

TANDY®

GETTING
STARTED
WITH
COLOR
BASIC



Getting Started with Color BASIC:

© 1984 Tandy Corporation, Fort Worth, Texas 76102 U.S.A.
All Rights Reserved.

Reproduction or use, without express written permission from Tandy Corporation, of any portion of this manual is prohibited. While reasonable efforts have been taken in the preparation of this manual to assure its accuracy, Tandy Corporation assumes no liability resulting from any errors or omissions in this manual, or from the use of the information contained herein.

TRS-80 Color BASIC System Software:

© 1984 Tandy Corporation and Microsoft.
All Rights Reserved.

The system software in the Color Computer is retained in a read-only memory (ROM) format. All portions of this system software, whether in the ROM format or other source code form format, and the ROM circuitry, are copyrighted and are the proprietary and trade secret information of Tandy Corporation and Microsoft. Use, reproduction, or publication of any portion of this material without the prior written authorization by Tandy Corporation is strictly prohibited.

Welcome, Newcomers!

If you don't know anything about computers and want us to spare you the long, technical explanations, relax—this book's for you!

Using this as your guide, you can enjoy your computer *right* away. The first section's all you need to get going. The rest is frills.

You'll find—especially at first—that this book has you do many games, songs, and “fun” programs. If you want to do “practical” programs instead, be patient. You'll find plenty of that later. We start you off with the fun programs because they're the quickest way to feel at ease with the computer. Once you feel it's truly an extension of yourself, you can make it do whatever you want.

So sit down and spend a couple of hours with the computer. Type whatever you want. Play with it. Make it do something strange. In other words . . . feel comfortable with it. It can do endless things for you.

. . . And Hello, Old-Timers!

We haven't forgotten you. If you already know how to program, see your *Quick Reference Card*. It summarizes all Color BASIC words. If you want to learn more about Color BASIC words, use the index of this book to find the pages that describe them.

To learn what the Color Computer is capable of, read Section IV. It shows how to program high-resolution graphics and call machine-language programs.

To Get Started . . .

Connect your computer by referring to *Introducing Your Color Computer 2* or *Introducing Your Deluxe Color Computer*.

Then power up your computer:

1. Turn on your television set.
2. Select Channel 3 or 4 on the television set.
3. Set the antenna switch to "COMPUTER."
4. Turn on the computer. The POWER button is on the left rear of your keyboard (when you're facing the front).

This message appears on your screen:

```
COLOR BASIC v, r,  
© 1980 TANDY  
OK
```

(v.r. is two numbers specifying which version and release you have.)

If you don't get this message:

- Turn the computer on and off again.
- Adjust the brightness and contrast on your television set.
- Check all the connections.

If you still don't get this message, refer to "Troubleshooting and Maintenance" in *Introducing Your Color Computer 2* or *Introducing Your Deluxe Color Computer*.

Once you do get the above message, you're ready to start.

How Do You Talk to a Computer?

In this book, you'll learn how to talk to your computer. That's all programming is, by the way. Once you learn how to communicate, you'll be able to get your computer to do whatever you tell it. (Well, almost.)

The computer understands a language called Color BASIC. Color BASIC is a form of BASIC—Beginners All-purpose Symbolic Instruction Code. There are lots of computer languages. Color BASIC just happens to be the language your computer understands.

We'll introduce BASIC words in the order that it's easiest to learn them. When you get midway in the book, you may forget what one of the words means. If this happens, look up the word in your *Quick Reference Card*.



CONTENTS

Section I THE BASICS

Chapter 1	Meet Your Computer	7
	PRINT SOUND CLS	
Chapter 2	Your Computer Never Forgets (. . . unless you turn it off . . .)	13
	Strings Variables	
Chapter 3	See How Easy It is?	18
	NEW INPUT GOTO TURN PRINT, PRINT; LIST IF/THEN	
Chapter 4	Count the Beat	24
	FOR . . . TO . . . STEP NEXT	
Chapter 5	Sing Out the Time	29
	CLS Nested Loops	
Chapter 6	Decisions, Decisions	35
	IF/THEN END	
Chapter 7	Games of Chance	38
	RND PRINT@	
Chapter 8	School Days	42
	DATA READ RESTORE INT CLEAR	
Chapter 9	Arithmetic	48
	GOSUB RETURN REM	
Chapter 10	A Gift with Words	53
	LEN LEFT\$ RIGHT\$ MID\$	
Chapter 11	A Pop Quiz	59
	INKEY\$ VAL	
Chapter 12	More Basics	65
	STOP CONT MEM SGN ABS STR\$ AND OR	

Section II DRAWING PICTURES

Chapter 13	Color the Screen	72
	SET RESET JOYSTK PEEK	
Chapter 14	Games of Motion	80
	POINT	
Chapter 15	The Talking-Computer Teacher	83
	MOTOR AUDIO	
Chapter 16	Faster Graphics	87
	ASCII CHR\$	
Chapter 17	Let's Dance	91

Section III GETTING DOWN TO BUSINESS

Chapter 18	Taping 99 OPEN CLOSE PRINT#-1 INPUT#-1 EOF
Chapter 19	Managing Numbers 105 DIM
Chapter 20	Managing Words 110 LLIST PRINT#-2
Chapter 21	Sorting 114
Chapter 22	Analyzing 117 Multidimensional Arrays

Section IV A LITTLE BYTE OF EVERYTHING

Part A	High-Resolution Graphics 124
Part B	Using Machine-Language Subroutines with Color BASIC 141

APPENDIXES

Appendix A	Musical Tones 149
Appendix B	BASIC Colors and Graphics Characters 150
Appendix C	PRINT@ Screen Locations 151
Appendix D	Graphics Screen Locations 152
Appendix E	ASCII Character Codes 153
Appendix F	Answers to Exercises 155
Appendix G	Subroutines 164
Appendix H	Sample Programs 168
Appendix I	Error Messages 178
Appendix J	BASIC Summary 180
INDEX 185

SECTION I

THE BASICS

In this section you'll learn how to program. Before you start, though, put yourself in the right frame of mind . . .

Don't try to do everything the "correct" way. Don't try to understand everything. Above all, please don't take our word for anything!

Do have fun with your Color Computer. Try out your own ideas. Prove us wrong (if you can). Type anything and everything that comes to mind.

Ready? Turn the page and begin.

MEET YOUR COMPUTER


Have you connected and turned on your computer? Are you ready to give it a first workout?

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

Type whatever you want. Then 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, enough foolishness . . . as soon as you are ready . . ." (It patiently waits for your command.) You're the master—you tell the computer to do whatever you wish.



All letters you type should be **BLACK** with a **GREEN BACKGROUND**. If they're reversed (green with a black background), press the **SHIFT** and **0** (zero) keys at the same time.



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.

Now check your line. Did you put the quotation marks where we have them? If you made a mistake, no problem. Simply press the **←** key and the last character you typed disappears. Press it again and the next to the last disappears (. . . and so on and so on . . .).

See the blinking light?
Wherever you see it, you can
type something.

Ready? This should be on your screen:

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

Press the **(ENTER)** key and watch. Your screen should look like this:

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



Your computer just obeyed you by printing the message you have in quotes. Have it print another message. Type:

```
PRINT "2"
```

Press **(ENTER)**. The computer again obeys you and prints your next message:

```
2
```

Try another one:

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

The computer obeys you by printing:

```
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)
```

Much better. 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 on what the quotes do?

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 figures it out like a numerical problem.

The computer thinks of quotes as a journalist does. If the number's in quotes, the computer must PRINT it exactly as it appears. If it's not in quotes, the computer can interpret it by adding, subtracting, multiplying, or dividing it.

A Color Calculator, No Less!

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's multiplication sign is an asterisk (*), rather than the sign you use in math (X). The computer's so precise that it would get the X multiplication sign mixed up with the X alphabetical character.

Try a few more problems:


```
PRINT "15 * 2 = " 15*2 (ENTER)
PRINT 18 * 18 " IS THE SQUARE OF 18" (ENTER)
PRINT 33.3/22.82 (ENTER)
```

Notice how the computer handles parts in quotes v parts not in quotes.

Now it's your turn. Write two command lines that print these two problems as well as their answers:

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

DO-IT-YOURSELF COMMAND LINES



Actually, there's no "correct" command line. For that matter, there is no correct way of handling your computer. There are many ways of getting it to do what you want. Relieved? . . . Good!

If you use the "correct" command lines, this is what the computer prints on your screen:

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

Ready for the answers:

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


It Has Its Rules . . .

By now, the computer has probably printed some funny little messages on your screen. If it hasn't, type this line, deliberately misspelling the word PRINT:

```
PRINT "HI" (ENTER)
```

The computer prints:

```
?SN ERROR
```

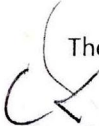
 ?SN ERROR stands for "syntax" error. This is the computer's way of saying, "The command 'PRIINT' is not in my vocabulary. . . I have no earthly idea what you want me to do." Any time you get the ?SN error, you probably made some kind of typographical mistake.

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 5/0 (ENTER)
```

The computer prints:

```
?/0 ERROR
```

 which means, "Don't ask me to 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.

It's a Show-off Too


So far, all you've seen your computer do is silently print on a green screen. But your color computer enjoys showing off. Type:

```
CLS(3) (ENTER)
```

Now your screen is a pretty shade of blue with a green stripe at the top. Your command told the computer to clear the screen and print color number 3—blue.

But why the green stripe? Whenever the computer prints characters, it must use a green background, not a blue background. Type some more characters. The computer uses a green background for them also.

Colors other than green are for printing pictures. You'll learn how to do that later.



If you don't get the right colors, refer to the color test in *Introducing Your Color Computer 2*.

Press **ENTER** to get the OK prompt. Then type:

CLS (7) ENTER

Now your screen is magenta (pinkish purple) with a green stripe at the top. Try some more colors. Use any number from 0 to 8. The Color Computer has nine colors. Each color has a numeric code.

Type CLS without a number code:

CLS ENTER

If you don't use a number code, the computer assumes you simply want a clear green screen.

Computer Sound Off—One, Two . . .

Type this:

SOUND 1, 100 ENTER

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

What you're hearing is 6 seconds of the lowest tone the computer can hum. How about the highest tone? Type:

SOUND 255, 100 ENTER



OK, so it has a good "hum-range" . . . hope you're suitably impressed. Try some other numbers. Hope you like the computer's voice (it's the only one it has).

You want to know what the other number is for? (Or maybe you've already found out.) The second number tells the computer *how long* to hum the tone. You can use any number from 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, 10 ENTER

The computer sounds the tone for 6/10ths of a second. Try variations of both numbers, but keep in the range of 1 to 255.

BUG: If you see a message saying MICROSOFT, or if you see a ?FC Error message, you're using a number other than 0 through 8.



BUG: Again, if you get a ?FC Error message, you're using a number other than 1 through 255.

Before You Continue . . .

Press the **(SHIFT)** and **(0)** (zero) keys, holding both down at the same time. Now release them and type some letters. The letters you type should be green on a black background. If they're not, try again, pressing **(SHIFT)** slightly before **(0)**. Be sure to hold down both keys at the same time and then release them.

Now, with the colors "reversed," press **(ENTER)** and then type this simple command line:

```
PRINT "HI" (ENTER)
```

The computer gives you a ?SN ERROR. It doesn't understand the command.

Press the **(SHIFT)** and **(0)** characters again and release them. Type some letters. They should be back to normal: black with the green background. Press **(ENTER)** and type the same command line again. This time it works.

The computer can't understand any commands you type with reversed colors. If you ever press **(SHIFT)(0)** by mistake and find you're typing with these reversed colors, press **(SHIFT)(0)** again to get the colors back to normal.

Curious about the reversed colors? They're for people with a printer. The printer prints all "reversed" letters in lowercase.

Learned in Chapter 1

BASIC WORDS

PRINT
SOUND
CLS

KEYBOARD CHARACTERS

←
(ENTER)

CONCEPTS

string v numbers
error messages

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

Notes

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

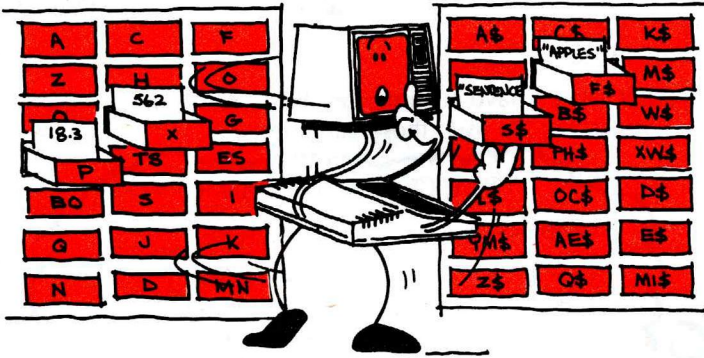
One skill that makes your computer so powerful is its "memory." Have it "remember" the number 13. Type:

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

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

```
PRINT A (ENTER)
```

Did it get confused? or forget?

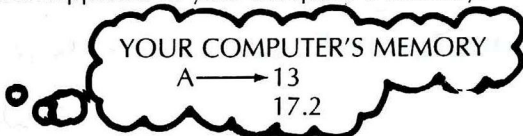


Your computer remembers that A is 13 as long as you have it on . . . or until you do this. Type:

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

If you ask it to PRINT A now, it prints 17.2.

This is what happened in your computer's memory:



If you already know BASIC, you may be accustomed to using the word LET before these command lines. The Color Computer doesn't let you use the word LET.

You don't have to use the letter A. You can use any letters 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)
```

Have it print all the numbers you've asked it to remember. Type:

```
PRINT A, B, C, BC
```

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"
```

To the computer, a dollar sign means a string.

Then type:

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



"Computer types" have a name for all the letters you've used: "variables." So far, you've 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's for numbers. Each box has a label.

Try to set the computer to remember a letter we haven't used yet. What happens? Interesting . . .

As we said before, the computer has its rules and might get a little fussy with you if you don't play by them.

The Computer Is Fussy About Its Rules

Do you think the computer accepts these lines?

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

TM stands for Type Mismatch error. It means you didn't go by the rules.

The computer responds to both above lines with TM ERROR. It's telling you that you have to play by its rules.

The rules "ignored" by the above lines are:

RULES ON STRING DATA

- (1) Any data in quotes is *STRING DATA*.
- (2) You can assign *STRING DATA* only to variables *WITH A \$ SIGN*.

To make the above lines obey the computer's rules, use a dollar sign with the D and Z. Type:

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

The computer now accepts these lines.

How about this line? Do you think the computer accepts it?

```
D$ = 6 ENTER
```

The above line ignored these rules:

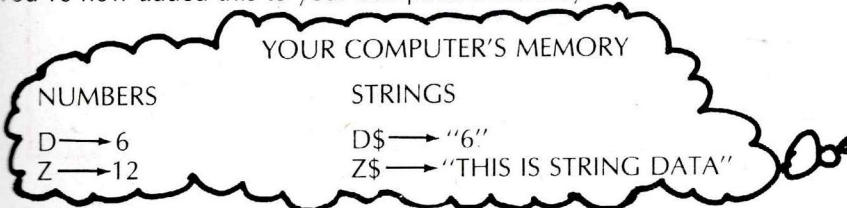
RULES ON NUMERIC DATA

- (1) Numbers not in quotes are *NUMERIC DATA*.
- (2) Numeric data can only be assigned to variables *WITHOUT A \$ SIGN*.

Type this, which the computer accepts:

```
D = 6 ENTER  
Z = 12 ENTER
```

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



Now do something interesting with what you've asked the computer to remember. Type:

```
PRINT D * 2 ENTER
```

The computer prints the product of D times 2.

Try this line:

```
PRINT Z/D
```

The computer remembers that $D = 6$.

The computer prints the quotient of Z divided by D.

Would this work?

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

Did you try it? This makes the computer print the same ?TM ERROR. *It 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 may use any two characters from A to Z for a variable. The first character must be a letter from A to Z; however, the second may be either a numeral or a letter. If you want to assign it string data, put a dollar sign after it. Otherwise, it can hold only numeric data.

Learned in Chapter 2

CONCEPTS

Variables
String v Numeric Variables

Now that you've learned how the computer thinks, it will be easy to write some programs. How about a break, though, before going to the next chapter?

SEE HOW EASY IT IS?

Type:

NEW **(ENTER)**

This erases whatever may 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 typed your first program. Type:

RUN **(ENTER)**

The computer obediently 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.



Since this works so well, add another line to the program. Type:

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

Now type:

LIST **(ENTER)**

Your computer obediently *lists* your entire program. Your screen should look exactly like this:

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

What do you think will happen when you run this? Try it. Type:

RUN **(ENTER)**

The computer prints:

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

Answer the computer's question and then press **ENTER**. . . . What? There's the ?SN Error again.

When you simply type your name, the computer doesn't understand what you mean. In fact, *the computer can't understand anything unless you talk to it in its own way.*

Use a word the computer understands: INPUT. Type this line:

```
30 INPUT A$ ENTER
```

This tells the computer to stop and wait for you to type something, which it labels as A\$. Add one more line to the program:

```
40 PRINT "HI , " A$ ENTER
```

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

```
LIST ENTER
```

Your program should look like this:

```
10 PRINT "HI , I'M YOUR COLOR COM  
PUTER"  
20 PRINT "WHAT IS YOUR NAME?"  
30 INPUT A$  
40 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:

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

```
HI , I'M YOUR COLOR COMPUTER  
WHAT IS YOUR NAME?  
? 772-36-8228  
HI , 722-36-8228
```

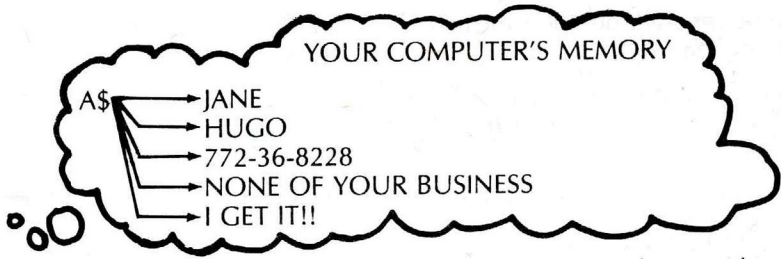
```
HI , I'M YOUR COLOR COMPUTER  
WHAT IS YOUR NAME?  
? NONE OF YOUR BUSINESS  
HI , NONE OF YOUR BUSINESS
```

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

(The computer doesn't care what you call yourself.)

Here's what Line 30 did to your computer's memory each time you ran the program (assuming you gave it the same names we did):

*If you make a mistake after pressing **ENTER**, simply type the line again.*



There's an easier way to run your program over and over without having to type the RUN command. Type this line:

```
50 GOTO 10
```



Now run it. The program runs over and over again without stopping. GOTO tells the computer to go back to Line 10:

```

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

```

Your program now runs perpetually. Each time it gets to Line 50, it goes back to Line 10. We call this a "loop." The only way you can stop this endless loop is by pressing the **BREAK** key.

Spotlight Your Name

Change Line 50 to give your name the attention it deserves. How do you change a program line? Simply type it again, using the same line number. Type:

```
50 GOTO 40
```

This is what the program looks like now:

```

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

```

Type RUN and watch what this loop does. When you've seen enough, press the **BREAK** key.

There's a big change you can make simply by adding a comma or a semicolon. Try the comma first. Type Line 40 again, but with a comma at the end:

```
40 PRINT A$ ,
```

Run the program. The comma seems to print everything in two columns.

Press **BREAK** and try the semicolon. Type:

To delete a program line, type and **ENTER** the line number. For example:
50 **ENTER**
erases Line 50 from the program.

We're leaving out the "HI" part this time.

40 PRINT A\$;

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

NEW **(ENTER)** . . . wish mine worked that easily!

RULES ON PRINT PUNCTUATION

This is what punctuation at the end of a PRINT line makes the computer do:

1. A *comma* makes the computer go to the next column. Use it to print in columns.
2. A *semicolon* makes the computer stay where it is. Use it to "cram" what you print together.
3. *No punctuation* makes the computer go to the next line. Use it to print in rows.

Color/Sound Demonstration

Want to play with color and sound some more? First, erase memory. Remember how?

Then enter this program:

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

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

BUG: If you get a ?FC Error when you run this program, you used a number other than 1 through 255. This error, like all errors, will make the computer stop running the program.

What happens if you change Line 40 to:

```
40 SOUND 50, T
```

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

Remember, if you make a mistake on one of the lines, simply type the line again.

In this program we're using *T* as a variable. However, we could use any letter.

Notice that Line 30 asks for *T* rather than *T\$*. This is because we want numeric data rather than string data.



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

DO-IT-YOURSELF PROGRAM

Press **(BREAK)** first and 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, there are 9 colors, 0 through 8.

HINT: Line 40 could be: 40 CLS(T).

This is our program:

```
10 PRINT "TO MAKE ME CHANGE MY COLOR"  
20 PRINT "TYPE A NUMBER BETWEEN 0 AND 8"  
30 INPUT T  
40 CLS(T)  
50 GOTO 10
```

Add Polish to the Program

Pressing the **BREAK** key is a sloppy way to stop the program from running. Why not have the computer politely ask if you're ready to end? Change Line 50 in the above program to:

Press **BREAK** before typing the line.

```
50 PRINT "DO YOU WANT TO SEE ANOTHER COLOR?"
```

Then add these lines:

```
60 INPUT R$  
70 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:

```
10 PRINT "TO MAKE ME CHANGE COLORS"  
20 PRINT "TYPE A NUMBER BETWEEN 0 AND 8"  
30 INPUT T  
40 CLS(T)  
50 PRINT "DO YOU WANT TO SEE ANOTHER COLOR?"  
60 INPUT R$  
70 IF R$ = "YES" THEN 20
```

This is what the new lines do:

Don't worry about IF/THEN right now. We devote a whole chapter to it later.

- Line 50 prints a question.
- Line 60 tells the computer to stop and wait for an answer: R\$.
- Line 70 tells the computer to go back to Line 20 *IF* (and only if) your answer (R\$) is "yes." If not, the program ends, since it has no more lines.

You've covered a lot of ground in this chapter. Hope we're just whetting your appetite for more.

Don't worry if you don't yet understand it perfectly. Just enjoy using your computer.

Learned in Chapter 3

BASIC WORDS

Characters
NEW
INPUT
GOTO
RUN
PRINT,
PRINT;
LIST
IF/THEN

CONCEPTS

How to Change and Delete a Program Line

KEYBOARD

BREAK

COUNT THE BEAT

The logic of this will become clear later.

In this chapter you'll experiment with computer sound effects. Before doing this, you need to teach the computer to count.

Type:

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

Run the program.

Remember to type **NEW** (**ENTER**) before typing a new program.



Run the program a few more times. Each time, replace Line 10 with one of these lines:

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

Do you see what FOR and NEXT make the computer do? They make it count. Look at the last program we suggested you try:

```
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 should be 20 and the last number should be 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 must print 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 ..."
```

and run the program. With each count, your computer runs 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 program line.

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 Twos

Now make the computer count somewhat differently. Erase your program by typing NEW and then type the original program, using a new Line 10:

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



"2, 4, 6, 8..."

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

- The first X is 2
- The last X is 10
... AND STEP 2 ...
- All the Xs between 2 and 10 are two apart . . . that is 2, 4, 6, 8, and 10. (STEP 2 tells the computer to add two to get each NEXT X.)

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

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

Run the program. This prints on your screen:

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

It passes up the last X (number 10) because $9 + 3 = 12$. Try a few more FOR . . . STEP lines so you can see more clearly how this works:

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

Counting the Sounds

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

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.

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

Don't type the arrow, of course. It's there to help you understand.

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

- Line 20—It prints X, the current count.
- Line 30—It 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.
- Then Line 30 has it sound tone #1.
- Then it goes back to Line 10 and makes X equal to 2
- Etc.

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

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

Did you try it?

PROGRAMMING EXERCISE

Using STEP, change Line 10 so the computer will sound 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
```

Try this: To pause the program while it's running, press the **(SHIFT)** and @ keys at the same time. Then press any key to continue.

