOP
```

TIMER = *n*

Sets timer to *n*.

```
TIMER=120
```

TROFF

Turns off program tracer.

```
TROFF
```

TRON

Turns on program tracer.

```
TRON
```

WIDTH *n*

Sets the text screen to Resolution *n*:

32 — 32 X 16 (low-resolution text)
40 — 40 X 24 (high-resolution text)
80 — 80 X 24 (high-resolution text)

```
WIDTH 8Ø
```

Functions

BASIC functions are built-in subroutines that perform some kind of computation on data, such as computing the square root of a number. Use BASIC functions as data within your program lines.

ABS (*n*)

Returns the absolute value of *n*.

```
A=ABS(B)
```

ASC (*string*)

Returns the code of the first character in *string*.

```
A=ASC(B$)
```

ATN (*n*)

Returns the arctangent of *n* in radians.

```
A=ATN(B/3)
```

BUTTON (*n*)

Returns 1 if Joystick Button *n* is being pressed; 0 if Joystick Button *n* is not being pressed. *n* can be:

0 — Right joystick, Button 1 (old joystick)
1 — Right joystick, Button 2
2 — Left joystick, Button 1 (old joystick)
3 — Left joystick, Button 2

```
A=BUTTON(Ø)
```

CHR$ (*n*)

Returns the character corresponding to character code *n*.

```
A$=CHR$(65)
```

COS (*angle*)

Returns the cosine of an *angle* using radians.

```
A=COS(B)
```

EOF (*d*)

Returns FALSE (0) if there is more data; TRUE (-1) if end of file has been read.

d Device number:

−1 Cassette

```
IF EOF(-1)=-1 THEN 22Ø
```

ERLIN

Returns the BASIC line number where an error has occurred.

```
IF ERLIN=11Ø THEN 2ØØ
```

ERNO

Returns the BASIC error number for the error that has occurred.

```
IF ERNO=20 THEN CLOSE
```

EXP (*n*)

Returns a natural exponential number (e↑*n*).

```
A=EXP(B*1.15)
```

FIX (*n*)

Returns the truncated integer of *n*. Unlike INT, FIX does not return the next lower number for a negative *n*.

```
A=FIX(B-.2)
```

HEX$ (*n*)

Returns a string with the hexadecimal value of *n*.

```
PRINT HEX$(A);"=";A
```

HPOINT (*x,y*)

Returns information on point *x,y* from the high-resolution graphics screen:

- 0 Point is reset.
- Code Point is set.

```
IF HPOINT(22,33)=0 THEN 200
```

INKEY$

Checks the keyboard and returns the key being pressed or, if no key is being pressed, returns a null string ("").

```
A$=INKEY$
```

INSTR (*p,s,t*)

Searches a string. Returns location of a target string in a search string.

- *p* Start position of search
- *s* String being searched
- *t* Target string

```
A=INSTR (1,M5$,"BEETS")
```

INT (*n*)

Converts *n* to the largest integer that is less than or equal to *n*.

```
A=INT(B+.5)
```

JOYSTK (*j*)

Returns the horizontal or vertical coordinate (j) of the left or right joystick:

- 0 — Horizontal, right joystick
- 1 — Vertical, right joystick
- 2 — Horizontal, left joystick
- 3 — Vertical, left joystick.

```
A=JOYSTK(0)
```

LEFT$ (*string, length*)

Returns the left portion of a string.
length specifies number of characters returned.

```
A$=LEFT$(B$,3)
```

LEN (*string*)

Returns the length of *string*.

```
A=LEN(B$)
```

LOG (*n*)

Returns the natural logarithm of *n*.

```
A=LOG(B/2)
```

LPEEK (*memory location*)

Returns the contents of a virtual memory location (0-524287 decimal or 0-$7FFFF hexadecimal).

```
A=LPEEK(&H7FFF0)
```

MEM

Returns the amount of free memory.

```
A=MEM
```

MID$ (*s,p,l*)

Returns a substring of string *s*

- *s* Source string
- *p* Starting position of substring
- *l* Length of substring

```
A$=MID$(B$,Z,2)
```

PEEK (*memory location*)

Returns the contents of a memory location (0-65535 decimal or 0-&HFFFF hexadecimal).

```
A=PEEK(30020)
```

POINT (*x,y*)

Returns information on point *x,y* from the low-resolution text screen:

- −1 Point is part of an alphanumeric character
- 0 Point is reset
- Code Point is set

```
A=POINT(22,33)
```

POS (*dev*)

Returns the current print position.

dev Print device number:
- 0 — Screen
- −2 — Printer

```
A=POS(0)
```

PPOINT (*x,y*)

Returns information on point *x,y* from the low-resolution graphics screen:

- 0 Point is reset
- Code Point is set

```
A=PPOINT(22,33)
```

RIGHT$ (*string,length*)

Returns the right portion of a string.

length Specifies number of characters returned.