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 is in the back of this book.

But Can It Sing?

Yes. Although your computer is slightly off pitch, it can warble out most songs. The next chapter shows how to teach it some of your favorite songs.

Learned in Chapter 4

BASIC WORDS
FOR ... TO ... STEP
NEXT

KEYBOARD CHARACTER

SHIFT @

Notes

SING OUT THE TIME

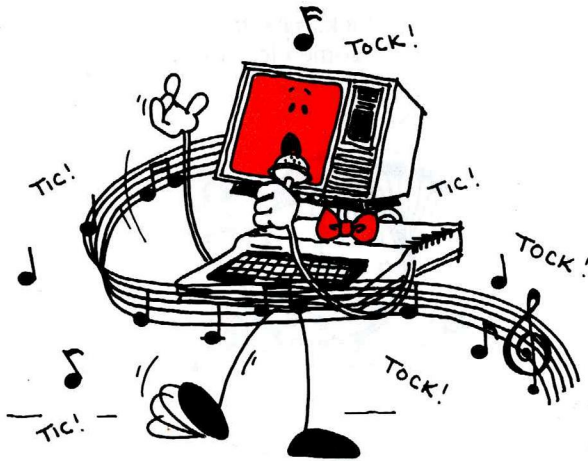
You're now ready to show your computer how to do two tasks: tell time and sing (well, as good as the computer can sing). Since both are closely related—especially to your computer!—we're covering them in the same chapter.

Start by typing this:

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

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

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



As you can see, 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 it. Input the number of seconds you want timed on your stopwatch.

DO-IT-YOURSELF PROGRAM

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.

Here's the program we wrote:

```
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
```

This is how computerized timers work.

Notice the GOTO line at the end of the program. It causes the message to keep printing and the alarm to keep ringing until you press **BREAK** or **SHIFT@**.

Counting Within the Time

Before doing more with the clock, have the computer keep count *within* the time. This concept will become clear to you shortly.



Type this new program:

```
10 FOR X = 1 TO 3  
20 PRINT "X = " X  
30 FOR Y = 1 TO 2  
40 PRINT, "Y = " Y  
50 NEXT Y  
60 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
```

Notice the comma in Line 40. Try it without the comma. The comma makes "Y = " Y print on the next column.

Call it a count within a count or a loop within a loop—whatever you prefer. 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:
 1. It prints the value of Y

Whenever you put a loop inside another loop, you must close the inner loop before closing the outer loop:

Right

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

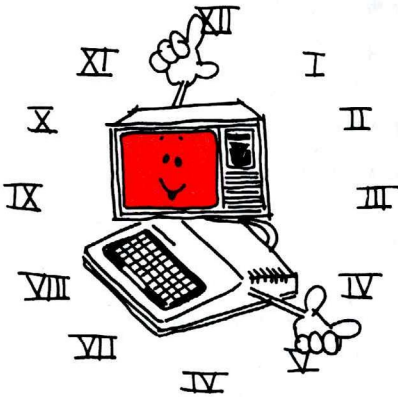
Wrong

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

Making a Clock

With these tools, you can make the computer do much more. Type this:

```
10 FOR S = 0 TO 59
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 it does:

- I. It counts the seconds from 0 to 59. *Each time* it counts one second:
 - A. It prints the second.
 - B. It sounds a tone.
 - C. It pauses long enough for one second to pass.
- II. When it finishes counting all the seconds from 0 to 59, it prints a message that one minute is up.

There's a way to make this program look better. Add this line to clear the screen:


```
15 CLS
```

Now run the program. This time the computer goes through these steps:

- 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).

Using this as groundwork, it's easy to make 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
```



Here's an outline of what the computer does in this program:

- 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 this line, 120
GOTO 10, the clock will run
perpetually.

Having a tough time with
this program? Skip it for
now. It'll seem easy later.

DO-IT-YOURSELF PROGRAM 5-1

Between Lines 90 and 100 you can add some tones that will sound each minute. Write a program that does this.

DO-IT-YOURSELF PROGRAM 5-2

Write a program that makes your computer show each of its nine colors for 1 second each.

The answers to both programs are in the back.

For a Computer, It Sings Great!

But who said this computer could make the opera?

Now, to teach your computer to sing . . .

Flip to the Appendix. There's a table, "Musical Tones," that gives the computer's tone number for each note in the musical scale. The tone number, for example, for middle C is 89.

Unfortunately, the computer's tones can't exactly match most of the notes. That's why it sings somewhat off key . . . But to those without perfect pitch, it's still very close to music.

Type this:

```
20 SOUND 125, 8
30 SOUND 108, 8
40 SOUND 89, 8
```

Run the program. It's the first three notes of . . . well, you know that. Great piece!

To get these first three notes to play over again, put a FOR/NEXT loop in the program:

```
10 FOR X = 1 TO 2
20 SOUND 125, 8
30 SOUND 108, 8
40 SOUND 89, 8
50 NEXT X
```

Now run the program again. It's missing a pause, isn't it? This is easy enough to add. Type these lines:

```
44 FOR Y = 1 TO 230
46 NEXT Y
```

Then run the program again. Now it's starting to sound like the real thing!

Here's a program that gets through the first two phrases:

THREE BLIND MICE

```
10 FOR X = 1 TO 2
20 SOUND 125, 8           "Three"
30 SOUND 108, 8          "blind"
40 SOUND 89, 8           "mice"
44 FOR Y = 1 TO 230      (pause)
46 NEXT Y
50 NEXT X

60 FOR X = 1 TO 2
70 SOUND 147, 8          "See"
80 SOUND 133, 4          "how"
90 SOUND 133, 4          "they"
100 SOUND 125, 8         "run"
110 FOR Y = 1 TO 230     (pause)
120 NEXT Y
130 NEXT X
```

repeat (curved arrow from line 10 to 50)
pause (curved arrow from line 44 to 46)

repeat (curved arrow from line 60 to 130)
pause (curved arrow from line 110 to 120)



Are your programs getting too long to list? Try this: LIST 10-48 (ENTER). Only the first half of this program will be listed.

Finish the song if you like. Or write a better one. A good computer song helps jazz up any program.

Learned in Chapter 5

BASIC WORD

CLS

PROGRAMMING CONCEPT

Nested Loops

Notes

DECISIONS, DECISIONS . . .

Here's an easy decision for the computer:

- If you type "red" . . . then make the screen red
... or
- If you type "blue" . . . then make the screen blue

Easy enough? Then have the computer do it. Type this program:

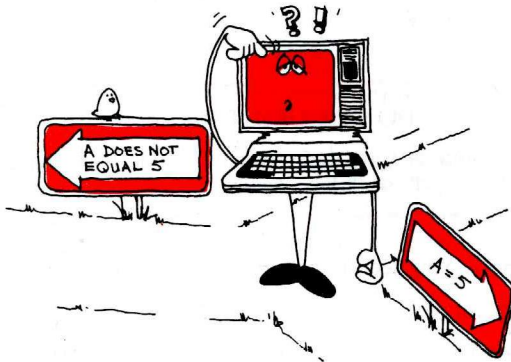
```

10 PRINT "DO YOU WANT THE SCREEN RED OR BLUE?"
20 INPUT C$
30 IF C$ = "RED" THEN 100
40 IF C$ = "BLUE" THEN 200
100 CLS(4)
110 END
200 CLS(3)

```

see the checked
arrows

Don't be confused by the arrows or the spaces between program lines. We just put them in to illustrate the flow of the program.



Run the program a few times. Try both "red" and "blue" as answers.

This is what the program does:

If you answer "red" . . . then . . .

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

. . . On the other hand . . .

If you answer "blue" . . . then . . .

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

What happens if you answer with something different from "red" or "blue"? Run the program again. This time, answer "green."

This makes the screen red. Do you know why?

HINT: If the condition is not true, the computer ignores the THEN part of the line and proceeds to the next program line.

PROGRAMMING EXERCISE

There's a way to get this program to reject any answer but "red" or "blue." These are the two lines to add. You figure out where they go in the program:

```
.... PRINT "YOU MUST TYPE EITHER RED OR BLUE"  
.... GOTO 20
```

Insert the line numbers.

HINT: The lines must come *after* the computer has had a chance to test your answer for "red" or "blue."

HINT: The lines must come *before* the computer makes your screen "red."

Answer: The lines need to come after Line 40 and before Line 100:

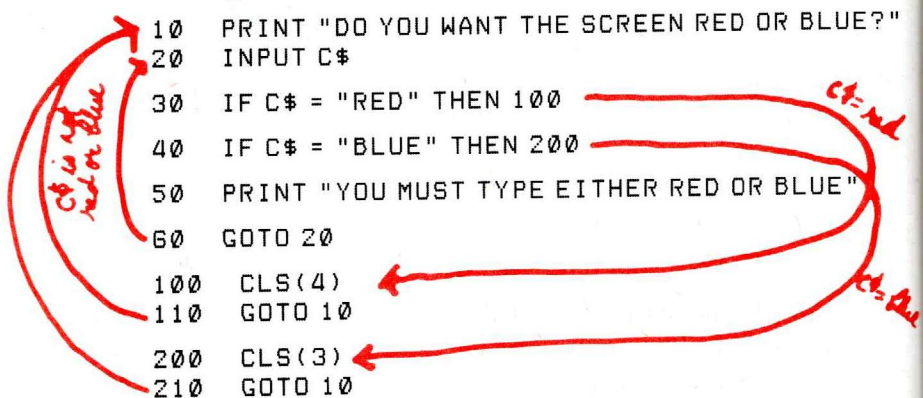
```
50 PRINT "YOU MUST TYPE EITHER RED OR BLUE"  
60 GOTO 20
```

DO-IT-YOURSELF PROGRAM 6-1

After the computer turns the screen red or blue, have it go back and ask you to type "red" or "blue" again.

HINT: You need to change Line 110 and add Line 210.

Here's a diagram of how we wrote this program.



Trace the path the computer takes through this program. Go from one line to the next; follow the arrows where indicated. Notice the difference between the arrows going from the IF/THEN and the GOTO lines.

RULES ON IF/THEN AND GOTO

IF/THEN is conditional. The computer "branches" only if the condition is true.

GOTO is unconditional. The computer always branches.

Although this chapter is short, you've learned an important programming concept. You'll have the computer make decisions all through this book.

Learned in Chapter 6

BASIC WORDS

IF/THEN
END

Notes

GAMES OF CHANCE

Thanks to a BASIC word called RND, the computer can play almost any game of chance.

And even if you don't want to play computer games, you'll want to learn two words this chapter introduces: RND and PRINT @. You'll also find in this chapter some more uses of IF/THEN.

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.



To make the computer pause while running the program, press the **SHIFT** and **@** keys at the same time. Press any key to continue.

Type and run this next program. Press **BREAK** when you satisfy yourself 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);
```

How can you change the program to get random numbers from 1 to 255?

The answer is:

```
10 PRINT RND(255);
```

A Random Show

Just for fun, 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 7-1

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

Here's our program:

```
10 T = RND(255)
14 C = RND(8)
16 CLS(C)
20 SOUND T, 1
30 GOTO 10
```

We have a few simple games in this chapter. Feel free to use your imagination to add interest to them—or invent your own.

Russian Roulette

In this game, a gun has 10 chambers. The computer picks, at random, which of the 10 chambers has the fatal bullet. Type:

```
10 PRINT "CHOOSE YOUR CHAMBER (1-10)"
20 INPUT X
30 IF X = RND(10) THEN 100
40 SOUND 200, 1
50 PRINT "--CLICK--"
60 GOTO 10

100 PRINT "BANG--YOU'RE DEAD"
```

First, in Line 20, the player inputs X (a number from 1 to 10). Then, the computer compares X with RND(10) (a random number from 1 to 10).

Then it follows the "arrows":

- If X is equal to RND(10), the computer goes to Line 100, the "dead routine."
- If X is not equal to RND(10), the computer "clicks" and goes back to Line 10, where you get another chance . . .

Make the dead routine in Line 100 better. Type:

```
100 FOR T = 133 TO 1 STEP -5
110 PRINT "          BANG!!!!!"
120 SOUND T, 1
130 NEXT T
140 CLS
150 PRINT @ 230, "SORRY, YOU'RE DEAD"
160 SOUND 1, 50
170 PRINT @ 390, "NEXT VICTIM, PLEASE"
```

Run the program. Here's what the routine does:

Lines 100-130 make the computer sound descending tones and print BANG!!!! over and over again on the screen.

Remember always to type **NEW (ENTER)** before entering a new program.

Remember how to list part of a program? **LIST 50-130** lists the program's middle part.

Try this when listing a long program: At the start of the listing, press **(SHIFT)** and **(@)**. This causes the listing to pause. Then press any key to continue.

Line 140 clears the screen. Since no color is given, the computer makes the screen green.

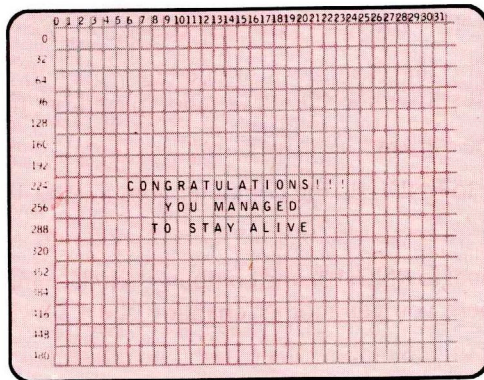
Lines 150 and 170 use a new word—PRINT @—to position two messages on your screen: SORRY, YOU'RE DEAD and NEXT VICTIM, PLEASE.

The grid below shows the 511 positions on your screen. Line 150 prints SORRY, YOU'RE DEAD at position 230 (224 + 6). Line 170 prints NEXT VICTIM, PLEASE at position 390 (384 + 6).



The grid is in the Appendix, "PRINT @ Screen Locations." Use it to plan your programs' screen formats.

DO-IT-YOURSELF PROGRAM 7-2
Change this program so that if the player *does* manage to stay alive for 10 clicks, the computer pronounces the player the winner, printing this message on the screen:



HINT: You can use the FOR/NEXT loop, so that the computer can keep count of the number of clicks.

Our answer is in the Appendix.

Rolling the Dice

This game has the computer roll two dice. To do this, it must come up with two random numbers. Type:

A\$ = "yes"

```
10 CLS
20 X = RND(6)
30 Y = RND(6)
40 R = X + Y
50 PRINT @ 200, X
60 PRINT @ 214, Y
70 PRINT @ 394, "YOU ROLLED A" R
80 PRINT @ 454, "DO YOU WANT ANOTHER ROLL?"
90 INPUT A$
100 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 90 lets you input whether you want another roll. If you answer "yes," the program goes to Line 10 and runs again. Otherwise, since this is the last line in the program, the program ends.

DO-IT-YOURSELF PROGRAM 7-3

Since you know how to roll dice, it should 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's a 2 ("snake eyes"), a 3 ("cock-eyes"), or a 12 ("boxcars"), the player loses and the game's over.
2. If the first roll's a 7 or 11 ("a natural"), the player wins and the game's over.
3. If the first roll's any other number, it becomes the player's "point." 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. Do it. Make the computer print it in an attractive format on your screen and keep the player informed about what is happening. It may take you a while to finish, but give it your best. Good luck!

Our answer's in the back.

Learned in Chapter 7

BASIC WORDS

RND
PRINT @

Notes

SCHOOL DAYS

Your computer is a natural at teaching. It's patient, tireless, and never makes a mistake. Depending on the programmer (you, of course), it also can be imaginative, consoling, and enthusiastic.

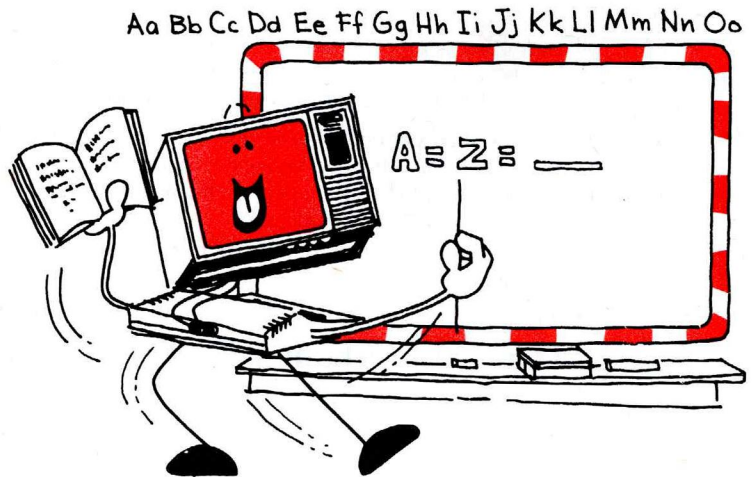
Using RND, have it teach you math. Type:

```

10 CLS
20 X = RND(15)
30 Y = RND(15)
40 PRINT "WHAT IS" X "*" Y " ? "
45 INPUT A
50 IF A = X * Y THEN 90
60 PRINT "THE ANSWER IS" X*Y
70 PRINT "BETTER LUCK NEXT TIME"
80 GOTO 100
90 PRINT "CORRECT!!!"
100 PRINT "PRESS <ENTER> WHEN READY FOR
    ANOTHER"
105 INPUT A$
110 GOTO 10
  
```

Are your programs getting long? If you have a cassette recorder, read your computer's introduction manual to learn how to save your programs on tape. If you have a Deluxe Color Computer, you can also save programs in memory. See your introduction manual to learn how.

The above program drills you on the multiplication tables, from 1 to 15, and checks your answers.



DO-IT-YOURSELF PROGRAM 8-1

Make the program drill you on addition problems from 1 to 100.

Here are the lines we changed:

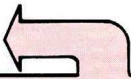
```
20 X = RND(100)
30 Y = RND(100)
40 PRINT "WHAT IS" X "+" Y
45 INPUT A
50 IF A = X + Y THEN 90
60 PRINT "THE ANSWER IS" X + Y
```

Make the program more interesting. Have it keep a running total of all the correct answers. Type:

```
15 T = T + 1
95 C = C + 1
98 PRINT "THAT IS" C "CORRECT OUT OF" T
   "ANSWERS"
```

T is a "counter." It counts how many questions you're asked. When you first start the program, T equals zero. Then each time the computer gets to Line 15, it adds 1 to T.

C is also a counter. It counts your correct answers. Since C's in Line 95, the computer doesn't increase C unless your answer's correct.



When you first turn on the computer, all numeric variables equal 0. When you type NEW (ENTER), all numeric variables also equal 0.

DO-IT-YOURSELF PROGRAM 8-2

Make the program more fun. Have it do one or more of the following:

1. Call you by name.
2. Reward your correct answer with a sound and light show.
3. Print the problem and messages attractively on your screen. (Use PRINT @ for this.)
4. Keep a running total of the percentage of correct answers.
5. End the program if you get 10 answers in a row correct.

Use your imagination. We have a program in back that does this all.

First, Build Your Computer's Vocabulary . . .

To build your computer's vocabulary (so that it can build yours!), type and run this program:

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

What happened . . . nothing? Nothing that you can see, that is. To see what the computer is doing, add this line and run the program:

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

Line 30 tells the computer to:

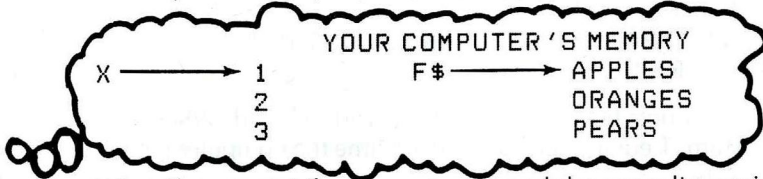
1. Look for a DATA line.
2. READ the first item in the list—APPLES.

3. Give APPLES an F\$ label.
4. "Cross out" APPLES.

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. "Cross out" ORANGES.

When you run the program, this happens in the computer's memory:



What if you want the computer to read the same list again? It's already "crossed out" all the data . . . Type:

```
60 GOTO 10
```

Run the program. You get an error: ?OD ERROR IN 30. OD means "out of data." The computer's crossed out all the data.

Type this line and run the program:

```
50 RESTORE
```

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

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

Remember how to make the computer pause while running a program? Press **(SHIFT) @** to pause and any key to get it to continue.

<pre> 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 </pre>	<pre> 10 DATA APPLES, ORANGES 20 DATA PEARS 30 FOR X = 1 TO 3 40 READ F\$ 50 PRINT "F\$ = :" F\$ 60 NEXT X </pre>
<pre> 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 </pre>	<pre> 30 FOR X = 1 TO 3 40 READ F\$ 50 PRINT "F\$ = :" F\$ 60 NEXT X 70 DATA APPLES, ORANGES, PEARS </pre>

Now Have It Build Your Vocabulary

Here are some words and definitions to learn:

- 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

Now get the computer to select one of these words at random. Hmm... there are ten items. Maybe this works:

```

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. It doesn't work quite right. The computer's 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. Fortunately, BASIC has a word that helps with this. Type:

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

Now run the program a few times again. This time, it should work.

INT tells the computer to look at only the "whole part" of the number and ignore the decimal part. For instance, the computer sees INT(3.9) as 3.

Assume N, the random number, is 10. The IF clause in Line 65 does this:

```

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

```

The above is true: 5 does equal 5. Since it's true, the computer completes the THEN clause. N is adjusted to equal 9 (10 - 1).

Now assume N, the random number, is 9. The IF clause in Line 65 does this:

```

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

```

The above is not true: 4 does not equal 4.5. Since it's not true, the computer doesn't complete the THEN clause. N remains 9.

Besides reading a random word, the computer also must read the word's definition. Add these lines to the end of the program:

```

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

```

Now run the program a few times.

Have the computer print one random word and definition after the next. Add this to the start of the program:

```
5 CLEAR 100
```

This reserves plenty of "string space." Add these lines to the end of the program:

```
130 RESTORE
140 GOTO 60
```

This lets the computer pick a new random word and its definition from a "restored" group of data items.

If you like, add some more words and definitions by adding DATA lines.

For variations on this program, you might try states and capitals, cities and countries, foreign words and meanings.

Here's how the program now looks:

```
5 CLEAR 100
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 8-3

Want to complete this program? Program it so that the computer:

1. Prints the definition *only*.
2. Asks you for the word.
3. Compares the word with the correct random word.
4. Tells you if your answer is correct. If your answer is incorrect, prints the correct word.

Here's our program:

```
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
```

Feel free to add frills such as a good-looking screen format or sound.

Learned in Chapter 8

BASIC WORDS

DATA
READ
RESTORE
INT
CLEAR

Notes

ARITHMETIC

Solving long math problems fast and accurately is a task your computer does with ease. Before typing long, difficult formulas, though, there're some shortcuts you'll want to use.

An easy way to handle complicated math formulas is with "subroutines." 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

```

$$Ax (BY + C) - D + E (G/W) - F$$



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.

Delete Line 40 and see what happens when you run the program.

.....
If you did this, your screen shows this:

```

EXECUTING THE MAIN PROGRAM
EXECUTING THE SUBROUTINE
NOW BACK IN THE MAIN PROGRAM
EXECUTING THE SUBROUTINE
?RG ERROR IN 510

```

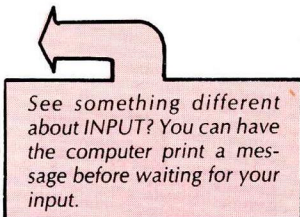
RG means "RETURN without GOSUB." Do you see why deleting END in Line 40 causes this error?

At first, the program runs just as it did before. It goes to the subroutine in Line 500 and then returns to the PRINT line that immediately follows GOSUB.

Then, since you deleted END, it goes to the next line—the subroutine in Line 500. This time, though, it doesn't know where to return. This is because it's merely "dropping" into the subroutine; it is not being sent to the subroutine by a GOSUB line.

This subroutine raises a number to any power:

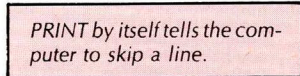
```
10 INPUT "TYPE A NUMBER"; N
20 INPUT "TYPE THE POWER YOU WANT IT RAISED
   TO"; P
30 GOSUB 2000
40 PRINT : PRINT N "TO THE POWER OF" P "IS" E
50 GOTO 10
2000 REM FORMULA FOR RAISING A NUMBER TO A
    POWER
2010 E = 1
2020 FOR X = 1 TO P
2030 E = E * N
2040 NEXT X
2050 IF P = 0 THEN E = 1
2060 RETURN
```



See something different about INPUT? You can have the computer print a message before waiting for your input.

Also introduced in this program are:

- The colon (:), in Line 40. You can combine program lines using the colon to separate them. Line 40 contains the two lines: PRINT and PRINT N "TO THE" P "POWER IS" E.
- REM, in Line 2000. REM means nothing to the computer. Put REM lines wherever you want in your program to help you remember what the program does; they make *no difference in the way the program works*. To see for yourself, add these lines and run the program:



PRINT by itself tells the computer to skip a line.

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

DO-IT-YOURSELF PROGRAM 9-1

Change the above program so that the computer prints a table of squares (a number to the power of 2) for numbers, say, from 2 to 10.

The answer's in the back.

Give the Computer a Little Help

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 may want the computer to arrive at the answer this way:

$$13 + 3 / 8 = 16 / 8 = 2$$

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

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

The computer solves problems logically, using its own rules:



An "operation" is a problem you want the computer to solve. Here the operations are addition, subtraction, multiplication, and division.

RULES ON ARITHMETIC

The computer solves arithmetic problems in this order:

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

In the problem above, the computer follows its rules:

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

For 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, though, it solves the inside parentheses and then moves to the outside parentheses. In other words, it does this:

$$(10 - (5 - 1)) / 2$$

$$(10 - 4) / 2$$

$$6 / 2$$

$$6 / 2 = 3$$

RULES ON PARENTHESES

1. The computer solves operations enclosed in parentheses first, before solving any others.
2. 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)))

Saving Routines

The program below uses two subroutines. It's for those of you who save by putting the same amount of money in the bank each month:

```
10 INPUT "YOUR MONTHLY DEPOSIT" ; D
20 INPUT "BANK'S ANNUAL INTEREST RATE" ; I
30 I = I/12 * .01
40 INPUT "NUMBER OF DEPOSITS" ; P
50 GOSUB 1000
60 PRINT "YOU WILL HAVE $" FV "IN" P "MONTHS"
70 END

1000 REM COMPOUND MONTHLY INTEREST FORMULA
1010 N = 1 + I
1020 GOSUB 2000
1030 FV = D * ((E - 1) / I)
1040 RETURN

2000 REM FORMULA FOR RAISING A NUMBER TO A
    POWER
2010 E = 1
2020 FOR X = 1 TO P
2030 E = E * N
2040 NEXT X
2050 IF P = 0 THEN E = 1
2060 RETURN
```



Notice that one subroutine “calls” another. This is fine with the computer as long as:

- there’s a GOSUB to send the computer to each subroutine, and
- there’s a RETURN at the end of each subroutine.

Turn to the Appendix, “Subroutines.” You’ll find useful math subroutines you can add to your programs.

Learned in Chapter 9

BASIC WORDS

GOSUB
RETURN
REM

BASIC SYMBOLS

()
:

BASIC CONCEPTS

Order of operations

Notes

A GIFT WITH WORDS

A great skill of the computer is its gift with words. It can tirelessly twist and combine words any way you want. With this gift, you can get it to read, write, and even talk.

Not impressed? Later, we'll show practical uses of this unusual skill.

Combining Words

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



Erase the program and run this, 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$

```

Here the plus sign (+) combines strings. For example, D\$ ("IS A ROSE") is a combination of B\$ + C\$.

There are two problems you may encounter 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").

You will not get the OS error if you have not started your computer since you ran the program from Chapter 8 with the CLEAR 500 line.

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 (up to 500), add this line to the start of the program and run:

```
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 255 characters. When storing more than 255 characters, you need to put these characters into several strings.

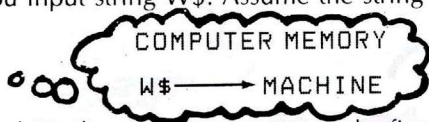
Twisting Words

Now that you can combine strings, try to take a string apart. 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:



In Lines 20 and 30, the computer computes the first *left* letter and the last two *right* letters of 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
```

so that your computer will 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 that the computer will give you the first 5 letters and the last 6 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)
```



```
30 PRINT "THE LAST SIX LETTERS ARE :" RIGHT$
(W$,6)
```

.....

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
```

Remember how to erase a program? Type: NEW **ENTER**

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

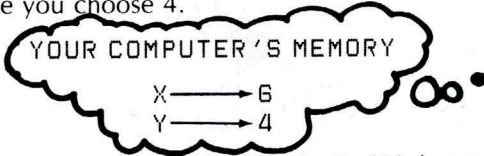
Here's how the program works:

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

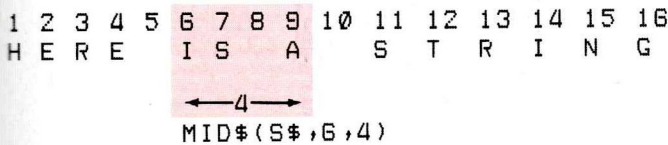


In Line 30, the computer first computes the length of S\$, which is 16 characters. It then asks you to choose a number from 1 to 16. Assume you choose 6.

In Line 60, the computer asks you to choose another number from 1 to 12 (16-6+1). Assume you choose 4.



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



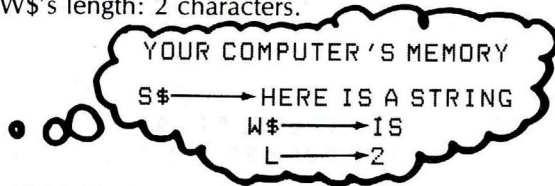
For another example of MID, erase the program and run this:

```
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
```

You can use this kind of program to sort through information. For instance, by separating strings, you could look through a mailing list for TEXAS addresses.

Here's how the program works:

In Line 20, assume you input the word IS for W\$. In Line 30, the computer counts W\$'s length: 2 characters.



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 finally finds IS, the mid-string for which it is searching.



DO-IT-YOURSELF PROGRAM 10-1

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 10-2

Add to the above program to make it:

- Find the start of this mid-string:
A SENTENCE
- Delete the above mid-string to form this new string:
IT'S EASY TO CHANGE
- Add these words to the end of the new string:
ANYTHING YOU WANT
- 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$
```

This program is the basis of a "word processing" program—a popular program that cuts down typing expenses.

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

Learned in Chapter 10

BASIC WORDS

LEN
LEFT\$
RIGHT\$
MID\$

BASIC String OPERATOR

+

Notes

A POP QUIZ

By using a word named INKEY\$, you can get the computer to constantly "watch," "time," or "test" what you're typing. Type and run this program:

```

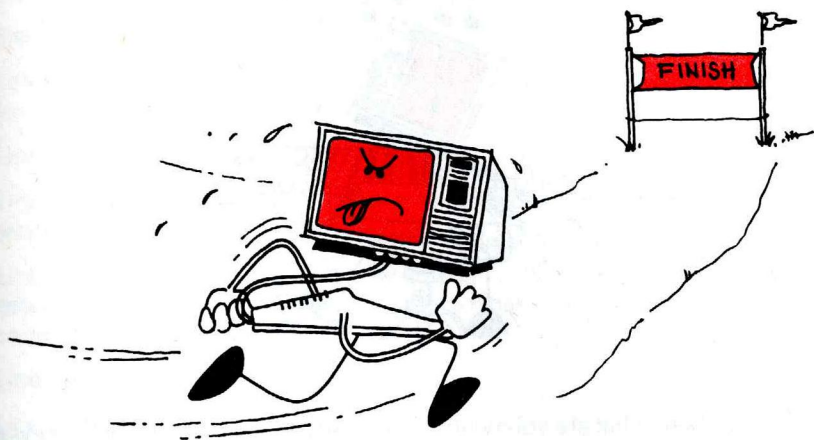
10 A$ = INKEY$
20 IF A$ <>" " GOTO 50
30 PRINT "YOU PRESSED NOTHING"
40 GOTO 10
50 PRINT "THE KEY YOU PRESSED IS----" A$

```

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 (" ").

Remember that < > means "not equal to."

" " is an "empty string" (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.

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.

An Electronic Piano

Look again at "Musical Tones" in the Appendix. It lists these as the tones for middle C through the next higher C:

C - 89	E - 125	G - 147	B - 170
D - 108	F - 133	A - 159	C - 176

Erase memory and type this program:

```
10 A$ = INKEY$
20 IF A$ = "" THEN 10
30 IF A$ = "A" THEN T = 89
40 IF A$ = "S" THEN T = 108
50 IF A$ = "D" THEN T = 125
60 IF A$ = "F" THEN T = 133
70 IF A$ = "G" THEN T = 147
80 IF A$ = "H" THEN T = 159
90 IF A$ = "J" THEN T = 170
100 IF A$ = "K" THEN T = 176
110 IF T = 0 THEN 10
120 SOUND T,5
130 T = 0
140 GOTO 10
```



How would this change the program?

```
120 SOUND T,1
```

Run it. Well, what are you waiting for? Play a tune. Type any of the keys on the third row down on your keyboard—from A to K.

Why will the program not work right if you use INPUT rather than INKEY\$?

Answer: If you use INPUT, the computer waits until you press **ENTER** before it sees what you type. With INKEY\$, it sees everything you type.

There's another way of writing this program using READ and DATA lines. Do you know how?

This is what we came up with:

```
10 A$ = INKEY$
20 FOR X = 1 TO 8
30 READ B$,T
40 IF A$ = B$ THEN SOUND T,5
50 NEXT X
60 RESTORE
70 GOTO 10
80 DATA A, 89, S, 108
90 DATA D, 125, F, 133
100 DATA G, 147, H, 159
110 DATA J, 170, K, 176
```

Beat the Computer

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 have the computer 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—in a split second.

Line 60 adds 1 to T, the timer. T now equals 1. The next time the computer gets to line 60 it again adds 1 to the timer to make T equal 2. Each time the computer runs Line 60 it adds 1 to T.

Line 70's there just to make you nervous.

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 should 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

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
```

Remember the problem of mixing strings with numbers? Chapter 2 will refresh your memory.

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

```
?TM ERROR IN 100
```

That's because you can't make a *string* (A\$) equal to a *number* (X + Y). You somehow must change A\$ to a number.

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," for example, 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

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 = S/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 1 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've 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), S by 60 (to get the minutes). They then divide the eight words by M to get the words per minute.

We could have made this calculation in one line by using parentheses:
$$120 R = 8 / ((T / 74) / 60)$$

How about a variation of this program—a speed-reading test?

Learned in Chapter 11

BASIC WORDS

INKEY\$
VAL

Notes

MORE BASICS

Before you're finished with the "basics," you need to know a few more words.

The first is STOP. Type and run this program:

```
10 A = 1
20 A = A + 1
30 STOP
40 A = A * 2
50 STOP
60 GOTO 20
```

The computer starts running the program. When it gets to Line 30, it prints:

```
BREAK IN 30
OK
```

You now can type a command line to see what's happening. For example, type:

```
PRINT A (ENTER)
```

The computer prints 2—A's value when the program's at Line 30. Now type:

```
CONT (ENTER)
```

The computer continues the program. When it gets to Line 50, it prints:

```
BREAK IN 50
```

Type:

```
PRINT A (ENTER)
```

This time the computer prints 4—A's value at Line 50.

Type CONT again, and the computer breaks again at Line 30. If you have it again print A, it prints 5—the value of A at Line 30 the second time through the program.

Inserting STOP lines in your program helps you figure out why it's not working the way you expect. When you fix the program, take the STOP lines out.

For Long Programs . . .

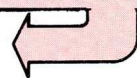
Clear memory and type:

```
PRINT MEM (ENTER)
```

The computer prints how much storage space remains in the computer's memory.

When you're typing a long program, you will want to have the computer PRINT MEM from time to time to make sure you're not running out of memory.

To save memory, you can omit spaces in your program before and after punctuation marks, operators, and BASIC words.



Help with Typing


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.

ON . . . GOSUB in Line 20 works the same as three lines:

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

ON . . . GOSUB looks 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? Since there's no 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

```

Notice the word ELSE in Lines 1020, 2020, 3020, and 4020. You can use ELSE if you want the computer to do something special when the condition is not true. In Line 1020, if your answer—A—equals $X + Y$, then the computer prints CORRECT or else it prints WRONG.

You may use ON . . . GOTO in a similar way as ON . . . GOSUB. The only difference is that ON GOTO sends the computer to another line number rather than to a subroutine.

Here's part of a program using ON . . . GOTO:

```

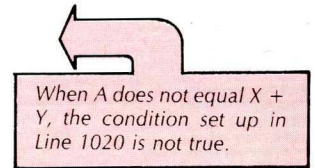
10 CLS
20 PRINT @ 134, "(1) CRAZY EIGHTS"
30 PRINT @ 166, "(2) 500"
40 PRINT @ 198, "(3) HEARTS"
50 PRINT @ 354, "WHICH DO YOU WANT TO PLAY"
60 INPUT A
65 CLS
70 ON A GOTO 1000, 2000, 3000

1000 PRINT @ 230, "CRAZY EIGHTS GAME"
1010 END

2000 PRINT @ 236, "500 GAME"
2010 END

3000 PRINT @ 235, "HEARTS GAME"
3010 END

```



Does the Job Say "AND" or "OR"?

Anyone who speaks English knows the difference between "and" and "or"—even your computer. For example, assume there's a programming job opening. The job requires:

A degree in programming
AND
Experience in programming

Erase memory and type:

```

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
HAVE THE JOB" ELSE PRINT "SORRY, WE CAN'T
HIRE YOU"
50 GOTO 10

```

Run the program. You may answer the questions this way:

```
DO YOU HAVE--  
A DEGREE IN PROGRAMMING? NO  
EXPERIENCE IN PROGRAMMING? YES  
SORRY, WE CAN'T HIRE YOU
```

Now, assume the requirements change so that "or" becomes "and." The job now requires:

```
A degree in programming  
OR  
Experience in programming
```

To make this change in the program type:

```
40 IF D$ = "YES" OR E$ = "YES" THEN PRINT  
"YOU'VE GOT THE JOB" ELSE PRINT "SORRY, WE  
CAN'T HIRE YOU"
```

Run the program and see what a difference AND and OR makes:

```
DO YOU HAVE--  
A DEGREE IN PROGRAMMING? NO  
EXPERIENCE IN PROGRAMMING? YES  
YOU HAVE THE JOB
```

More Arithmetic

These words can save many program lines:

SGN

SGN tells you whether a number is positive, negative, or zero:

```
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
```

Run the program, inputting these numbers:

```
15 -30 -.012 0 .22
```

ABS

ABS tells you the absolute value of a number (the magnitude of the number without respect to its sign). Type:

```
10 INPUT "TYPE A NUMBER"; N  
20 PRINT "ABSOLUTE VALUE IS" ABS(N)  
30 GOTO 10
```

Run the program inputting the same numbers as the ones above.

STR\$

STR\$ converts a number to a string. Example:

```
10 INPUT "TYPE A NUMBER"; N  
20 A$ = STR$(N)  
30 PRINT A$ + " IS NOW A STRING"
```

Exponents

Type and run this program to see how the computer deals with very large numbers:

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

The computer prints very large or very small numbers in "exponential notation." "One billion" (1,000,000,000), for example, becomes $1E+09$, which means "the number 1 followed by nine zeros."

If an answer comes out "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).

Exponential 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.

Notice the OV (overflow) error at the end. The computer can't handle numbers larger than $1E+38$ or smaller than $-1E+38$. (It rounds off numbers around $1E-38$ and $-1E-38$ to 0.)

Or technically 1×10^9 , which is 1 times ten to the ninth power: $1 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10$

In our BASIC, that's 5/10/10/10/10/10/10

Congratulations, Programmer!

You've now learned the "basics" and can no doubt write some decent programs. The rest of the book is extra—to help expand and refine your skills.

If you want to keep learning, skip to any of these sections:

- To improve your graphics programs, read Section II, "Drawing Pictures."
- To write programs that handle large volumes of information, read Section III, "Getting Down to Business."
- To call "machine-language programs" from BASIC and, using machine language, create high-resolution graphics, read Section IV, "Bits and Bytes." You need to already understand machine language to use this section.

Learned in Chapter 12

BASIC WORDS

STOP SGN
CONT ABS
MEM STR\$

BASIC SYMBOLS

AND
OR

BASIC CONCEPT

Exponential
notation

Notes

1950	1951
1952	1953
1954	1955
1956	1957
1958	1959
1960	1961
1962	1963
1964	1965
1966	1967
1968	1969
1970	1971
1972	1973
1974	1975
1976	1977
1978	1979
1980	1981
1982	1983
1984	1985
1986	1987
1988	1989
1990	1991
1992	1993
1994	1995
1996	1997
1998	1999
2000	2001
2002	2003
2004	2005
2006	2007
2008	2009
2010	2011
2012	2013
2014	2015
2016	2017
2018	2019
2020	2021
2022	2023
2024	2025

SECTION II

DRAWING PICTURES

This section shows how to write colorful and exciting programs. Here, you'll put pictures on your screen that move, dance, and even sing songs.

COLOR THE SCREEN

Having fun? If so, you're sure to enjoy the subject of this chapter: computer graphics.

Since graphic ideas will come to you quickly—and your programs may end up long—this chapter just shows how to start. While running this chapter's programs, you may want to stop and improve or rewrite them. We hope you do. That's a fast way to learn.

Start by making the screen black. Type:

```
10 CLS(0)
```

Add these two lines and run the program:

```
20 SET(0,0,3)
30 GOTO 30
```

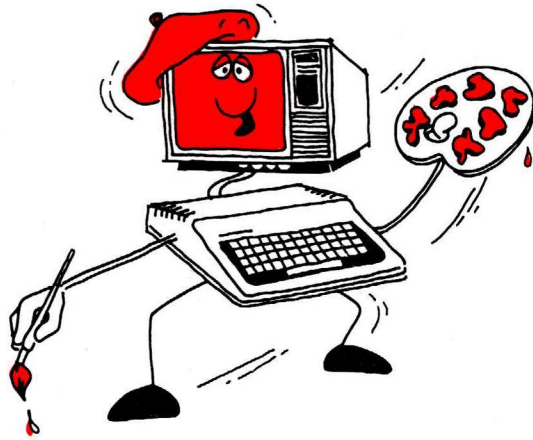
See the blue dot? It's at the screen's top left-hand corner. To put the dot at the bottom right-hand corner, change Line 20 and run the program:

```
20 SET(63,31,3)
```

Want to center the dot? Use this for Line 20:

```
20 SET(31,14,3)
```

Be sure to type Line 30. We'll explain why later.



SET tells the computer to set a dot on your screen at the position you choose:

- With the first number, you choose the dot's horizontal position (a number from 0 to 63).
- With the second number, you choose the dot's vertical position (a number from 0 and 31).

In the Appendix, there's a screen grid, "Graphics Screen Locations." It divides your screen into the 64 (0 to 63) horizontal positions and 32 (0 to 31) vertical positions. Use this grid to position dots on your screen.

What about 3, the third number? Try replacing 3 with other numbers. Type each of these lines and run the program:

```
20 SET (31,14,4)
20 SET (31,14,1)
```

Have you decided what this number does? When you use 4, the dot's red; with 1, it's green. The color codes for SET are the same as those for CLS (codes 0-8). They're listed in your *Quick Reference Card*.

Now see why the GOTO line is necessary. Delete Line 30 and run the program:

```
10 CLS (0)
20 SET (31,14,1)
```

Although you can't see it, a dot is set. But when the program ends, the computer prints OK on top of the dot.

The GOTO line sets an infinite loop in the program so that it will never end (that is, unless you press **BREAK**).

The screen positions for SET are different than those for PRINT (@). That's why there are two grids in the Appendix. Be sure to use "Graphics Screen Locations" for SET.

Setting Two Dots

To set more than one dot, you need to do more planning. To find out why, run a few programs. First, run this:

```
10 CLS (0)
20 SET (32,14,3)
30 SET (33,14,3)
40 GOTO 40
```

You should now have two blue dots—side by side—in the middle of your screen. Change the color of the right dot so you'll have one blue and one red dot. Type:

```
30 SET (33,14,4)
```

Run the program again. This time, *both dots are red*.

Look again at the "Graphics Screen Locations" grid. Note 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

The above program asks the computer to set two different-colored dots (red and blue) within the same block. Since the computer can't set them in different colors, it sets them both in the second color—red.

Type this and run the program:

```
30 SET (34,14,4)
```

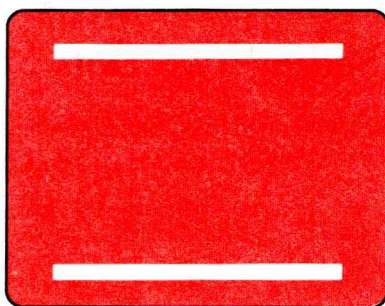
Since the dot in position 34, 14 is in a different block, the computer can set the two dots in different colors.

The Computer's Face

Using dots, you can draw whatever you want. We'll draw a simple picture of a computer. First draw the top and the bottom of the head. We'll make it buff. Type:

```
5  CLS(0)
10  FOR H = 15 TO 48
20  SET (H,5,5)
30  SET (H,20,5)
40  NEXT H
50  GOTO 50
```

Run the program. This is what you see on your screen. (The lines should be buff rather than white, like we have them.)



Lines 10 and 40 set up a FOR/NEXT loop for H, making the horizontal positions 15 through 48 for the top and the bottom lines.

Line 20 sets the top line. The horizontal position is 15 through 48, and the vertical position is 5.

Line 30 sets the bottom line. The horizontal position, again, is 15 through 48, and the vertical position is 20.

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
90  GOTO 90
```

and run the program.