```
A$=RIGHT$(B$,4)
```

RND (*n*)

Generates a "random" number between 1 and *n* if *n*> 1, or between 0 and 1 if *n*=0.

```
A=RND(0)
```

SGN (*n*)

Returns the sign of *n*:

- −1 — Negative
- 0 — 0
- 1 — Positive

```
A=SGN(A+.1)
```

SIN (*angle*)

Returns the sine of *angle* using radians.

```
A=SIN(B/3.14159)
```

STRING$ (*l,c*)

Returns a string of a repeated character.

l Length of string
c Character used. Can be a code, or a string.

```
A$=STRING$(22,"A")
```

STR$ (*n*)

Converts *n* to a string.

```
A$=(1.234)
```

SQR (*n*)

Returns the square root of *n*.

```
A=SQR(B/2)
```

TAN (*angle*)

Returns the tangent of *angle* using radians.

```
A=TAN(B)
```

TIMER

Returns the contents of the timer (0-65535).

```
A=TIMER/18
```

USR*n* (*argument*)

Calls machine-language subroutine *n*, passes it an argument, and returns a value from the subroutine to the BASIC program.

```
A=USR0(B)
```

VAL (*string*)

Converts a string to a number.

```
A=VAL("1.23")
```

VARPTR (*variable*)

Returns a pointer to where a variable is located in memory.