We'll make the nose orange. Type:

```
90  SET (32,13,8)
```

And the mouth red. Type:

```
100  FOR H = 28 TO 36
110  SET (H,16,4)
120  NEXT H
```

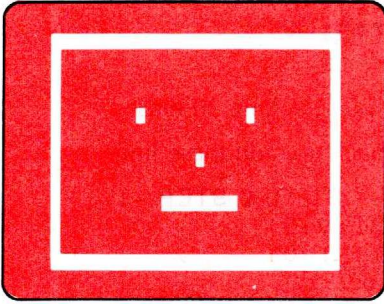


Notice we've changed Line 50—the GOTO line.

And the eyes blue. Type:

```
130 SET(25,10,3)
140 SET(38,10,3)
150 GOTO 150
```

Run the program. This is what your screen should look like now:



A Blinking Computer

With a few more lines, you can make the computer "blink." 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 the horizontal position 38 and the vertical position 10. That's the right eye.

To make the eye blink, simply set and reset it by adding this line:

```
160 GOTO 140
```

List your program to see if it's the same as ours.

```
5 CLS(0)
10 FOR H = 15 TO 48
20 SET(H,5,5)
30 SET(H,20,5)
40 NEXT H
50 FOR V = 5 TO 20
60 SET(15,V,5)
70 SET(48,V,5)
80 NEXT V
90 SET(32,13,8)
100 FOR H = 28 TO 36
110 SET(H,16,4)
120 NEXT H
130 SET(25,10,3)
140 SET(38,10,3)
150 RESET(38,10)
160 GOTO 140
```

Run and improve it (if you can).

You don't need to tell the computer the color of the dot to reset (erase) it.

The Bouncing Dot

You may now see how to program pictures that move. This program makes a ball move down:

Remember always to erase your program before typing a "new" one.

```
5  CLS(0)
10 FOR V = 0 TO 31
20  SET(31,V,3)
30  RESET(31,V)
40  NEXT V
```

Each dot that Line 20 sets, Line 30 erases.

To move the ball back up, add these lines:

```
50  FOR V = 31 TO 0 STEP -1
60  SET(31,V,3)
70  RESET(31,V)
80  NEXT V
```

Add this line to make the ball "bounce":

```
90  GOTO 10
```

Run the program. To slow the dot down (it'll look better), change Lines 30 and 70:

```
30  IF V > 0 THEN RESET(31,V-1)
70  IF V < 31 THEN RESET(31,V+1)
```

The > sign means the same as it does in math—greater than. The < sign means less than.

If You Have 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.

Now run this short program to see how joysticks work:

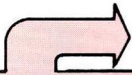
```
10  CLS
20  PRINT @ 0, JOYSTK(0);
30  PRINT @ 5, JOYSTK(1);
40  PRINT @ 10, JOYSTK(2);
50  PRINT @ 15, JOYSTK(3);
60  GOTO 20
```

See the four numbers on your screen? They're the horizontal and vertical positions of the two joysticks' "floating switches."

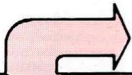
Grasp the right joystick's floating switch. (The joystick connected to the RIGHT JOYSTICK jack on the back of the computer.) Keeping it in the center, move it from left to right. The first number on the screen changes: to numbers from 0 and 63.

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



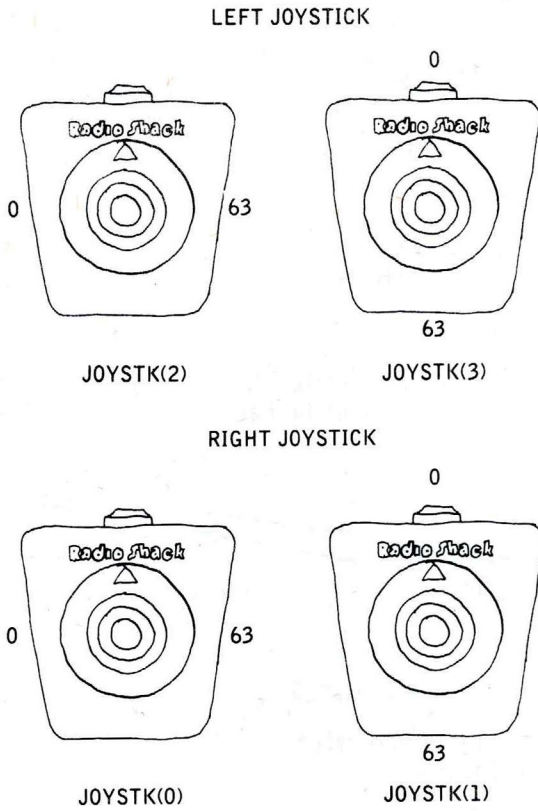
Be sure to type the semicolons at the ends of Lines 20, 30, 40, and 50.



The second or fourth number may change also, but not from 0 to 63.

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's not printing JOYSTK(0)'s coordinates, it's still reading them. Because of this, it's able to read the other joystick coordinates. Whenever you want to read JOYSTK(1), JOYSTK(2), or JOYSTK(3), you first need to read JOYSTK(0).

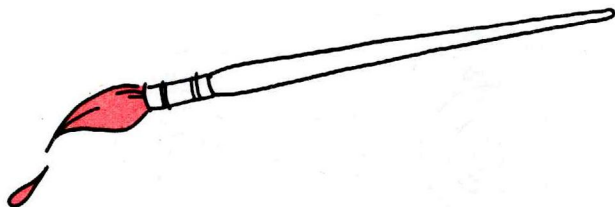
Painting with Joysticks

> = means greater than or equal to

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 from 0 to 63.

Line 30 reads V—its vertical position. This can also be a number from 0 to 63. Since the highest vertical position on your screen is 31, Line 40 is necessary. It makes V always equal a number from 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 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 3; the dots it sets are red. Move it to the left, and the computer makes C equal to 4; the dots it sets are blue.

Want to use your joystick buttons? Add these lines to the program:

```
100 P = PEEK(65280)
110 PRINT P
120 GOTO 100
```

Now type:

```
RUN 100 ENTER
```


This tells the computer to run the program starting at Line 100. Your computer should be printing either 255 or 127 over and over.

PEEK tells the computer to look at a certain spot in its memory to see what number's there. Line 100 looks at the number in position 65280. As long as you're not pressing either of the buttons, this spot contains the number 255 or 127.

Press the right button. When you press it, this memory location contains either the number 126 or 254.

Press the left button. This makes this memory location contain either the number 125 or 253.

Using this information, you can make the computer do whatever you want when you press one of the buttons. We'll make it go back to Line 10 and CLS(0) (clear the screen to black) when you press the right button. Change Lines 110 and 120:

```
110 IF P = 126 THEN 10
120 IF P = 254 THEN 10
```

Delete Line 90 and add this line:

```
130 GOTO 20
```

Run the program and start "painting." Press the right button when you want to clear the screen and start again.

If you press the buttons when you're not running the program, you'll see @ ABCDEFG or HIJKLMNO.

Some of the joysticks will not read six "blocks" in each of the four corners of your screen.

Learned in Chapter 13

BASIC WORDS

SET
RESET
JOYSTK
PEEK

Notes

GAMES OF MOTION

Ready for video games? Because of a word named POINT, you can program almost any kind of motion game.

Type these lines. They set orange dots at random horizontal and vertical positions:

```

5  CLS(0)
10  FOR X = 1 TO 5
20  SET(RND(64)-1, RND(30) + 1, 8)
30  NEXT X

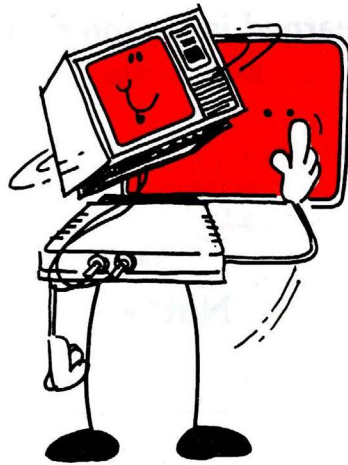
```

Add these lines and run the program:

```

40  FOR V = 2 TO 31
50  FOR H = 0 TO 63
60  IF POINT(H,V) <> 0 THEN GOSUB 100
70  NEXT H
75  NEXT V
80  END
100  PRINT @ 0, "LOCATION" H ", " V " IS SET"
110  RETURN

```



Line 60 checks each "point" in vertical positions 2-31 and horizontal positions 0-63.

- If the point equals 0, it's "off" (black).
- If the point equals some other number, it's "set." Line 100 prints the point's position.

You can also find out what color each point is. Erase memory, and then type and run this program:

```

5  CLS(0)
10  C = RND(9) - 1
20  SET(31,15,C)
30  IF POINT(31,15) = 2 THEN PRINT @ 0,
    "LOCATION 31,15 IS YELLOW";

```

```

40 IF POINT(31,15) = 3 THEN PRINT @ 480,
   "LOCATION 31,15 IS BLUE";
50 FOR T = 1 TO 1000: NEXT T
60 GOTO 5

```

If the point is "set," it equals one of the 8 color code numbers listed in Appendix B.

Plotting Through Asteroids

This game uses the right joystick, so be sure it's connected. (If you don't have joysticks, skip to the next chapter.)

Erase memory and type these lines. They create "asteroids."

```

5 CLS (0)
10 FOR X = 1 TO 200
20 SET (RND(64) - 1, RND(30) + 1, 8)
30 NEXT X

```

Type these lines to create a "planet."

```

40 FOR H = 54 TO 63
50 FOR V = 28 TO 31
60 SET(H,V,3)
70 NEXT V,H

```

The above lines set blue dots in each of these positions: horizontal 54-63 and vertical 28-31. Note that Line 70 contains two instructions: NEXT V and NEXT H.

To read the right joystick's position, type:

```

100 A = JOYSTK(0)
110 B = JOYSTK(1)
120 B = B/2
130 B = INT(B)

```

A reads the horizontal coordinates (0-63), and B reads the vertical coordinates (0-63). Since the highest vertical position on your screen is 31, Lines 120 and 130 are necessary.

To set the entire block surrounding the joystick's position, add these lines:

```

200 IF INT(A/2) <> A/2 THEN A = A - 1
210 IF INT(B/2) <> B/2 THEN B = B - 1
220 FOR H = A TO A + 1
230 FOR V = B TO B + 1
240 SET(H,V,6)
250 NEXT V,H
999 GOTO 100

```

Lines 200 and 210 ensure that the first horizontal and vertical dots set are even numbers, and Lines 220-250 set the entire block.

Run the program. Move your joystick around. The cyan colored line moves wherever you position the joystick.

Now turn this into a game. Type these lines and run the program:

```

212 FOR H = A TO A + 1
214 FOR V = B TO B + 1

```

```

216 IF POINT(H,V) = 8 THEN SOUND 128,1:
    T = T + 1
218 NEXT V,H

```

Each time you hit an orange point, Line 216:

- Sounds a tone
- Adds 1 to T, a counter

Add these lines and run the program:

```

235 IF POINT(H,V) = 3 THEN PRINT @ 0,
    "CONGRATULATIONS - YOU MADE IT": END
300 PRINT @ 28, T
310 IF T > 10 THEN 1000
1000 FOR X = 1 TO 40
1010 CLS(RND(8))
1020 SOUND RND(255), 1
1030 NEXT X
1040 PRINT @ 228, "YOUR SPACESHIP EXPLODED"

```

Want the rules printed on the screen? Add some more lines:

```

80 FOR X = 1 TO 8
82 READ A$
84 PRINT @ 0, A$
86 FOR Y = 1 TO 1500: NEXT Y
88 NEXT X
90 R$ = INKEY$: IF R$ = "" THEN 90
92 FOR H = 4 TO 63
94 SET(H,0,8): SET(H,1,8)
96 NEXT H

2000 DATA YOUR GOAL IS TO PLOT A COURSE
2010 DATA TO GUIDE YOUR SPACESHIP
2020 DATA THROUGH THE ASTEROIDS
2030 DATA TO THE BLUE PLANET
2040 DATA HIT MORE THAN 10 ASTEROIDS
2050 DATA AND YOUR SPACESHIP EXPLODES!!!
2060 DATA PRESS ANY KEY WHEN YOUR SPACE-
2070 DATA SHIP IS AT TOP LEFT CORNER

```

Learned in Chapter 14

BASIC WORD

POINT

Notes

THE TALKING-COMPUTER TEACHER

Who says the computer can't talk? Its voice, though, sounds strangely like your own. You can program the computer to "talk" using your own taped voice. This adds interest and fun to any program.

This chapter requires that you have a tape recorder.

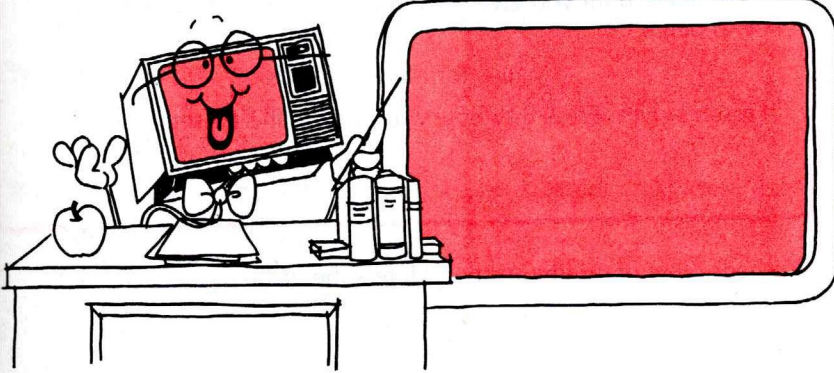
- Unplug the three-pronged cable connecting your tape recorder to the computer.
- Put in a tape, rewind it, press the PLAY and RECORD buttons, and talk into the microphone. (Plug in a microphone if your recorder doesn't have one built in.) Say whatever you want.

Now type this program:

```
5  CLS
10  INPUT "PRESS <ENTER> TO HEAR THE
    RECORDING" ; A$
20  MOTOR ON
30  AUDIO ON
```

Even if you don't have a microphone, you can try this program using a tape of music or one of your program tapes.

Aa Bb Cc Dd Ee Ff Gg Hh Ii Jj Kk Ll Mm Nn Oo Pp Qq



Before running the program, prepare the recorder:

- Rewind the tape you've just recorded.
- Connect the recorder to the computer (as shown in your introduction manual).
- Press the recorder's PLAY button.
- Turn up your T.V.'s volume.

Run the program. You'll hear your voice over the T.V.

MOTOR ON turns on your cassette recorder. AUDIO ON connects your recorder's sound to the T.V. speaker.

There's a way of programming your tape recorder to stop, but for now simply press RESET. It's on the back right-hand side of your keyboard (when you're facing it). List your program. It's still intact.

Add these lines:

```
35 CLS
40 A$ = INKEY$
50 PRINT @ 255, "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 labels whatever key you are pressing or not pressing as A\$. When you press X, the recorder's audio connection and motor are turned off.

Now you can record the "talking-computer teacher." Here's the script:

SCRIPT

"Hi, I'm your talking-computer teacher. The first lesson is math. I'll give you a series of addition problems. Press the 'W' key —"

(pause for a few seconds)

"You'll hear that every time you give me a wrong answer. Press the 'R' key —"

(pause for a few seconds)

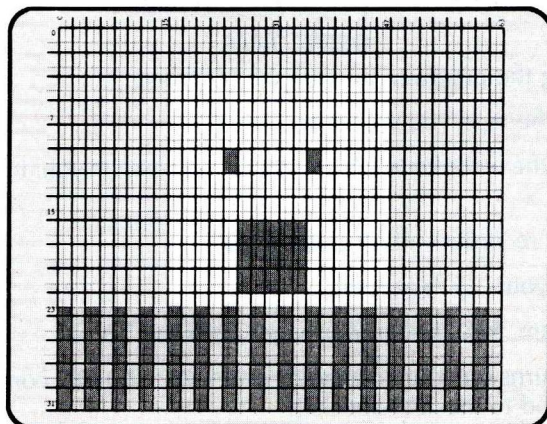
"That's how I'll reward you when you answer correctly. I won't talk to you again until you give me three correct answers. Press the 'G' key to begin."

(pause for a few seconds)

"Lesson's over. Press the 'E' key to turn off the cassette."

This program is a little long, but we think you'll enjoy it. If you want, you can go on to the next chapter and come back to this later.

Finished? Now draw the teacher. Here's the grid:



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:

```
100 FOR H = 16 TO 47
110 FOR V = 4 TO 23
120 SET(H,V,5)
130 NEXT V,H
```

Remember, you can always press RESET to stop your recorder when it is connected to the computer.

The body:

```
140 FOR H = 0 TO 63 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
```

The eyes:

```
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"
```

Want to make the eyes blink? Type:

```
505 IF RND(4) = 4 THEN SET(24,10,5):
    SET(37,10,5)
```

Run the program. That's the teacher. To get it to talk, add these lines:

```
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 89 STEP -10
2010 SOUND T,1
2020 NEXT T
2030 RETURN

3000 FOR T = 89 TO 176 STEP 10
3010 SOUND T,1
3020 NEXT T
3030 RETURN
```

Before running the program, prepare your tape for playing. (Rewind the tape, connect the recorder to the computer, and press PLAY.) Then run the program.

Do what your voice tells you. When you press W, you should hear descending tones; R gives you ascending tones. G just ends the program. That's because you haven't typed the arithmetic routine yet.

Change Line 430 and add Line 460:

```
430 IF A$ = "G" THEN MOTOR OFF: GOSUB 1000
460 IF A$ = "E" THEN MOTOR OFF: END
```

Then add the arithmetic routine:

```
1000 X = RND(100): Y = RND(100)
1010 PRINT @ 0, "WHAT IS" X "+" Y
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
```

Notice Line 1015. It sets the PRINT position for what you type in Line 1020.

Rewind the tape and press PLAY. Run the program.



The talking-computer teacher. Perfect for making arithmetic fun.

Learned in Chapter 15

BASIC WORDS

MOTOR
AUDIO

FASTER GRAPHICS

Up to now, you've used only one method to draw pictures on your screen. Using SET is easy, but slow and tedious. This chapter shows a faster method to use—graphic character codes.

Character Codes

Type:

```
PRINT ASC("A") (ENTER)
```

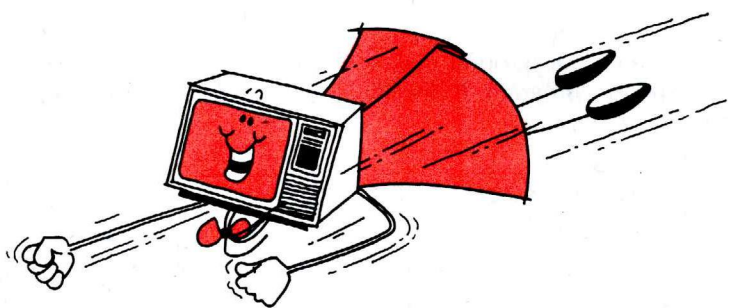
The computer displays 65—the "ASCII" code for the character A. Type:

```
PRINT CHR$(65) (ENTER)
```

The computer displays A—the character represented by the ASCII code number 65.

Look at the list of "ASCII Character Codes" in your *Quick Reference Card*. Each keyboard character has a code. Test some other characters.

"ASCII" stands for the American Standard Code for Information Interchange. By using these standard codes, your computer can communicate with other computers.




Note that even the "nondisplayable" characters—such as \leftarrow —have a code. Erase memory, and type this program:

```
10 CLS(0)
20 H = 63
25 SET(H,14,3)
30 A$ = INKEY$
40 IF A$ = CHR$(8) THEN G0
50 GOTO 30
60 H = H - 1
65 IF H < 0 THEN END
70 SET(H,14,3)
75 RESET(H + 1,14)
80 GOTO 30
```


RUN the program. Press the \leftarrow character. Each time you press it, it backspaces the blue dot.

Lines 30 and 40 check to see if you're pressing the \leftarrow key (Code 8).

Need to review INKEY\$? See Chapter 11.

If you are pressing , Lines 60 and 70 “backspace” H, the horizontal coordinate, and set a blue dot. Line 75 then resets (blacks out) the previously set blue dot.


DO-IT-YOURSELF PROGRAM 16-1

Write some more lines to the program so that you can press  to move the dot forward.


Graphic Character Codes


The ASCII codes in your *Quick Reference Card* represent only about half the Color Computer’s ASCII codes. The other half of the codes—Codes 128-255—are for graphic characters.


Type:

```
PRINT CHR$(128) 
```

The computer displays a black block. Try other graphic codes:

```
PRINT CHR$(129) 
```

```
PRINT CHR$(130) 
```

```
PRINT CHR$(131) 
```

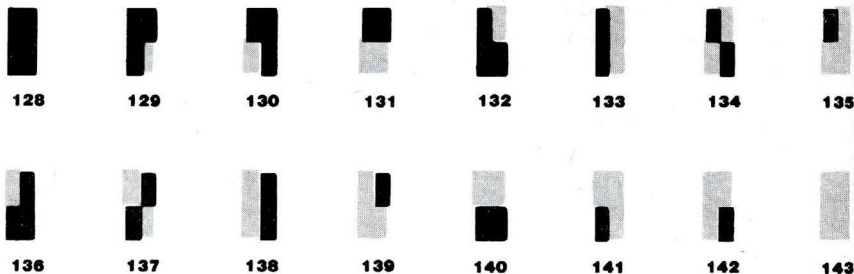
The computer displays three blocks with different combinations of green and black.

Since the green background makes it difficult to see the outline of the blocks, type this program. It displays the first block against a buff background:

```
10 CLS(5)
20 PRINT @ 239, CHR$(129);
30 GOTO 30
```

A grid of “PRINT @ Screen Locations” is in the Appendix. (We explained how to use it in Chapter 7.) Be sure to type the semicolon.

Look at “Graphics Screen Locations” in the Appendix. As we explained earlier, the darker lines divide the grid into blocks. Each block contains 4 dots. These 4 dots can be arranged in 16 ways to form these graphic characters:



To display all 16 graphic characters, type and run this program:

```
10 CLS(5)
20 FOR C = 128 TO 143
30 PRINT @ 0, "PRESS ANY KEY TO CONTINUE";
40 PRINT @ 173, C;
50 PRINT @ 240, CHR$(C);
60 K$ = INKEY$ : IF K$ = "" THEN G0
70 NEXT C
80 GOTO 10
```

Line 50 displays the graphic characters for Codes 128-143 at Position 240 on your screen.

Try something a little different. Type:

```
PRINT CHR$(129 + 16) (ENTER)
```

The computer displays the graphic character for 129, except the area that should be green is yellow. Type:

```
PRINT CHR$(129 + 32) (ENTER)
PRINT CHR$(129 + 48) (ENTER)
PRINT CHR$(129 + 64) (ENTER)
```

These are the numbers you can add to the 16 graphic codes above to create different colors:

0—green	64—buff
16—yellow	80—cyan
32—blue	96—magenta
48—red	112—orange

To see all the graphic characters in each color, add these lines and run the program:

```
15 FOR X = 0 TO 7
17 IF X = 1 THEN CLS(1)
40 PRINT @ 170, C "+" X * 16;
50 PRINT @ 240, CHR$(C + X * 16);
75 NEXT X
```

PROGRAMMING EXERCISE

Write 3 lines to create the characters below. Make the first buff; the second, magenta; and the third, blue:

Answers:

```
PRINT CHR$(133 + 64)
PRINT CHR$(137 + 96)
PRINT CHR$(140 + 32)
```

Graphic Strings

BASIC treats graphic characters the same as any other characters: as strings. You can combine and store graphic characters the same way you combine and store strings.

Know why it's important to type a semicolon at the end of these PRINT @ lines? Try it with and without the semicolon.

The semicolon makes the computer stop as soon as it displays your characters. Otherwise, it continues to display its customary green background for the rest of the line.

Notice these numbers are all multiples of 16. ($16 = 16 \times 1$; $32 = 16 \times 2$; $48 = 16 \times 3$. . . $112 = 16 \times 7$).

If you prefer, you can use the formula on your Quick Reference Card. It gives the same results.

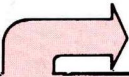
Erase memory and form two graphic strings. Type:

```
10 A$ = CHR$(129 + 32) + CHR$(131 + 32)
20 B$ = CHR$(133 + 112) + CHR$(143 + 112) +
   CHR$(130 + 112)
```

You can position these "strings" on your screen in the same way you position any other strings: with PRINT @. Add these lines and run the program:

```
30 CLS(0)
40 PRINT @ 237, A$;
50 PRINT @ 241, B$;
60 GOTO 60
```

The computer displays what looks like a blue car and an orange truck at the center of your screen.



Note the difference: You "print" graphic characters using "PRINT @ Screen Locations" (Appendix B). You "set" dots using Graphic Screen Locations (Appendix C).

DO-IT-YOURSELF PROGRAM 16-2

Using graphic characters, write a program to create this image in the center of your screen. Make the chairs yellow and the table orange.

Learned in Chapter 16

BASIC WORDS

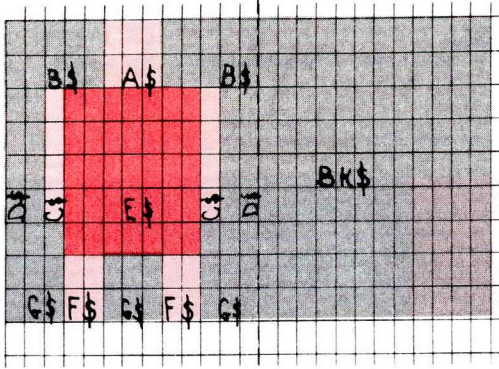
ASCII
CHR\$

BASIC CONCEPT

graphic characters

LET'S DANCE

This chapter lets you catch your breath, have some fun, and, at the same time, review what you've learned. You'll create a "dancing computer" that looks, at rest, like this:

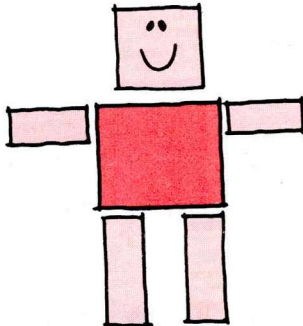


Start by typing this line to reserve plenty of string space:

```
1 CLEAR 1000
```

Then add these lines to form the black strings (D\$, G\$, B\$, and BK\$), the buff strings (C\$, F\$, and A\$), and the red string (E\$):

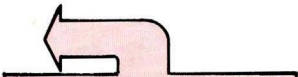
```
10 D$ = CHR$(128) + CHR$(128)
20 G$ = D$ + CHR$(128)
30 B$ = G$ + D$
40 BK$ = B$ + B$ + B$ + D$ + D$
50 C$ = CHR$(143 + 64)
60 F$ = C$ + C$
70 A$ = F$ + C$
80 FOR X = 1 TO 7
90 E$ = E$ + CHR$(143 + 48)
100 NEXT X
```



Run the program. Then display all the strings you've formed. For example, to display BK\$, type:

```
PRINT BK$ <ENTER>
```

On your screen, the light green will be buff; the dark green, red; and the gray area, black.



B\$ is actually 5 characters long. On your screen it will line up with the word PRINT.

D\$ is 2 characters; G\$ is 3; BK\$ is 19; A\$ is 3.

C\$ is 1 character long; F\$ is 2; E\$ is 7.

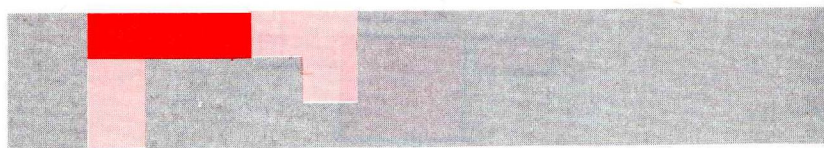
Now combine the above strings to form the head (HD\$):



The body (BD\$):



And three leg positions: L1\$, L2\$, and L3\$:



To do this, add these lines:

```
110 HD$ = B$ + A$ + B$ + BK$ + B$ + A$ + B$ + BK$
120 FOR X = 1 TO 4
130 BD$ = BD$ + D$ + C$ + E$ + C$ + D$ + BK$
140 NEXT X
150 L1$ = G$ + E$ + G$ + BK$ + G$ + F$ + G$ + F$ +
  G$ + BK$ + G$ + F$ + G$ + F$ + G$
160 H$ = G$ + G$
170 I$ = H$ + D$
180 L2$ = G$ + E$ + A$ + BK$ + G$ + F$ + H$ + F$ +
  BK$ + G$ + F$
190 L3$ = A$ + E$ + G$ + BK$ + F$ + H$ + F$ + G$ +
  BK$ + I$ + F$
```

Run the program. Then display the five new strings you've formed.

Now add these lines:

```
500 INPUT "LOCATION (0-243)"; L
510 INPUT "POSITION (1-3)"; P
520 GOSUB 1000
530 GOTO 500

1000 CLS(0)
1010 PRINT @ L, HD$ + BD$;
1020 ON P GOSUB 2000, 3000, 4000
1030 PRINT @ L + 32 * 6, LG$; : RETURN

2000 LG$ = L1$ : RETURN
3000 LG$ = L2$ : RETURN
4000 LG$ = L3$ : RETURN
```

Run the program. The computer shows you each location and position you request.

Line 1010 prints the head and the body at the location you requested.

Line 1020 sends the program to a subroutine that makes LG\$ equal to L1\$, L2\$, or L3\$ (depending on whether you typed 1, 2, or 3 for the position). Line 1030 then displays LG\$ directly under the head and body (6 columns below your requested location).

To make the computer dance, change Lines 500 and 510 and add these lines:

```
500 FOR X = 1 TO 17
510 IF X = 1 OR X = 5 THEN RESTORE
5 INPUT "SPEED (1-10)"; S
515 READ L, P, T, D
525 SOUND T, S * D
527 NEXT X
5000 DATA 137, 2, 89, 1, 240, 1, 133, 2
5010 DATA 137, 3, 159, 1, 229, 1, 133, 2
5020 DATA 5, 1, 89, 1, 229, 1, 133, 2
5030 DATA 5, 1, 147, 1, 229, 1, 159, 1
5040 DATA 229, 1, 147, 1, 5, 1, 133, 1
5050 DATA 229, 1, 125, 2, 5, 1, 133, 1
5060 DATA 229, 1, 147, 2
```

Examples: To display the head, type PRINT HD\$
ENTER. To display the body, type PRINT BD\$
ENTER.

Remember READ and DATA from Chapter 13?

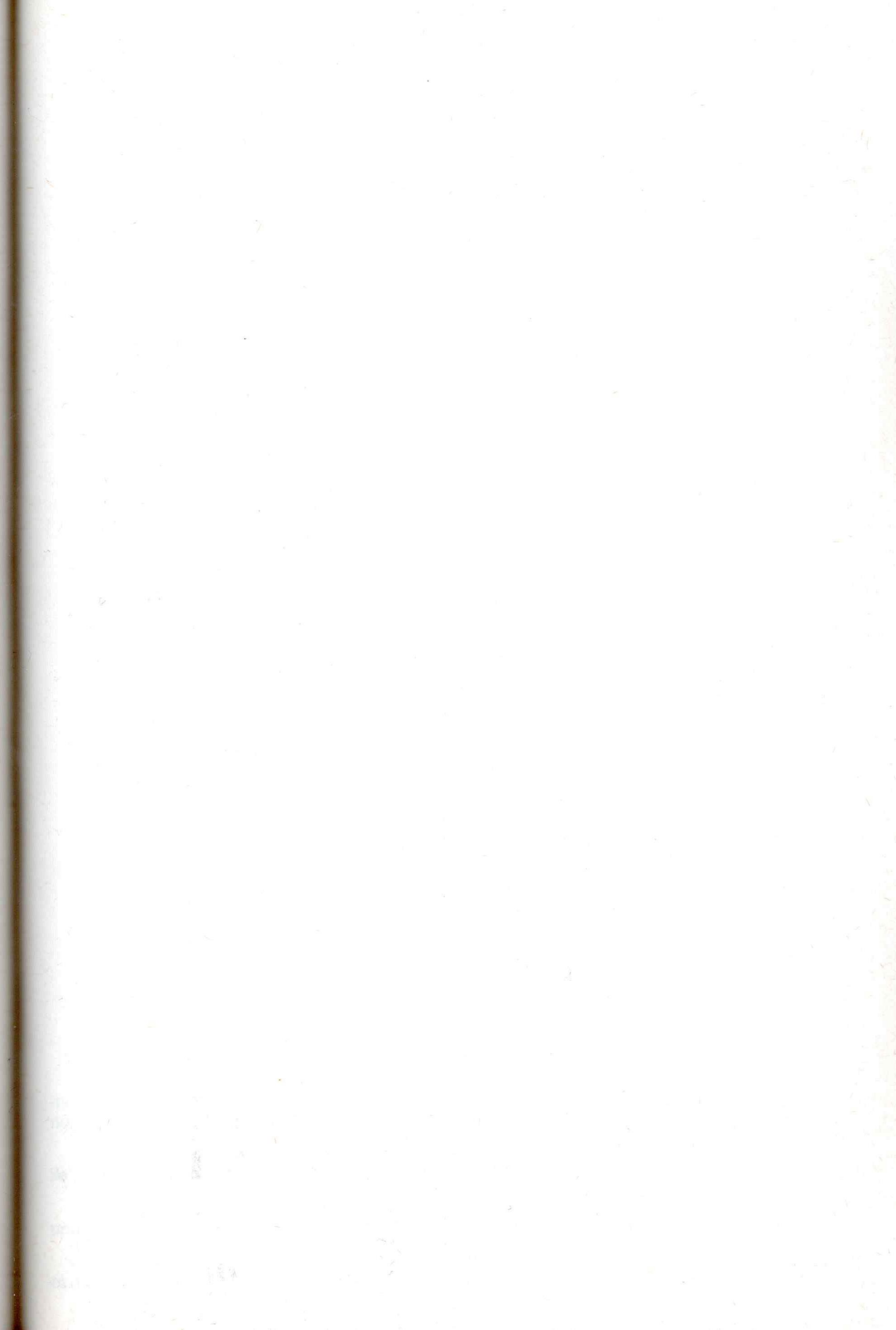
Run the program and watch the dance. Line 515 reads Lines 5000-5060 to determine each screen location, leg position, tone, and tone duration.

For example, at first, the "dancing computer" appears at Screen Location 137 with Leg Position 2. It sounds Tone 89 for Tone Duration S*1.

Next the dancing computer appears at Screen Location 240 with Leg Position 1. It sounds Tone 133 for Tone Duration S*2.

If you're still with us, you no doubt have many of your own ideas. If you plan to do much graphic programming, you may want to consider upgrading to Extended Color BASIC.





SECTION III

GETTING DOWN TO BUSINESS

This section deals with information you want to manage. For example, you may want to manage:

- Checkbook receipts
- Shopping items
- Tax records
- Inventory
- Addresses
- Records, books, or tape collections

In this section, you'll learn how to store, update, sort, and analyze information to fit your own needs.

THE
LIBRARY OF THE
MUSEUM OF MODERN ART

1000 5th Ave. New York, N.Y. 10028

TAPING

Your first and foremost task is to store your information permanently on cassette tape. This, of course, requires a tape recorder.

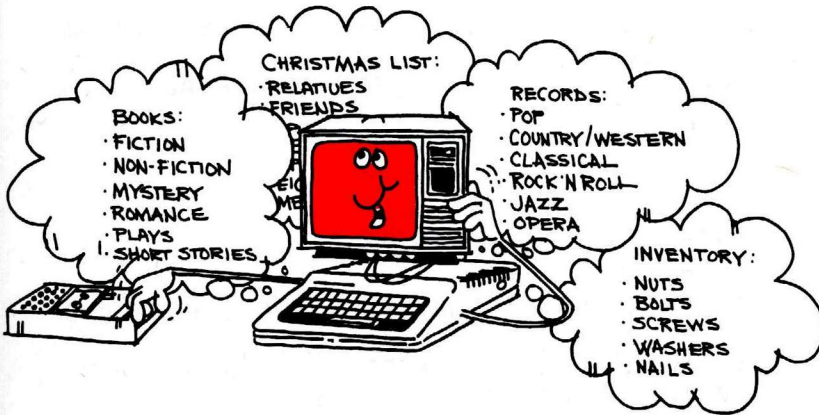
Ready to get organized? We'll start with your book collection. Here's a small list of books:

1. WORKING
2. CAT'S CRADLE
3. SMALL IS BEAUTIFUL
4. STEPPENWOLF

If you've read your introduction manual, you know how to save BASIC programs on tape. To save *information*, you need a program that follows these steps:

STEPS FOR STORING INFORMATION ON TAPE

1. *Open* communication to the tape recorder so that you can *output* (send out) information to a *file*.
2. *Output* all information to the tape recorder file.
3. *Close* communication to the tape recorder.



Start the program with this line:

```
10 OPEN "0", #-1, "BOOKS"
```

This "opens" communication to the tape recorder ("device #-1") so that you can "output" ("O") information. Whatever information you output, the computer stores on tape in a "file" named BOOKS.

A "file" is a collection of information—such as book titles—stored under one name.

Now output the information. Type:

```
15 CLS: PRINT "INPUT YOUR BOOKS--TYPE <XX>
    WHEN FINISHED"
20 INPUT "TITLE"; T$
30 PRINT #-1, T$
40 GOTO 15
```

Line 20 "prints" (outputs) your book titles—not to the screen, but to device #-1, the tape recorder.

Then close communications. Type:

```
25 IF T$ = "XX" THEN 50
50 CLOSE #-1
```

The computer then closes communication to the tape recorder.

Add three more lines to the program:

```
1 CLS
2 PRINT "POSITION TAPE - PRESS PLAY AND
    RECORD"
4 INPUT "PRESS <ENTER> WHEN READY"; R$
```

The program should now look like this:

```
1 CLS
2 PRINT "POSITION TAPE--PRESS PLAY AND
    RECORD"
4 INPUT "PRESS <ENTER> WHEN READY"; R$
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
30 PRINT #-1, T$
40 GOTO 15
50 CLOSE #-1
```

opens Communication with recorder →

Prints title to recorder →

Closes Communication with recorder →

Prepare the recorder.

- Connect the recorder. Your computer's introduction manual shows how.
- Position a tape in the recorder, and, if necessary, rewind the tape so you'll have room for recording. (If you're using a non-Radio Shack tape, position it past the starting leader.)
- Press the recorder's RECORD and PLAY buttons so that they are both down.

Then run the program. As soon as you press **ENTER**, the cassette motor turns on: The computer is opening a "file" on tape and naming it BOOKS.

The program then asks for titles. Type:

```
TITLE? WORKING
TITLE? CAT'S CRADLE
TITLE? SMALL IS BEAUTIFUL
TITLE? STEPPENWOLF
TITLE? XX
```

The computer clears the screen after each title.

Each time you input a title, the computer prints it in a special place in memory reserved for the tape recorder. When you finish, the tape recorder motor turns on: The computer is printing all the titles to the recorder (Line 30) and then closing communication with the recorder (Line 50).

Your book titles are now all saved on tape in a file named BOOKS. To read them back into memory, use just about the same steps.

STEPS FOR INPUTTING INFORMATION FROM TAPE

1. *Open* communication to a tape recorder so that you can *input* information from a *file*.
2. Check to see if you're at the *end of the file*.
3. *Input* information from the tape recorder file.
4. Repeat Steps 2 and 3 until you reach the end of the file.
5. *Close* communication to the tape recorder.

To open communication, type:

```
60 CLS: PRINT "REWIND THE RECORDER AND
   PRESS PLAY"
70 INPUT "PRESS <ENTER> WHEN READY"; R$
80 OPEN "I", #-1, "BOOKS"
```

This opens communication to the tape recorder—this time, to input information from the BOOKS file.

To input information, add these lines:

```
90 INPUT #-1, B$
100 PRINT B$
```

Line 90 inputs the first book title (B\$) from the BOOKS file stored on tape. (The variable name you choose makes no difference.) Line 100 displays this title on your screen.

To check for the end of the file and close the file, add these lines:

```
85 IF EOF (-1) THEN 120
110 GOTO 85
120 CLOSE #-1
```

Are you wondering what the -1 means? EOF returns a -1 when you reach the end of the file.

Line 85 says if you are at the end of this file (in this case, the BOOKS file), go to 120 and close communication with the tape recorder.

Note that EOF(-1) comes *before* the INPUT #-1 line. If it's *after* INPUT #-1, you'll get an IE error—"input past the end of the file."

List this last part of the program by typing LIST 60 - **ENTER**. It should look like this:

```
60 CLS: PRINT "REWIND THE RECORDER AND
PRESS PLAY"
70 INPUT "PRESS <ENTER> WHEN READY"; R$
80 OPEN "I", #-1, "BOOKS" ← Opens Communication
85 IF EOF (-1) THEN 120 ← with recorder
90 INPUT #-1, B$ ← Inputs title from recorder
100 PRINT B$
110 GOTO 85
120 CLOSE #-1 ← closes communication with recorder
```

Be sure to press only the PLAY button, Not RECORD Also, be sure to rewind the tape.

Now run this part of the program. Type:

```
RUN 60 ENTER
```

If your computer becomes "hung up" communicating with the tape recorder, you can regain control by pressing the RESET button. It's on the back right-hand side of your keyboard. Then look for missing or mistyped lines in your program.

When you press **ENTER**, the recorder's motor comes on while the computer inputs items from tape. When finished, it displays the four items on your screen.

An Electronic Card Catalog

Assume you need to change the program so it can also store the books' authors and subjects:

TITLE	AUTHOR	SUBJECT
<i>Working</i>	Studs Terkel	Sociology
<i>Cat's Cradle</i>	Kurt Vonnegut	Fiction
<i>Small Is Beautiful</i>	E. F. Schumacher	Economics
<i>Steppenwolf</i>	Hermann Hesse	Fiction

Start by changing the "output" part of the program (the first half). Type these lines:

```
26 INPUT "AUTHOR"; A$
28 INPUT "SUBJECT: S$
29 IF A$ = "XX" OR S$ = "XX" THEN 50
30 PRINT #-1, T$, A$, S$
```

Then change the "input" part of the program. Type these lines:

```
90 INPUT #-1, B$, A$, S$
100 PRINT "TITLE : " B$
102 PRINT "AUTHOR : " A$
104 PRINT "SUBJECT : " S$
```

Now take advantage of this organization. For example, have the program print a book list on any given subject. Add these lines:

```
130 CLS
140 INPUT "WHICH SUBJECT"; C$
150 PRINT "REWIND THE TAPE - PRESS PLAY"
160 INPUT "PRESS <ENTER> WHEN READY"; E$
170 CLS: PRINT C$ " BOOKS" : PRINT
180 OPEN "I", #-1, "BOOKS"
190 IF EOF (-1) THEN 230
200 INPUT #-1, B$, A$, S$
210 IF S$ = C$ THEN PRINT B$, A$
220 GOTO 190
230 CLOSE #-1
```

Run the input part of the program by typing RUN 130 **(ENTER)**. If you choose "fiction," this happens:

```
WHICH SUBJECT? FICTION
REWIND THE TAPE - PRESS PLAY
PRESS <ENTER> WHEN READY

FICTION BOOKS:

CAT'S CRADLE      KURT VONNEGUT
STEPPENWOLF      HERMANN HESSE
```

DO-IT-YOURSELF PROGRAM 18-1

Assume you have these checks:

NO.	DATE	PAYABLE TO	ACCOUNT	AMOUNT
101	5/13	Safeway	food	\$52.60
102	5/13	Amoco	car	32.70
103	5/14	Joe's Cafe	food	10.32
104	5/17	American Airlines	vacation	97.50
105	5/19	Holiday Inn	vacation	72.30

Write a program that outputs all the checks to tape. Then have it input them from tape so that you can type one account—such as food—and the computer will tell you the total amount you've spent on food.

See "Sample Programs" in the Appendix for examples of how to store data on tape.

Learned in Chapter 18

BASIC WORDS

OPEN
CLOSE
PRINT #-1
INPUT #-1
EOF

BASIC CONCEPT

data files

Notes

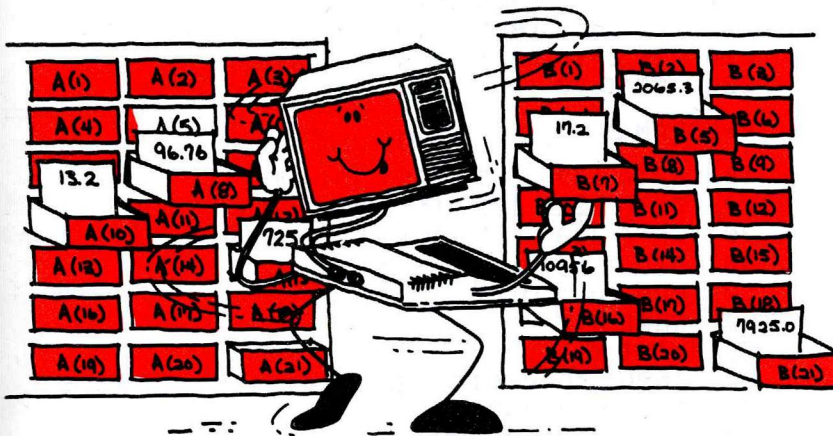
MANAGING NUMBERS

Have you tried to write programs to handle much information? If so, you'll be glad to know Color BASIC has an easy-to-manage way to keep track of information.

Assume, for example, you want to write a program that lets you manage this information:

ELECTION RESULTS

District	Votes for Candidate A
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



Up to now, you've used variables to store information in memory. For example, to store the votes of the first three districts, type:

```
A = 143 (ENTER)
B = 215 (ENTER)
C = 125 (ENTER)
```

But there's a better kind of variable you can use. Type:

```
A(1) = 143 (ENTER)
A(2) = 215 (ENTER)
A(3) = 125 (ENTER)
```

Each of the above variables has a "subscript"—(1), (2), and (3). Other than how they use the subscript, these variables work the same as any other variables. To see for yourself, type both of these lines:

```
PRINT A; B; C <ENTER>
PRINT A(1); A(2); A(3) <ENTER>
```

Now 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.

Actually, this leaves room for 15 subscripted items when you count 0 as a subscript.

Enter and run Program 2. Here's how it works:

Line 40 reserves space for a list of information—called an "array" named A—with 14 subscripted items.

- 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(10) → 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 input a subscript, and Line 90 prints the item you requested.

Now that you've stored information in an array, it's easy to manage it. For instance, you can add these lines, which let you change the information:

```

92 INPUT "DO YOU WANT TO ADD TO THIS" ; R$
94 IF R$ = "NO" THEN B0
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 you can add these lines to display the information:

```

72 INPUT "DO YOU WANT TO SEE ALL THE TOTALS" ;
S$
74 IF S$ = "YES" THEN GOSUB 110
100 GOTO 72
110 PRINT "DISTRICT", "VOTES"
120 FOR X = 1 TO 14
130 PRINT X, A(X)
140 NEXT X
150 RETURN

```

The name of the array is A. The X or Z in parentheses refers to the subscript of one of the items.