```
A=VARPTR(B)
```

Operators

BASIC operators perform some kind of operation on data, such as adding two numbers.

↑	Exponentiation
– , +	Unary negative, positive
* , /	Multiplication, division
+ , –	Addition and concatenation, subtraction
NOT, AND, OR	Logical operators
<, >, =, <=, >=, <>	Relational tests

INDEX

TERMS AND CONDITIONS OF SALE AND LICENSE OF TANDY COMPUTER EQUIPMENT AND SOFTWARE PURCHASED FROM RADIO SHACK COMPANY-OWNED COMPUTER CENTERS, RETAIL STORES AND RADIO SHACK FRANCHISEES OR DEALERS AT THEIR AUTHORIZED LOCATIONS

LIMITED WARRANTY

I. CUSTOMER OBLIGATIONS

A. CUSTOMER assumes full responsibility that this computer hardware purchased (the "Equipment"), and any copies of software included with the Equipment or licensed separately (the "Software") meets the specifications, capacity, capabilities, versatility, and other requirements of CUSTOMER

B. CUSTOMER assumes full responsibility for the condition and effectiveness of the operating environment in which the Equipment and Software are to function, and for its installation

II. LIMITED WARRANTIES AND CONDITIONS OF SALE

A. For a period of ninety (90) calendar days from the date of the Radio Shack sales document received upon purchase of the Equipment. RADIO SHACK warrants to the original CUSTOMER that the Equipment and the medium upon which the Software is stored is free from manufacturing defects. **This warranty is only applicable to purchases of Tandy Equipment by the original customer from Radio Shack company-owned computer centers, retail stores, and Radio Shack franchisees and dealers at their authorized locations**. The warranty is void if the Equipment or Software has been subjected to improper or abnormal use. If a manufacturing defect is discovered during the stated warranty period, the defective Equipment must be returned to a Radio Shack Computer Center, a Radio Shack retail store, a participating Radio Shack franchisee or a participating Radio Shack dealer for repair, along with a copy of the sales document or lease agreement. The original CUSTOMER'S sole and exclusive remedy in the event of a defect is limited to the correction of the defect by repair, replacement, or refund of the purchase price, at RADIO SHACK'S election and sole expense. RADIO SHACK has no obligation to replace or repair expendable items.

B. RADIO SHACK makes no warranty as to the design, capability, capacity, or suitability for use of the Software, except as provided in this paragraph. Software is licensed on an "AS IS" basis, without warranty. The original CUSTOMER'S exclusive remedy, in the event of a Software manufacturing defect, is its repair or replacement within thirty (30) calendar days of the date of the Radio Shack sales document received upon license of the Software. The defective Software shall be returned to a Radio Shack Computer Center, a Radio Shack retail store, a participating Radio Shack franchisee or Radio Shack dealer along with the sales document.

C. Except as provided herein no employee, agent, franchisee, dealer or other person is authorized to give any warranties of any nature on behalf of RADIO SHACK

D. **EXCEPT AS PROVIDED HEREIN, RADIO SHACK MAKES NO EXPRESS WARRANTIES, AND ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE IS LIMITED IN ITS DURATION TO THE DURATION OF THE WRITTEN LIMITED WARRANTIES SET FORTH HEREIN.**

E. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation(s) may not apply to CUSTOMER

III. LIMITATION OF LIABILITY

A. **EXCEPT AS PROVIDED HEREIN, RADIO SHACK SHALL HAVE NO LIABILITY OR RESPONSIBILITY TO CUSTOMER OR ANY OTHER PERSON OR ENTITY WITH RESPECT TO ANY LIABILITY, LOSS OR DAMAGE CAUSED OR ALLEGED TO BE CAUSED DIRECTLY OR INDIRECTLY BY "EQUIPMENT" OR "SOFTWARE" SOLD, LEASED, LICENSED OR FURNISHED BY RADIO SHACK, INCLUDING, BUT NOT LIMITED TO, ANY INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR OPERATION OF THE "EQUIPMENT" OR "SOFTWARE." IN NO EVENT SHALL RADIO SHACK BE LIABLE FOR LOSS OF PROFITS, OR ANY INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY BREACH OF THIS WARRANTY OR IN ANY MANNER ARISING OUT OF OR CONNECTED WITH THE SALE, LEASE, LICENSE, USE OR ANTICIPATED USE OF THE "EQUIPMENT" OR "SOFTWARE."**
NOTWITHSTANDING THE ABOVE LIMITATIONS AND WARRANTIES, RADIO SHACK'S LIABILITY HEREUNDER FOR DAMAGES INCURRED BY CUSTOMER OR OTHERS SHALL NOT EXCEED THE AMOUNT PAID BY CUSTOMER FOR THE PARTICULAR "EQUIPMENT" OR "SOFTWARE" INVOLVED.

B. RADIO SHACK shall not be liable for any damages caused by delay in delivering or furnishing Equipment and/or Software.

C. No action arising out of any claimed breach of this Warranty or transactions under this Warranty may be brought more than two (2) years after the cause of action has accrued or more than four (4) years after the date of the Radio Shack sales document for the Equipment or Software, whichever first occurs

D. Some states do not allow the limitation or exclusion of incidental or consequential damages, so the above limitation(s) or exclusion(s) may not apply to CUSTOMER

IV. SOFTWARE LICENSE

RADIO SHACK grants to CUSTOMER a non-exclusive, paid-up license to use the TANDY Software on **one** computer, subject to the following provisions.

A. Except as otherwise provided in this Software License, applicable copyright laws shall apply to the Software

B. Title to the medium on which the Software is recorded (cassette and/or diskette) or stored (ROM) is transferred to CUSTOMER, but not title to the Software.

C. CUSTOMER may use Software on a multiuser or network system only if either, the Software is expressly labeled to be for use on a multiuser or network system, or one copy of this software is purchased for each node or terminal on which Software is to be used simultaneously.

D. CUSTOMER shall not use, make, manufacture, or reproduce copies of Software except for use on **one** computer and as is specifically provided in this Software License. Customer is expressly prohibited from disassembling the Software

E. CUSTOMER is permitted to make additional copies of the Software **only** for backup or archival purposes or if additional copies are required in the operation of **one** computer with the Software, but only to the extent the Software allows a backup copy to be made. However, for TRSDOS Software, CUSTOMER is permitted to make a limited number of additional copies for CUSTOMER'S own use

F. CUSTOMER may resell or distribute unmodified copies of the Software provided CUSTOMER has purchased one copy of the Software for each one sold or distributed. The provisions of this Software License shall also be applicable to third parties receiving copies of the Software from CUSTOMER

G. All copyright notices shall be retained on all copies of the Software

V. APPLICABILITY OF WARRANTY

A. The terms and conditions of this Warranty are applicable as between RADIO SHACK and CUSTOMER to either a sale of the Equipment and/or Software License to CUSTOMER or to a transaction whereby Radio Shack sells or conveys such Equipment to a third party for lease to CUSTOMER.

B. The limitations of liability and Warranty provisions herein shall inure to the benefit of RADIO SHACK, the author, owner and or licensor of the Software and any manufacturer of the Equipment sold by Radio Shack

VI. STATE LAW RIGHTS

The warranties granted herein give the **original** CUSTOMER specific legal rights, and the **original** CUSTOMER may have other rights which vary from state to state

6/86

RADIO SHACK, A Division of Tandy Corporation
FORT WORTH, TEXAS 76102

PRINTED IN KOREA
811019410A

7A6