A Second Array

Assume you also want to keep track of a second candidate's votes—Candidate B:

ELECTION RESULTS

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

To do this, 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, 114, 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) - saves room
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

```

data for array A (lines 10-60)
data for array B (lines 40-60)
saves room (line 70)
reads array A data (lines 80-100)
reads array B data (lines 110-130)

DO-IT-YOURSELF PROGRAM 19-1

Write an inventory program that keeps track of 12 items (numbered 1-12) and the quantity you have of each item.

Deal the Cards

To keep track of 52 "cards," you need to use an array. Erase your program and 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 of the cards are the same.

To make 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 above lines build Array T and put all 52 cards in it: T(1) = 1, T(2) = 2, T(3) = 3 ... T(52) = 52.

Then 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

```

This program is a little tough. Skip it and come back to it later if it's slowing you down too much.

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 have been dealt. You can do this by building another array—Array D. Add these lines, which 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 19-2

Add lines to the program so that it displays only your "hand"—the first 5 cards dealt.

Learned in Chapter 19

BASIC WORD

DIM

BASIC CONCEPT

arrays

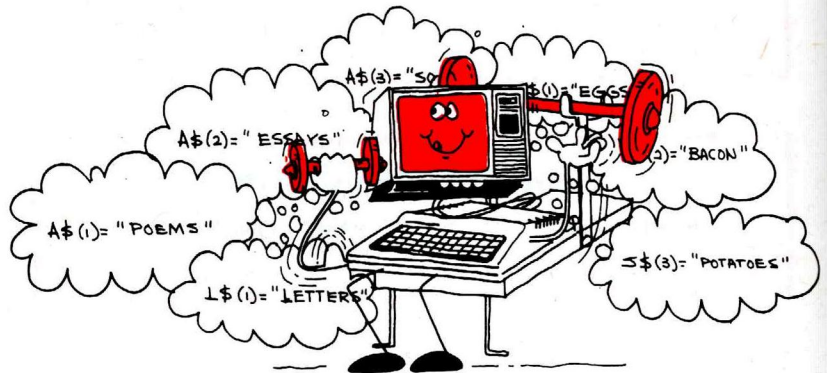
Notes

MANAGING WORDS

In the last chapter, you used arrays to manage numbers. Here, you'll use arrays to manage words by editing, updating, and printing an entire essay.

Start with a simple list of words: a shopping list:

- | | |
|-------------|-------------|
| 1. EGGS | 7. TOMATOES |
| 2. BACON | 8. BREAD |
| 3. POTATOES | 9. MILK |
| 4. SALT | 10. CHEESE |
| 5. SUGAR | 11. FISH |
| 6. LETTUCE | 12. JUICE |



The dollar sign's the only difference between these subscripted variables and the ones in the last chapter.

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
```

Handwritten red annotations:
 A bracket on line 50 points to the right and says "reads array S\$".
 A bracket on line 90 points to the right and says "print array S\$".

DO-IT-YOURSELF PROGRAM 20-1

Add some lines to the above program so that you can change any item on this list.

DO-IT YOURSELF PROGRAM 20-2

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 that you can revise any line.

Want to compose music? Look up "Music Composer" in the "Sample Programs" appendix.

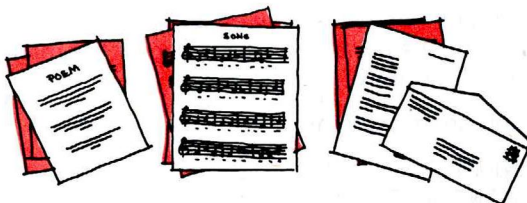
Writing an Essay (... A Novel, Term Paper ...)

Now that you've learned how to use string arrays, it will be easy to write a program that stores and edits what you type. Type this program:

```
1 CLEAR 1000
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
150 PRINT A$(Y);
160 NEXT Y
```

Haven't heard of word processing? It's a kind of program that lets you type and store information, make changes to it, and print it out on demand.

Need a refresher on some of this? CLEAR is in Chapter 8 and INKEY\$ is in Chapter 11.



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:

Line 1 clears plenty of string space.

Line 5 saves room for an array named A\$ that may have up to 50 sentences.

Line 30 makes X equal to 1. X will be used to label all the sentences.

Line 40 checks to see which key you are pressing. If it is nothing (" "), Line 50 sends the computer back to Line 40.

Line 60 prints the key you pressed.

Line 70 sends the computer to the lines that print your paragraph when you press the " " key.

Line 80 builds a string and labels it with number X. X is equal to 1 until you press a period (.). Then Line 80 makes X equal to X + 1.

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 that 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).

Lines 140–160 print your paragraph.

DO-IT-YOURSELF CHALLENGER PROGRAM 20-3

Here's a tough one (but it can be done!) for those intrigued with word processing. Change the above program so that you can:

1. Print any sentence
2. Revise any sentence

You may need to review the challenger program in Chapter 12. Our answer's in the back.

Using the Printer

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, "on-line," 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.

Press the **(SHIFT)** and **(0)** (zero) keys simultaneously and release them so that the letters you type appear in reversed colors on your screen (green with a black background). You are now in an upper-lowercase mode. The reversed colored letters are actually lowercase (noncapitalized) letters.

To type a capital letter, use the **(SHIFT)** key as you do with a typewriter. It appears in regular colors.

Run the program, using the **(SHIFT)** key so that the word RUN is capitalized. Input a sentence with both upper- and lowercase letters. Type:

MY PRINTER PRINTS LOWERCASE LETTERS **(ENTER)**

Having trouble getting into this mode? Read the end of Chapter 1.

All the letters in RUN should appear in regular (not reversed) colors.

DO-IT-YOURSELF PROGRAM 20-4

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 the screen.

Learned in Chapter 20

BASIC WORDS

LLIST
PRINT # - 2

BASIC CONCEPT

string arrays

Notes

SORTING

Any file clerk knows it's easier to find information that's sorted alphabetically. Type this program and run it, until you're convinced the computer can alphabetize:

```

10 INPUT "TYPE TWO WORDS"; A$, B$
20 IF A$ < B$ THEN PRINT A$ " COMES BEFORE " B$
30 IF A$ > B$ THEN PRINT A$ " COMES AFTER " B$
40 IF A$ = B$ THEN PRINT "BOTH WORDS ARE THE
   SAME"
50 GOTO 10

```

With strings, the greater than (>), less than (<), and equal (=) signs have a new meaning. They tell which of two strings comes before the other in alphabetical sequence:

< precedes alphabetically
 <= precedes or is the same alphabetically
 > follows alphabetically
 >= follows or is the same alphabetically
 = is the same



Since the computer can alphabetize, it's easy to write a sorting program. Type and run this program, which sorts 5 words:

```

10 DIM A$(5)
20 FOR I = 1 TO 5
30 INPUT "TYPE A WORD"; A$(I)
40 NEXT I
50 X = 0
60 X = X + 1
70 IF X > 5 THEN GOTO 70
80 IF A$(X) = "ZZ" THEN 60
90 FOR Y = 1 TO 5
100 IF A$(Y) < A$(X) THEN X = Y
110 NEXT Y
120 PRINT A$(X)
130 A$(X) = "ZZ"
140 GOTO 50

```

You can easily make the computer alphabetize more words by changing the 5 to say, 100, in Lines 10, 20, 70, and 90.

To see how the program works, delete Line 120 and add the following lines. (These lines only show what the program does—they have nothing to do with sorting.)

```

120
5 CLS
45 CLS
85 V = V + 1
105 PRINT @ 15+32*(V-1), A$(X)
135 GOSUB 500
500 FOR I = 1 TO 5
510 PRINT @ 0+32*(I-1), A$(I); " ";
520 NEXT I
530 RETURN

```

Run the program. Too fast? Type this line. It slows down the program so you can see what's happening:

```
107 FOR T = 1 TO 600: NEXT T
```

Now run the program again. Input these words and watch carefully:

```

MICHAEL
TRAVIS
DYLAN
ALEXIA
SUSAN

```

Look at Column 2. See how the first name changes from Michael to Dylan to Alexia. Next, notice what happens to Alexia in the first column. Alexia becomes ZZ.

This illustrates how the program sorts the first and second words:

FIRST WORD

MICHAEL	MICHAEL	MICHAEL	MICHAEL	MICHAEL	MICHAEL
TRAVIS		TRAVIS		TRAVIS	
DYLAN		DYLAN		DYLAN	
ALEXIA		ALEXIA		ALEXIA	
SUSAN		SUSAN		SUSAN	
MICHAEL	DYLAN	MICHAEL	ALEXIA	MICHAEL	ALEXIA
TRAVIS		TRAVIS		TRAVIS	
DYLAN		DYLAN		DYLAN	
ALEXIA		ALEXIA		ZZ	
SUSAN		SUSAN		SUSAN	

SECOND WORD

MICHAEL	ALEXIA	MICHAEL	ALEXIA	MICHAEL	ALEXIA
TRAVIS	MICHAEL	TRAVIS	MICHAEL	TRAVIS	MICHAEL
DYLAN		DYLAN		DYLAN	
ZZ		ZZ		ZZ	
SUSAN		SUSAN		SUSAN	
MICHAEL	ALEXIA	MICHAEL	ALEXIA	MICHAEL	ALEXIA
TRAVIS	DYLAN	TRAVIS	DYLAN	TRAVIS	DYLAN
DYLAN		DYLAN		ZZ	
ZZ		ZZ		ZZ	
SUSAN		SUSAN		SUSAN	

Here's how the program works:

Lines 50 and 60 set X's value. At the start, X is 1.

Then Lines 90–110 compare A\$(X)—Michael—with every other name in Array A\$ until a word is reached that precedes Michael—Dylan.

Line 100 then makes A\$(X) equal to Dylan's place in the array: A\$(3). When Dylan is compared with the fourth word—Alexia—A\$(X) becomes A\$(4).

When all the words have been compared with one another, Line 120 displays the first sorted word: Alexia. Line 130 changes Alexia's position—A\$(4)—to ZZ.

At this point, Lines 50 and 60 make X equal 1 again. A\$(X)—Michael—is compared with other names in the array to find the second sorted word.

When Michael's place in the array becomes ZZ, Line 60 sets X to 2. Then, A\$(X)—which is now Travis—is compared with all the names in the array to find the next sorted word.

When the array's values are all changed to ZZ, Line 70 ends the program.

DO-IT-YOURSELF PROGRAM 21-1

Using this sort routine, change the program from the last chapter so that it alphabetizes your books by title, author, or subject.

This chapter shows a simple way to sort. If you need to sort many items, you may want to research faster sorting methods (such as the bubble sort).

Learned in Chapter 21

BASIC SYMBOLS

>
<
=

ANALYZING

If you have more than 4K RAM, you have an easy way to analyze information. By giving each item more than one subscript, you can see it through different dimensions.

Take the voting program from Chapter 19. Here's the information. (We're using only the first three districts to make the program simple.)

ELECTION POLL

District	Votes for Candidate 1	Votes for Candidate 2
1	143	678
2	215	514
3	125	430

We're only using three districts to keep it simple.

We're calling them Candidates 1 and 2 this time rather than Candidates A and B.

In Chapter 19, you stored the above "items" (groups of votes) in two *one-dimensional arrays*: Arrays A and B. In this chapter, you'll store them in *one easy-to-manage two-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

```

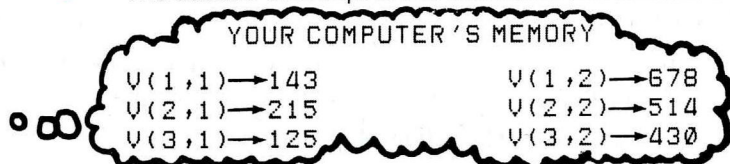
Type and run the program. Notice that each item is labeled by two subscripts.

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, 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).



Remember how to delete lines? 70 **ENTER** Deletes Line 70.

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.

(Delete Lines 70-120 first.)

```
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
```

If you are truly an analytical type, you're going to love the rest of this chapter. If you're definitely NOT that type, skip it!

The Third Dimension

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	157
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, 157, 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 "CANDIDTE 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)	→ 157
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. 2"
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
3080 PRINT @ P, "GROUP" G
3090 FOR D = 1 TO 3
3100 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 22-1

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 22

BASIC CONCEPT

Multidimensional arrays

Notes



SECTION IV

A LITTLE BYTE OF EVERYTHING

This section is for people who want to access the full power of the Color Computer. It assumes some knowledge in machine-language programming.

If you're technical, jump right in! If not, be forewarned. You'll have to be extra careful typing the sample programs. Then triple-check them against our program listings before running them. If your program contains typing errors, you'll probably have to reset the computer to regain control.

The results of your labors will be impressive. Part A shows how to create high-resolution graphics on your screen. Part B shows how to access the Color Computer hardware directly by calling machine-language routines.

HIGH-RESOLUTION GRAPHICS

CONTENTS OF THIS PART

- INTRODUCTION
- SAMPLE PROGRAMS (3)
- A FEW DEFINITIONS
- PREPARING THE COLOR COMPUTER FOR GRAPHICS
- PUTTING GRAPHICS TO WORK
- TABLES:
 1. DESCRIPTION OF THE GRAPHICS MODES AVAILABLE
 2. DISPLAY MODE SELECTION
 3. VIDEO RAM PAGE SECTION
 4. DETAILED DESCRIPTION OF THE GRAPHICS MODES

INTRODUCTION

The Color Computer has many graphics capabilities that you cannot access using the ordinary statements of Color BASIC. However, with the special memory functions PEEK and POKE, you can use and experiment with many of these powerful features. It does take some extra work on your part, but the results can be impressive. In this part we're going to demonstrate how you activate and use these graphics features.

Note: In Extended Color BASIC, many of the graphics capabilities are quite simple to use. That's one of the main attractions of Extended Color BASIC. However, even if you have Extended BASIC, you may find this part interesting. Some of the graphics modes described may only be used via the techniques presented in this part.

First, we'll list two Color BASIC programs that demonstrate how to select and use a graphics mode. The first runs on 4K or 16K RAM systems; the second, on 16K only. We've also included a general-purpose program that you can modify to select any graphics mode (it'll be up to you to put the graphics to use).

After you've tried the programs, you'll be ready for an explanation of how they work. We'll start with a few definitions you'll need. Then we'll go over the steps required to put the computer into any graphics mode. These steps aren't meant to be followed one at a time; they should be put into your BASIC program and then executed in succession.

Finally, we'll suggest a few ways you can put graphics to work.

SAMPLE PROGRAMS

PROGRAM #1: 64 x 64 GRAPHICS MODE FOR 4K OR 16K RAM SYSTEMS

This program makes Color Computer act like a drawing board with a 64 x 64 grid. You may choose between two sets of four colors:

Color#	Set 0	Set 1
0	Green	Buff
1	Yellow	Cyan
2	Blue	Magenta
3	Red	Orange

Type in the program. **Be sure to omit all remarks (lines or a portion of a line beginning with an apostrophe). Also delete all spaces before and after punctuation marks and arithmetic operators (, ; : + - / * > < =).** You must have at least

335 bytes (characters) remaining in memory to run the program. You can check this by having the computer PRINT MEM after the program is typed in. Check the program carefully. Then run it.

After a few seconds, a block appears in the middle of the screen. You may move the block, drawing a line in any of four colors; you may switch color sets; and you may stop the line. Here is a list of the keys that control the drawing board:

Direction of motion:

↑	North (up)
↓	South
←	West
→	East
Q	Northwest
E	Northeast
A	Southwest
S	Southeast
SPACEBAR	Stops motion

Four-Color Set:

1	Color 1
2	Color 2
3	Color 3
0	Color 0 (background color)
/	Change to other four-color set

To return to BASIC's normal text screen, press the RESET button.

PROGRAM #1 LISTING

```

10 'RESERVE 1K
20 CLEAR 10,3071
30 'SET VIDRAM = 3072
40 FOR I = 0 TO 6: READ DT: POKE 65478 + I*2 + DT,
   0: NEXT
50 DATA 0,1,1,0,0,0,0
60 'SELECT VDG MODE G1C
70 FOR I = 0 TO 2: READ DT: POKE 65472 + I*2 + DT,
   0: NEXT
80 DATA 1,0,0
90 'SET UP VIDEO CONTROL REG.
100 POKE 65314, 135
110 'CLEAR VIDRAM
120 FOR I = 3072 TO 4095: POKE I,0: NEXT
130 'BEGIN MAIN PROGRAM
140 'MP( ) IS A LIST OF POWERS OF 4
150 ' TO BE USED BY THE MAPPING FUNCTION
160 DIM MP(3): FOR I = 0 TO 3: READ MP(I): NEXT
170 DATA 1,4,16,64
180 CC = 3: CS = 0 'CC = COLOR, CS = COLOR SET
   SELECT
190 X = 31: Y = 31: XI = 0: YI = 0 'STARTING POINT
   AND INCREMENT
200 'SET UP KEYBOARD TABLE
210 U$ = "↑": D$ = CHR$(10): W$ = CHR$(8): E$ =
   CHR$(9)
220 NW$ = "Q": NE$ = "W": SW$ = "A": SE$ = "S"
230 C0$ = "0": C1$ = "1": C2$ = "2": C3$ = "3"
240 'CHECK FOR KEYBOARD CHARACTER
250 A$ = INKEY$
260 IF A$ = U$ THEN YI = -1: XI = 0: GOTO 400
270 IF A$ = D$ THEN YI = 1: XI = 0: GOTO 400
280 IF A$ = W$ THEN XI = -1: YI = 0: GOTO 400
290 IF A$ = E$ THEN XI = 1: YI = 0: GOTO 400
300 IF A$ = NW$ THEN XI = -1: YI = -1: GOTO 400
310 IF A$ = NE$ THEN XI = 1: YI = -1: GOTO 400
320 IF A$ = SW$ THEN XI = -1: YI = 1: GOTO 400
330 IF A$ = SE$ THEN XI = 1: YI = 1: GOTO 400
340 'CHANGE COLORS IF 0-3 WAS PRESSED

```

```

350 IF C0$ < = A$ AND A$ < = C3$ THEN CC = ASC(A$)
    - 48: GOTO 400
360 'CHANGE COLOR SET IF "/" WAS PRESSED
370 IF A$ = "/" THEN CS = (NOT CS AND 8) OR (CS AND
    NOT 8): POKE 65314,135 + CS: GOTO 400
380 IF A$ = CHR$(32) THEN XI = 0: YI = 0 'STOP
    DRAWING IF <SPC> WAS PRESSED
390 'GET NEW (X,Y) POSITION
400 X = X + XI: Y = Y + YI: IF X < 0 THEN X = 0
410 IF X > 63 THEN X = 63
420 IF Y < 0 THEN Y = 0
430 IF Y > 63 THEN Y = 63
440 ' PLOT THE (X,Y) POINT
450 X1 = INT(X/4): OF = X1 + Y*16: BYTE = 3072 + OF
460 MOD4=INT(X-X1*4):BIT=3-XMOD4
470 X3 = MP(BIT)*CC: X4 = MP(BIT)*3
480 OL = PEEK(BYTE)
490 TE = (255 AND NOT X4) OR ( -256 AND X4): NU =
    (TE AND OL) OR X3
500 POKE BYTE, NU
510 GOTO 230

```

*Should be: 'Typo'
BIT=3-MOD4*

Note for Extended BASIC Users: The 64 x 64 mode is not available in Extended BASIC; however, this program will get it for you. First, however, make these changes in the program:

```

20 CLEAR 10, 15359
30 'SET VIDRAM = 15360
50 DATA 0,1,1,1,1,0,0
120 FOR I = 15360 TO 16383: POKE I, 0: NEXT
450 X1 = INT(X/4): OF = X1 + Y*16: BYTE = 15360 +
    OF

```

PROGRAM #2: 235 x 192 GRAPHICS FOR 16K RAM SYSTEMS

This program shows the highest resolution available on Color Computer. Because it requires 6144 bytes of RAM for the graphics screen, it will not run on a 4K RAM system.

The program draws lines on the screen. You type in (X,Y) coordinates for the starting and ending points, then the program goes into the graphics mode and draws the points. You can then press any key, and the program will ask you for another pair of coordinates.

Type in the program. **BE SURE TO OMIT ALL REMARKS (STATEMENTS BEGINNING WITH AN APOSTROPHE)**. Check the program carefully. Then run it. There will be a one-minute delay before you see the program begin.

If you interrupt the program while it is in the graphics mode, you will need to reset the computer to get back in the normal mode.

PROGRAM LISTING

```

10 'RESERVE 6K
20 CLEAR 10,10239
30 'SET START AND END OF VIDRAM
40 VIDRAM = 10240:VND = 16383
50 PSEL = 65478 'START OF PAGE SELECT REG.
60 VDG = 65472 'START OF VDG REG.
70 VCTRL = 65314 'VIDEO CONTROL REG.
80 'X(0) AND Y(0) WILL BE COORDINATES OF START POINT
90 'M$(0) AND M$(1) WILL BE MESSAGES
100 DIM X(1),Y(1),M$(1)
110 'RH( ) AND VH( ) CONTAIN HI-RES. BIT PATTERN
120 'PA( ) AND VA( ) CONTAIN TEXT BIT PATTERN
130 'TWO( ) CONTAINS A LIST OF POWERS OF 2
140 DIM PH(6),PA(6),VH(2),VA(2),TWO(7)
150 FOR I = 0 TO 6: READ PH(I):NEXT
160 DATA 0,0,1,0,1,0,0
170 FOR I = 0 TO 6: READ PA(I): NEXT

```



```

180 DATA 0,1,0,0,0,0,0
190 FOR I = 0 TO 2: READ VH(I): NEXT
200 DATA 0,1,1
210 FOR I = 0 TO 2: READ VA(I): NEXT
220 DATA 0,0,0
230 READ CH 'HI-RES BIT MASK FOR VID.CTRL, REG.
240 DATA 240
250 READ CA 'TEXT BIT MASK FOR VID.CTRL, REG.
260 DATA 0
270 FOR I = 0 TO 7: READ TWO(I): NEXT
280 DATA 1,2,4,8,16,32,64,128
290 GOSUB 800 'CLEAR OUT VIDRAM
300 'MAIN PROGRAM
310 M$(0) = "FIRST": M$(1) = "SECOND"
320 FOR I = 0 TO 1
330 PRINT "ENTER "; M$(I); " X AND Y"
340 PRINT "0 <= X <= 255, 0 <= Y <= 191"
350 INPUT X(I), Y(I)
360 IF X(I) < 0 OR X(I) > 255 OR Y(I) < 0 OR Y(I) > 191
THEN 340
370 NEXT
380 GOSUB 620 'GO INTO GRAPHICS
390 'DX,DY CONTAIN X,Y DISPLACEMENTS
400 'SX,SY CONTAIN DIRECTION OF THE LINE
410 DX = X(1) - X(0): DY = Y(1) - Y(0): SX = SGN(DX): SY
= SGN(DY)
420 'USE EQUATION Y = SLOPE * X + B
430 'SL = SLOPE OF LINE: B = OFFSET FROM X-AXIS
440 IF DX = 0 THEN 550 'SPECIAL CASE FOR VERTICAL LINES
450 SL = DY/DX: B = Y(0) - SL * X(0)
460 T = SL * SL + 1: GOSUB 930 'GET SQR(T)
470 NX = 1/T1 * SX 'NX IS INCREMENT FOR X
480 FOR XT=X(0) TO X(1) STEP NX
490 X = INT(XT + .5)
500 Y = INT(SL * XT + B + .5)
510 GOSUB 830
520 NEXT
525 A$ = INKEY$: IF A$ = "" THEN 525
530 GOSUB 630 'GO INTO TEXT
540 GOTO 320 'GET NEXT PAIR OF POINTS
550 X = X(0)
560 FOR Y = Y(0) TO Y(1) STEP SY 'DRAW VERTICAL LINE
THRU X(0)
570 GOSUB 830
580 NEXT
585 IF INKEY$="" THEN 585
590 GOSUB 630 : GOTO 320
600 'END OF MAIN PROGRAM
610 'SUBRTNS TO SELECT GBR AND TEXT
620 GOSUB 650 : GOSUB 700 : GOSUB 750 : RETURN
630 GOSUB 670 : GOSUB 720 : GOSUB 770 : RETURN
640 'PAGE-SELECT SUBRTNS
650 FOR I = 0 TO 6: POKE PSEL + I * 2 + PH(I),0: NEXT
660 RETURN
670 FOR I = 0 TO 6: POKE PSEL + I * 2 + PA(I),0: NEXT
680 RETURN
690 'VDG SELECT SUBRTNS
700 FOR I = 0 TO 2: POKE VDG + I * 2 + VH(I),0: NEXT
710 RETURN
720 FOR I = 0 TO 2: POKE VDG + I * 2 + VA(I),0: NEXT
730 RETURN
740 'SUBRTNS TO SET UP VIDEO CONTROL REG.
750 POKE VCTRL, CH OR (PEEK(VCTRL) AND 7)
760 RETURN
770 POKE VCTRL, CA OR (PEEK(VCTRL) AND 7)
780 RETURN
790 'SUBRTN TO CLEAR OUT VIDEO RAM
800 FOR I = VIDRAM TO VND:POKE I,0: NEXT
810 RETURN
820 'MAPPING FUNCTION

```

```

830 X1 = INT(X/8)
840 OF = X1 + Y * 32: BYTE = VIDRAM + OF
850 XMOD8 = INT(X - X1 * 8)
860 BIT = 7 - XMOD8
870 VLU = TWO(BIT)
880 OLD = PEEK(BYTE)
890 MASK = VLU OR OLD
900 POKE BYTE, MASK
910 RETURN
920 'SQR(X) SUBRTN
930 IF T <= 0 THEN T1 = 0: RETURN
940 T1 = T * .5: T2 = 0
950 T3 = (T/T1 - T1) * .5
960 IF (T3 = 0) OR (T3 = T2) THEN RETURN
970 T1 = T1 + T3: T2 = T3: GOTO 950

```

Note: This entire program can be duplicated using the LINE statement of Extended BASIC. However, if you wish to use it for experimentation, it will run without modification under 16K Extended BASIC.

PROGRAM #3: GENERAL-PURPOSE SUBROUTINES

You may use these subroutines to select any graphics mode (subject to the RAM limitations of your computer and the requirements of your main program). You supply the main program to write information onto the graphics screen. You also provide the correct values for Lines 20 and 40.

Later in this section, we provide hints on designing your main program (Putting Graphics to Work).

PROGRAM LISTING

```

10 'RESERVE RAM FOR GRAPHICS
20 'CLEAR STRINGSPACE, MEMEND
30 'SET START AND END OF VIDEO RAM
40 'VIDRAM = MEMEND + 1: VND = 4095 OR 16383
50 PSEL = 65478 'START OF PAGE SELECT REG.
60 VDG = 65472 'START OF VDG REG.
70 VCTRL = 65314 'VIDEO CONTROL REG.
100 DIM X(1), Y(1), M$(1)
110 'PH() AND VH() CONTAIN THE GRAPHICS BIT PATTERN
120 'PA() AND VA() CONTAIN THE NORMAL (TEXT) BIT
    PATTERN
140 DIM PH(6), PA(6), VH(2), VA(2)
150 FOR I = 0 TO 6: READ PH(I): NEXT
160 'DATA X,X,X,X,X,X,X (PAGE-SELECT BIT PATTERN)
170 FOR I = 0 TO 6: READ PA(I): NEXT 'READ NORMAL P-S
    BIT PATTERN
180 DATA 0,1,0,0,0,0,0
190 FOR I = 0 TO 2: READ VH(I): NEXT
200 'DATA X,X,X (GRAPHICS BIT PATTERN FOR VDG)
210 FOR I = 0 TO 2: READ VA(I): NEXT 'NORMAL VDG BIT
    PATTERN
220 DATA 0,0,0
230 READ CH 'GRAPHICS BIT MASK FOR VID.CTRL. REG.
240 'DATA XXX (VIDEO CONTROL VALUE)
250 READ CA 'TEXT BIT MASK FOR VID.CTRL. REG.
260 DATA 0
290 GOSUB 800 'CLEAR OUT VIDRAM
300 '
310 'YOUR MAIN PROGRAM GOES HERE
320 '
599 'END MAIN PROGRAM
600 '
610 'SUBRTNS TO SELECT GRAPHICS AND TEXT
620 GOSUB 650: GOSUB 700: GOSUB 750: RETURN
630 GOSUB 670: GOSUB 720: GOSUB 770: RETURN

```

```

640 'PAGE-SELECT SUBRTNS
650 FORI = 0 TO 6: POKE PSEL + I * 2 + PH(I),0: NEXT
660 RETURN
670 FORI = 0 TO 6: POKE PSEL + I * 2 + PA(I),0: NEXT
680 RETURN
690 'VDG SELECT SUBRTNS
700 FORI = 0 TO 2: POKE VDG + I * 2 + VH(I),0: NEXT
710 RETURN
720 FORI = 0 TO 2: POKE VDG + I * 2 + VA(I),0: NEXT
730 RETURN
740 'SUBRTNS TO SET UP VIDEO CONTROL REG.
750 POKE VCTRL, CH OR (PEEK(VCTRL) AND 7)
760 RETURN
770 POKE VCTRL, CA OR (PEEK(VCTRL) AND 7)
780 RETURN
790 'SUBRTN TO CLEAR OUT VIDEO RAM
800 FORI = VIDRAM TO VND: POKE I,0: NEXT
810 RETURN

```

A FEW DEFINITIONS

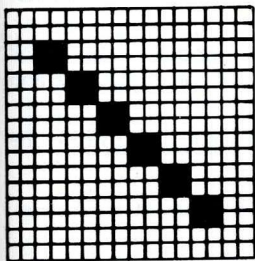
GRAPHICS

Graphics is a video mode of the computer in which you can set or reset blocks or points called "pixels." For each pixel, you may choose from 2, 4, or 8 colors, depending on the particular mode selected. By setting various combinations of pixels, you can generate lines, geometric figures, pictures, and so on.

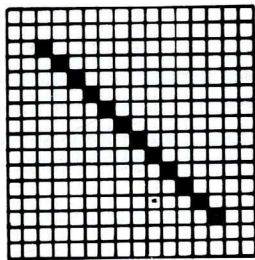
RESOLUTION

The pixel density (how many pixels to a screen) determines the degree of resolution. Depending on the graphics mode, the screen may contain from 2048 (SET/RESET) to 49152 (G6R) pixels. The higher the resolution, the finer the lines and the more detailed the pictures.

To see the importance of resolution, look at these two diagonal lines. The resolution of Line B is 4 times as fine as that of line A.



Line A.
Low Resolution



Line B.
High Resolution

RAM, BYTES, AND BITS

RAM is divided into individually addressed locations called "bytes." The addresses in RAM run from 0 to 4095 or 16383, depending on whether you have a 4K or 16K RAM system. Each address references one byte.

RAM is "Random Access Memory." This is the area where your computer stores programs and data. The computer also uses RAM to store internal values. RAM is erased when you turn off the computer.

One byte consists of 8 on/off switches called "bits." Here is one byte:

Bit # 7 6 5 4 3 2 1 0

Suppose you want to set (set to 1) bit 7 in byte #4000, without changing any of the other bits. You simply OR the current contents of #4000 with the binary value 10000000, which is equivalent to decimal 128:

`NB = PEEK (4000) OR 128`

Since bit 7 is set in the value 128, bit 7 will always be set as a result of the operation. The other bits in the result will be the same as those in address #4000.

VIDEO RAM

When you output to the screen, the information is actually stored in a portion of decimal 0 to 255. (See a math or computer science text for a discussion of binary numbering.)

PEEK AND POKE

These BASIC words let you examine (PEEK) or change (POKE) the contents of memory. Just for review, here is the syntax for each command. The syntax is the way the command should be put together. For an example, with POKE you should first specify the address, then the value.

`PEEK(address)`
`POKE address, value`

PEEK is a function. This means it cannot stand alone in a BASIC program, but must be used in a statement such as:

`OLD = PEEK (BYTE)`

OLD will be given the contents of address BYTE.

POKE can stand alone. It stores the value specified in the address specified.

`POKE BYTE, NU`

The address specified by BYTE will be given the value NU.

BITS AND BOOLEAN ALGEBRA

In the graphics modes, one or two bits may control the color or on/off status of a pixel. So we need a way to control a single bit or pair of bits without affecting other bits.

To change one or two bits in a byte requires a form of computer logic called Boolean algebra. Boolean algebra uses logical operators such as AND, OR, and NOT. These three are available in Color BASIC.

AND and OR compare two values bit-for-bit; NOT takes value and reverses the state of each of its bits. Here are table summaries:

AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

NOT 0 = 1
NOT 1 = 0

Here are some examples of Boolean operations on 1-byte binary values:

`AND 10101010`
`11110000`

`10100000`

`OR 01101110`
`10001000`

`11101110`

`NOT (10101010) = 01010101`

Note:

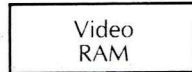
- (1) In this discussion, we refer to the individual bits using the numbers 0 through 7, as shown in the diagram.
- (2) When a bit has a value of 1, we say it is "set"; when it has a value of 0, we say it is "reset." We use these terms in this way throughout this section.

There are 256 possible on/off combinations for a single byte. The combinations are often interpreted as binary numbers ranging from 00000000 to 11111111 or memory. The video display circuitry reads from this "video RAM" in order to generate the screen display.

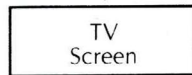
Text goes
into RAM

You type:

PRINT "HERE IS A MESSAGE"



Computer generates
the correct display



Normally, Color BASIC uses the memory area from 1024 to 1535 as video RAM. There are 512 distinct memory locations, or "bytes," in this area, enough to hold 512 alphanumeric characters or 2048 SET/RESET pixels.

You can program the Color Computer to use any area of RAM as "video RAM." This is desirable when:

- A. You want to use high-resolution graphics that require a large video RAM area.
- B. You want to switch back and forth between "pages" of video RAM.

High resolution requires a larger video RAM area than does normal text. For example, in the highest resolution mode, G6R, 6144 bytes of memory are required to store a screenful, or "page," of information.

This increased video RAM requirement has to be taken from the "user area" at the top of memory. This limits the space available to your BASIC program. If you have a 4K RAM machine, you are probably limited to using the G1C and G1R graphics modes, which take only 1024 bytes and leave approximately 1300 bytes for your BASIC program. If you have a 16K RAM machine, you may use the highest resolution mode and still have about 8400 bytes available for your BASIC program.

VIDEO DISPLAY GENERATOR (VDG) REGISTER

This consists of three pairs of addresses in RAM that control the graphics mode. (See Table 1 for a description of the graphics modes available.) These addresses are not actual bytes in RAM, but are direct links to the VDG circuitry in the computer.

DISPLAY CONTROL REGISTER

This is a single memory location that determines which color set is available; it also plays a role in selecting the graphics mode. This address is not an actual byte in RAM, but is a direct link to certain display control circuitry in the computer.

PAGE-SELECT REGISTER

This consists of seven pairs of addresses that determine the start address of video RAM. Using this register, you can start video RAM on any 512-byte boundary in RAM. This address is not an actual byte in RAM, but is a direct link to the page-select circuitry in the Computer.

PREPARING THE COLOR COMPUTER FOR GRAPHICS

1. CHOOSE WHICH GRAPHICS MODE YOU WANT

Using Table 1, decide which graphics mode you want. To do this, ask yourself the following questions:

What is the video RAM requirement? Does your computer have enough RAM to accommodate it? If it does, is there enough room for the program that uses the graphics mode?

How much resolution do you need? How many colors? There is a trade-off between colors and resolution.

For example, G1C and G1R both require 1024 bytes for video RAM, but after that they differ. G1C offers a 64 x 64 pixel density, with 4 colors available for each pixel. Further, you may select between 2 sets of 4 colors. G1R on the other hand, offers a 128 x 64 pixel density, with 2 colors available for each pixel. You may select between 2 sets of 2 colors.

Program #1 uses G1C; Program #2, G6R.

2. SELECT A PAGE OF VIDEO RAM FOR GRAPHICS USE

Color BASIC uses addresses 1024-1535 for video RAM. This is sufficient for alphanumerics and SET/RESET graphics, but not for any of the higher-resolution graphics modes. For these, you should reserve a sufficiently large area at the top of RAM. Use the CLEAR statement to do this.

CLEAR *stringspace*, *memend*

stringspace is the amount of space you'll require for string information. Use the smallest number possible that won't result in an OS error when your program runs.

memend is the highest address Color BASIC will use. You can use addresses above *memend* for your graphics video RAM.

To compute *memend*, use this formula:

$$\text{memend} = \text{memory size} - \text{pagesize}$$

memory size depends on how much RAM is in your system. For 4K systems, it is 4095; for 16K systems, 16383.

pagesize depends on which graphics mode you are going to use. For 4K systems, you are probably limited to G1C or G1R; in either mode, *pagesize* = 1024. For 16K systems, you may use any mode, even one that uses 6144 bytes.

For example, to use G1C in a 4K system, start your program with this statement:

```
CLEAR 20, 3071
```

This assumes you won't need more than 20 bytes for string storage, and it reserves the highest 1024 bytes for use as video RAM.

In Program #1, see Line 20; in Program #2, Line 20.

3. "CLEAR OUT" YOUR VIDEO RAM

You will probably want to start with a clean video screen. To clear out, store zero in each byte of video RAM. For example, in a 4K system, you might use these statements:

```
FOR I = 3072 TO 4095: POKE I,0: NEXT
```

In Program #1, see Line 120; in Program #2, Line 790.

Important Note: Perform Steps 4 and 5, which follow, consecutively, with no pauses in between. Otherwise, the screen will show what is often called "garbage."

4. SWITCH IN YOUR VIDEO RAM

Using the page-select register, tell the Color Computer where your "page" of video RAM starts. A graphics page must start on a 512-byte boundary. To tell Color Computer where the page starts, use a 7-bit value. (The 8 bit, bit 7, is always 0, so is not needed by the page-select register.) Table 3 lists the correct values for pages starting at $memend + 1$ (see Step 3).

Table 3 doesn't list all possible addresses where you might want to start video RAM. The following procedure lets you calculate the correct value for any valid start address for video RAM. (Addresses must be on 512-byte boundaries: 0,512,1024, etc.)

First calculate the video offset in 512-byte "blocks," as follows:

$$OFFSET = VIDRAM / 512$$

VIDRAM is the start address of your video RAM (usually $memend + 1$).

For example, in 4K systems with your video RAM starting at 3072, $OFFSET = 3072 / 512 = 6$.

Then express OFFSET as a 7-bit binary number. For example,

6 decimal =

0	0	0	0	1	1	0
---	---	---	---	---	---	---

 binary

Bit ♦ 6 5 4 3 2 1 0

After finding the correct value, give it to the page-select register.

Remember, this register consists of 7 pairs of addresses. Each pair controls whether a given bit in the page-select circuitry is on or off. To RESET a bit (make it equal to 0), POKE any value into the even-numbered address in the pair; to SET a bit (make it equal to 1), POKE any value into the odd-numbered address in the pair.

BIT #	TO RESET, POKE HERE	TO SET, POKE HERE
0	65478	65479
1	65480	65481
2	65482	65483
3	65484	65485
4	65486	65487
5	65488	65489
6	65490	65491

For example, to switch in the video RAM starting at 3072, give the value 000110 to the page control circuitry as follows:

```
POKE 65478,0 'RESET BIT 0
POKE 65481,0 'SET BIT 1
POKE 65483,0 'SET BIT 2
POKE 65484,0 'RESET BIT 3
POKE 65486,0 'RESET BIT 4
POKE 65488,0 'RESET BIT 5
POKE 65490,0 'RESET BIT 6
```

In Program #1, see Lines 40-50. The formula in Line 40

$$65478 + I * 2 + DT$$

is a shorthand way to poke the appropriate addresses in the page-select register. DT is the 0/1 value for each of the 7 bits.

In Program #2, Lines 640-670 do the same thing using bit patterns stored in PH() and PA().

5. SELECT THE DESIRED GRAPHICS MODE

To select a given graphics mode, you must:

- A. Set the VDG register
- B. Set the control register.

(A) First, look up the 3-bit VDG pattern that selects the graphics mode (see Column 2 in Table 2).

This is the binary value you must give to the VDG register. Remember, this register consists of 3 pairs of addresses. Each pair can be used to control whether a given bit in the VDG circuitry is on or off. To RESET a bit (SET it to zero), POKE any value into the even-numbered address in the pair; to SET a bit, POKE any value into the odd-numbered address in the pair.

BIT #	TO CLEAR, POKE HERE	TO SET, POKE HERE
0	65472	65473
1	65474	65475
2	65476	65477

For example, to select graphics mode G1C, give the value 001 to the VDG registers as follows:

```
POKE 65473,0 'SET BIT 0
POKE 65474,0 'RESET BIT 1
POKE 65476,0 'RESET BIT 2
```

(B) Now select the control value for the graphics mode you want (see Column 3 of Table 2). Then store this value in the control register without changing bits 0-3 of the control register.

For example, to select graphics mode G1C with color set 0.

1. Get temporary result with all bits off except 0, 1, 2. These are not changed.

```
POKE 65314, 128 OR (PEEK(65314) AND 7)
```

2. Turn on bit 7 without changing bits 0, 1, 2.

After you execute Steps 2-5, the computer is in the graphics mode you selected. The screen should be blank. You can devote the rest of your program to using the graphics mode.

In Program #1, see Line 100. In Program #2, see Lines 740-770.

PUTTING GRAPHICS TO WORK

After you select the graphics mode, you can control what appears on the screen by POKEing data into the graphics page you selected. How the data is interpreted depends on the mode you selected. In some modes, 1 byte may control a sequence of 8 bits; in others, 1 byte may control a 2 x 6, 2 x 12, etc., "block."

Table 4 explains how each pixel in a given mode is controlled by a byte or bit. If you're writing your own main program to use the subroutines in Program #3, you may want to experiment, storing various values from 0-255 into a single byte in your page of video RAM.

If you want to get more predictable results, read on . . .

MAPPING FUNCTIONS

In all graphics modes, the screen is divided into (X,Y) coordinates. Each pixel on the screen has a unique (X,Y) "address."

If you've used SET, RESET, and POINT, then you're familiar with this coordinate system. All these statements allow direct reference to (X,Y) coordinates. For example, to set the centerpoint on the screen to blue, use:

```
SET(31,15,3)
```

Using the higher-resolutions graphics modes is a little more difficult. You can't deal directly with (X,Y) coordinates; you must translate, or "map," the desired (X,Y) coordinates onto the appropriate byte of video RAM. When 1 byte controls 2 or more pixels, map the (X,Y) coordinates onto the appropriate bit or bits within a byte.

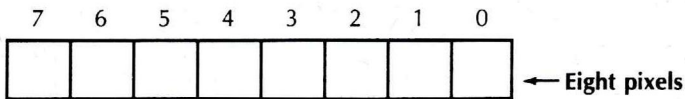
Table 4 shows how each byte of video RAM controls 1 or more pixels.

As an example, we'll take the 256 x 192 mode, G6R.

In this mode, the first 32 bytes of video RAM control the first row of 256 pixels; the second 32 bytes control the second row; etc.

Within each row, each byte of video RAM controls a sequence of 8 pixels:

**One Byte of Video RAM
seen as 8 bits:**



Bit 7 controls the leftmost pixel in the sequence; bit 0, the rightmost.

With this in mind, you can construct a series of BASIC operations to map (X,Y) onto one bit in one byte.

Note: In the following BASIC statements, we assume the following:

- X is the X-coordinate. (For illustration, X = 128.)
- Y is the Y-coordinate. (For illustration, Y = 96.)
- VIDRAM is the first address of video RAM. (For illustration, VIDRAM = 10240.)
- The expression "2 X" means "2 to the X power." (This function is not available in Color BASIC, but you can simulate it with a table of powers of 2.)

1. Which byte "contains" the pixel?

```
OFFSET = INT(X/8) + Y*32 = 16 + 3072 = 3088  
BYTE = VIDRAM + OFFSET = 10240 + 3088 = 13328
```

2. Which bit in BYTE controls the pixel?

```
XMOD8 = X - INT(X/8)*8 = 0  
BIT = 7 - XMOD8 = 7
```

3. What 1-byte value will set the pixel? What 1-byte mask will set the pixel without changing any of the others controlled by the same byte? For illustration, assume BYTE contains 8.

```
VLU = 2 ↑ BIT = 128 = binary 10000000  
OLD = PEEK(BYTE) = 8 = binary 00001000  
MASK = VLU OR OLD = 136 = 10001000  
POKE BYTE, MASK
```

4. What 1-byte value will reset the pixel? What 1-byte mask will reset the pixel without changing any of the others controlled by the same byte? For illustration, assume BYTE contains 136.

```
VLU = 255 - 2 ↑ BIT = 255 - 128 = 127 = binary  
01111111  
OLD = PEEK(BYTE) = binary 10001000 = 136  
MASK = VLU AND OLD = binary 00001000 = 8  
POKE BYTE, MASK
```

The mapping just described is used in Program #2. See Lines 820-910. Another mapping (64 x 64, G1C) is used in Program #1, Lines 440-500.

TABLE 1. DESCRIPTION OF THE GRAPHICS MODES AVAILABLE

Mode (1)	Resolution	Number of Colors (2)	Video RAM Req. (Bytes)
SG6	64 x 48	8	512
SG8	64 x 64	8	2048
SG12	64 x 96	8	3072
SG24	64 x 192	8	6144
G1C	64 x 64	4	1024
G1R	128 x 64	2	1024
G2C	128 x 96	4	2048
G2R	128 x 96	2	1536
G3C	128 x 96	4	3072
G3R	128 x 192	2	3072
G6C	128 x 128	4	6144
G6R	256 x 192	2	6144

128x64?

128x192?

Note:

- (1) The mode names are abbreviations. Read "SG6" as "semigraphics six"; read "G1C" as "graphics one with color"; read "G1R" as "graphics one with resolution"; and so on. In all "semigraphics" modes, you have 8 colors at once. In all "with color" modes, you have 4 colors at once. In all "with resolution" modes, you have 2 colors at once.
- (2) In the 4-color modes, you may select between two sets of 4 colors each. In the 2-color modes, you may select between 2 sets of 2 colors each. The color-set select bit (bit 3 of the video control register) determines which set is used. See Table 2 for more details on selecting the color set.

TABLE 2. DISPLAY MODE SELECTION

Mode	VDG Register Three-Bit Pattern	Video Control Register Value With Color Set* 0 / 1	Data Bits* 7 / 6
SG6	0 0 0	16 / 24	1 / X
SG8	0 1 0	0 / 0	1 / X
SG12	1 0 0	0 / 0	1 / X
SG24	1 1 0	0 / 0	X / X
G1C	0 0 1	128 / 136	X / X
G1R	0 0 1	144 / 152	X / X
G2C	0 1 0	160 / 168	X / X
G2R	0 1 1	176 / 184	X / X
G3C	1 0 0	192 / 200	X / X
G3R	1 0 1	208 / 216	X / X
G6C	1 1 0	224 / 232	X / X
G6R	1 1 0	240 / 248	X / X

"X" indicates "Don't care."

TABLE 3. VIDEO RAM PAGE SELECTION

	VIDRAM		Page Select Register Bit Pattern
	Size (Bytes)	Start Address	6 5 4 3 2 1 0
4K R A M	512	3584	0 0 0 0 1 1 1
	1024	3072	0 0 0 0 1 1 0
	1536	2560	0 0 0 0 1 0 1
16K R A M	512	15872	0 0 1 1 1 1 1
	1024	15360	0 0 1 1 1 1 0
	1536	14848	0 0 1 1 1 0 1
	2048	14336	0 0 1 1 1 0 0
	3072	13312	0 0 1 1 0 1 0
	6144	10240	0 0 1 0 1 0 0

TABLE 4. DETAILED DESCRIPTION OF THE GRAPHICS MODES

Table 4. Detailed Description of the Graphics Modes

COLOR SET	DATA BIT	Color		Resolution		Comments
		Character Color	Background	Columns x Rows	Detail	
0	0	Green	Black	32 x 16	8 dots 12 dots	The <i>Alphanumeric</i> internal mode uses an internal character generator, which consists of the following dot-by-dot characters: "ABCDEFGHIJKL MNOPQRSTUVWXYZ _# \$%&*+,-./0123456789.:;=?
	1	Orange	Black		ASCII code	
X	X	Lx C2 C1 C0	Color	64 x 32	one element	The <i>Semigraphics-4</i> mode uses an internal "coarse graphics" generator in which a rectangle (eight dots by 12 dots) is divided into four equal parts. The luminance of each part is determined by a corresponding bit on the VDG data bus. Each character is 512 bytes determined by three bits. It requires 512 bytes of display memory.
		0 0 0 0	Black		L1 L2	
0	X	Lx C1 C0	Color	64 x 48		The <i>Semigraphics-6</i> mode is similar to the <i>Semigraphics-4</i> mode, with the following differences: the eight dots by 12 dot rectangle is divided into six equal parts. Color is determined by the two remaining bits. It requires 512 bytes of display memory.
		0 0 0 0	Black		L1 L2 L3	
X	X	Lx C2 C1 C0	Color	64 x 64		The <i>Semigraphics-8</i> mode requires four column consecutive addresses, and produces a 2x4 block. It requires 2048 bytes of display memory.
		0 0 0 0	Black		L1 L2 L3 L4	
X	X	Lx C2 C1 C0	Color	64 x 96		The <i>Semigraphics-12</i> mode requires six column consecutive addresses, and produces a 2x6 block. It requires 3072 bytes of display memory.
		0 0 0 0	Black		L1 L2 L3 L4 L5 L6	

Table 4. Detailed Description of the Graphics Modes (Continued)

COLOR SET	DATA BIT 6	Color		Resolution		Comments																																																	
		Character Color	Background Color	Border	Columns x Rows		Detail																																																
X	X	Lx C2 C1 C0	Black	Black	64 x 192	<table border="1"> <tr><td>L₁</td><td>L₂</td><td>L₃</td><td>L₄</td><td>L₅</td><td>L₆</td><td>L₇</td><td>L₈</td><td>L₉</td><td>L₁₀</td><td>L₁₁</td><td>L₁₂</td><td>L₁₃</td><td>L₁₄</td><td>L₁₅</td><td>L₁₆</td><td>L₁₇</td><td>L₁₈</td><td>L₁₉</td><td>L₂₀</td><td>L₂₁</td><td>L₂₂</td><td>L₂₃</td><td>L₂₄</td></tr> <tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr> </table>	L ₁	L ₂	L ₃	L ₄	L ₅	L ₆	L ₇	L ₈	L ₉	L ₁₀	L ₁₁	L ₁₂	L ₁₃	L ₁₄	L ₁₅	L ₁₆	L ₁₇	L ₁₈	L ₁₉	L ₂₀	L ₂₁	L ₂₂	L ₂₃	L ₂₄	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		L ₁	L ₂				L ₃	L ₄	L ₅	L ₆	L ₇	L ₈	L ₉	L ₁₀	L ₁₁	L ₁₂	L ₁₃	L ₁₄	L ₁₅	L ₁₆	L ₁₇	L ₁₈	L ₁₉	L ₂₀	L ₂₁	L ₂₂	L ₂₃	L ₂₄																											
		X	X				X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X																										
		0	C1 C0				Green	64 x 64	<table border="1"> <tr><td>E₁</td><td>E₂</td><td>E₃</td><td>E₄</td><td>E₅</td><td>E₆</td></tr> </table>	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆	<p>The Graphics-1C mode uses 1024 bytes of display RAM in which one pair of bits specifies one picture element.</p>																																							
		E ₁	E ₂				E ₃			E ₄	E ₅	E ₆																																											
		1	C1 C0				Yellow																																																
		0	C1 C0				Blue																																																
		1	C1 C0				Red																																																
		0	C1 C0				Buff																																																
		1	C1 C0				Cyan																																																
		0	C1 C0				Magenta																																																
		1	C1 C0				Orange																																																
0	Lx	Color	128 x 64	<table border="1"> <tr><td>L₁</td><td>L₂</td><td>L₃</td><td>L₄</td><td>L₅</td><td>L₆</td><td>L₇</td><td>L₈</td></tr> </table>	L ₁	L ₂	L ₃			L ₄	L ₅	L ₆	L ₇	L ₈	<p>The Graphics-1R mode uses 1024 bytes of display RAM in which one bit specifies one picture element.</p>																																								
L ₁	L ₂	L ₃			L ₄	L ₅	L ₆			L ₇	L ₈																																												
1	Lx	Black																																																					
0	Lx	Green																																																					
1	Lx	Blue																																																					
0	Lx	Red																																																					
1	Lx	Buff																																																					
0	Lx	Cyan																																																					
1	Lx	Magenta																																																					
0	Lx	Orange																																																					
0	Lx	Color			128 x 64	<table border="1"> <tr><td>E₁</td><td>E₂</td><td>E₃</td><td>E₄</td><td>E₅</td></tr> </table>	E ₁	E ₂	E ₃	E ₄	E ₅	<p>The Graphics-2C mode uses 2048 bytes of display RAM in which one pair of bits specifies one picture element.</p>																																											
E ₁	E ₂	E ₃					E ₄	E ₅																																															
1	Lx	Black																																																					
0	Lx	Green																																																					
1	Lx	Blue																																																					
0	Lx	Red																																																					
1	Lx	Buff																																																					
0	Lx	Cyan																																																					
1	Lx	Magenta																																																					
0	Lx	Orange																																																					
0	Lx	Color	128 x 96	<table border="1"> <tr><td>L₁</td><td>L₂</td><td>L₃</td><td>L₄</td><td>L₅</td><td>L₆</td><td>L₇</td><td>L₈</td><td>L₉</td><td>L₁₀</td><td>L₁₁</td><td>L₁₂</td><td>L₁₃</td><td>L₁₄</td><td>L₁₅</td><td>L₁₆</td><td>L₁₇</td><td>L₁₈</td><td>L₁₉</td><td>L₂₀</td><td>L₂₁</td><td>L₂₂</td><td>L₂₃</td><td>L₂₄</td></tr> </table>			L ₁	L ₂	L ₃	L ₄	L ₅		L ₆	L ₇	L ₈	L ₉	L ₁₀	L ₁₁	L ₁₂	L ₁₃	L ₁₄	L ₁₅	L ₁₆	L ₁₇	L ₁₈	L ₁₉	L ₂₀	L ₂₁	L ₂₂	L ₂₃	L ₂₄	<p>The Graphics-2R mode uses 1536 bytes of display RAM in which one bit specifies one picture element.</p>																							
L ₁	L ₂	L ₃					L ₄	L ₅	L ₆	L ₇	L ₈		L ₉	L ₁₀	L ₁₁	L ₁₂	L ₁₃	L ₁₄	L ₁₅	L ₁₆	L ₁₇	L ₁₈	L ₁₉	L ₂₀	L ₂₁	L ₂₂	L ₂₃	L ₂₄																											
1	Lx	Black																																																					
0	Lx	Green																																																					
1	Lx	Blue																																																					
0	Lx	Red																																																					
1	Lx	Buff																																																					
0	Lx	Cyan																																																					
1	Lx	Magenta																																																					
0	Lx	Orange																																																					
0	Lx	Color			128 x 96	<table border="1"> <tr><td>E₁</td><td>E₂</td><td>E₃</td><td>E₄</td><td>E₅</td><td>E₆</td></tr> </table>	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆	<p>The Graphics-3C mode uses 3072 bytes of display RAM in which one pair of bytes specifies one picture element.</p>																																										
E ₁	E ₂	E ₃					E ₄	E ₅	E ₆																																														
1	Lx	Black																																																					
0	Lx	Green																																																					
1	Lx	Blue																																																					
0	Lx	Red																																																					
1	Lx	Buff																																																					
0	Lx	Cyan																																																					
1	Lx	Magenta																																																					
0	Lx	Orange																																																					

Table 4. Detailed Description of the Graphics Modes (Continued)

COLOR SET	DATA BIT 6	Color		Border	Resolution		Data Byte(s)	Comments
		Character Color	Background		Columns x Rows	Detail		
0	X	Same colors as Graphics one R		Green	128 x 192	L ₁ L ₂ L ₃ L ₄ L ₅ L ₆ L ₇ L ₈	L ₁ L ₂ L ₃ L ₄ L ₅ L ₆ L ₇ L ₈	The Graphics-3R mode uses 3072 bytes of display RAM in which one bit specifies one picture element.
1	Buff							
0	X	Same colors as Graphics one C		Green	128 x 192	E ₁ E ₂ E ₃	C ₁ C ₂ C ₃ C ₄ C ₅ C ₆ C ₇ C ₈	The Graphics-6C mode uses 6144 bytes of display RAM in which one pair of bits specifies one picture element.
1	Buff							
0	X	Same colors as Graphics one R		Green	256 x 192	L ₁ L ₂ L ₃ L ₄ L ₅ L ₆ L ₇ L ₈	L ₁ L ₂ L ₃ L ₄ L ₅ L ₆ L ₇ L ₈	The Graphics-6R mode uses 6144 bytes of display RAM in which one bit specifies one picture element.
1	Buff							

*Column-consecutive addresses starting at HEX 0400, 0420, 0440, 0460, etc.

USING MACHINE-LANGUAGE SUBROUTINES WITH COLOR BASIC

In this part we describe how to call a machine-language subroutine from a BASIC program, and we list certain ROM subroutines that you may find useful.

"Machine-language" (ML) is the low-level language used internally by your computer. It consists of microprocessor instructions. Machine-language subroutines are useful for special applications because they can do things very fast.

Writing such routines requires familiarity with assembly-language programming and with the microprocessor's instruction set. For more information, see *Basic Microprocessors and the 6800*, Ron Bishop, Hayden Book Company, 1979.

In this section, we take a step-by-step approach to using ML subroutines, as follows:

1. Protecting Memory
2. Storing the ML Subroutine in RAM
3. Telling BASIC Where the Subroutine Is
4. Calling the Subroutine
5. Returning to BASIC

As we go along, we'll present a BASIC program that performs all 5 operations. You may type in the BASIC program lines as they are given, but don't try to run the program until you've read this entire section.

Our ML subroutine is simple. It gets a character from the keyboard. The character is returned as an ASCII code rather than as a string.

The subroutine has a few features not available with INKEY\$ or INPUT. First, it will return any key code, including the one for **BREAK**. Second, it will let you key in control codes A-Z (CTRL-A through CTRL-Z). To key in a control character, press **(A)**, release it, then press any key from **(A)** to **(Z)**. The control codes generated range from 1 to 26.

Upon return from the subroutine, theUSR reference is "replaced" with a character code.

We'll call the subroutine "GETKEY." For a listing of the ML subroutine, see the end of this section.

STEP 1. PROTECTING MEMORY

With the CLEAR statement, you can reserve a section of RAM for storing your ML subroutine. The first CLEAR parameter sets the string space, and the second sets the memory protection address. For example:

```
10 CLEAR 25, 4050
```

sets the string space to 25 bytes and reserves memory addresses from 4051 to the end of RAM (see the Memory Map). Your ML program may then safely be stored in this area.

STEP 2. STORING THE MACHINE LANGUAGE SUBROUTINE IN RAM

You may load ML programs from tape via CLOADM, or you can POKE them into RAM. In our example, we store the individual codes in DATA statements, then read and POKE each code into the correct RAM location. The numbers in the DATA statements are derived from the ML subroutine listed later in this section.

```
20 FOR I = 1 TO 28  
30 READ B: POKE 4050 + I, B  
40 NEXT I
```

```

50 DATA 173, 159, 160, 0
60 DATA 39, 250, 129, 10, 38, 12
70 DATA 173, 159, 160, 0, 39, 250
75 DATA 129, 65, 45, 2
80 DATA 128, 64, 31, 137, 79
90 DATA 126, 180, 244

```

STEP 3. TELLING BASIC WHERE THE SUBROUTINE IS

Before you can use the subroutine, you have to tell your Color Computer where it starts. Do so by POKEing the 2-byte address into RAM locations 275-276. The most significant byte (MSB) goes first, then the least significant byte (LSB).

Our ML will start at decimal 4051, so:

```

Decimal 4051 = Hexadecimal 0F D3 =
Decimal 15(MSB), Decimal 211 (LSB)

```

Here's the program line to accomplish this:

```
100 POKE 275, 15: POKE 276, 211
```

STEP 4. CALLING THE SUBROUTINE

At the correct point in your program, insert a USR function reference:

```
110 A = USR(0)
```

In our example, 0 is a "dummy argument." It won't be used by the ML subroutine. When this statement is encountered, BASIC calls the ML subroutine.

Note: On entry to the subroutine, you can get the USR argument (the 0 in this case) by calling a ROM subroutine, INTCNV, which returns with the integer value in the D register. The address of INTCNV is hexadecimal B3ED.

STEP 5. RETURNING TO BASIC

If you do not want to return any values to the BASIC program, end the subroutine with an RTS instruction. If you want to return a 2-byte integer value, load the integer into register D in MSB-LSB sequence, then end the subroutine by calling a special ROM subroutine, GIVABF. The address of GIVABF is hexadecimal B4F4.

After an RTS, the USR-reference in your BASIC program returns the original dummy argument. After a call to GIVABF, the USR-reference in your BASIC program returns the value you loaded into the D register.

THE BASIC PROGRAM

The following program gets the object code into RAM and then uses the subroutine to get keyboard input. Type it in carefully; then run it.

Each time you press a key, control returns to BASIC with the ASCII code for that key. Try pressing **(BREAK)**. You'll get the code for **(BREAK)** 3. The BASIC program ends when you press **(ENTER)** or **(↓) (M)**.

To get any of the codes 1-26, press **(↓)**, release it, then press a key from **(A)** to **(Z)**.

```

10 CLEAR 25, 4050 'RESERVE MEMORY
15 CLS
20 FOR I = 1 TO 28 "STORE EACH BYTE OF OBJECT
CODE
30 READ B: POKE 4050 + I, B
40 NEXT I
45 'HERE IS THE OBJECT CODE
50 DATA 173, 159, 160, 0
60 DATA 39, 250, 129, 10, 38, 12
70 DATA 173, 159, 160, 0, 39, 250
75 DATA 129, 65, 45, 2
80 DATA 128, 64, 31, 137, 79
90 DATA 126, 180, 244
99 'TELL BASIC WHERE THE ROUTINE IS
100 POKE 275, 15: POKE 276, 211

```



```

110 A = USR(0) 'CALL THE SUBROUTINE AND GIVE
    RESULT TO A
115 IF A = 13 THEN END
120 PRINT "CODE ="; A
130 GOTO 110

```

Note to Customers with 16K RAM

You may change Lines 10 and 30:

```

10 CLEAR 25, 16350
30 READ B: POKE 16350 + I, B

```

For a variation in the program, change Line 120 to:

```

120 PRINT CHR$(A); 'DISPLAY THE CHARACTER

```

Most control keys (↓) followed by a key (A) — (Z) will have no effect when they are printed. But try control—H (backspace).

ML SUBROUTINE LISTING

Note: Don't type this in. It is here for those who want to understand how the ML subroutine works.

Hexadecimal Object Code	Source	Code	Comments
AD 9F A0 00	LOOP1	JSR (POLCAT)	;POLL FOR A KEY
27 FA		BEQ LOOP1	;IF NONE, RETRY
81 0A		CMPA #10	;CTRL KEY (DN ARW)?
26 0C		BNE OUT	;NO, SO EXIT
AD 9F A0 00	LOOP2	JSR (POLCAT)	;YES, SO GET
27 FA		BEQ LOOP2	;IF NONE, RETRY
81 20		CMPA #65	;IS IT A - Z?
2D 02		BLT OUT	;IF < A, EXIT
80 40		SUBA #64	;CONVERT TO CTRL A/Z
1F 89	OUT	TFR A,B	;GET RETURN BYTE READY
4F		CLRA	;ZERO MSB
7E B4 F4		JMP GIVABF	;RETURN VALUE TO BASIC
POLCAT EQU 40960			
GIVABF EQU 46324			

Note: "Source code" is not meaningful to the computer. It is a set of memory aids and symbols we use for convenience. The source code must be translated or "assembled" into object code, which the computer understands. In the listing above, the object code is given in hexadecimal form. We converted it to decimal numbers for our BASIC program.

ROM SUBROUTINES AVAILABLE FOR USE FROM BASIC

Color BASIC ROM contains many subroutines that can be called by a machine-language program; many of these can be called by a Color BASIC program via the USR function. Each subroutine is described in the following format:

NAME — *Entry address*
Operation Performed
Entry Condition
Exit Condition

Note: The subroutine **NAME** is only for reference. Your Color Computer does not recognize it. 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). CBMFAD 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 = CSRERR = 2 (if a memory error occurs).

(Note: CSRERR = 81)

Unless a memory error occurs, X = CBUFAD + BLKLEN. 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 55x should have been written if this first block to be written after a motor-on.

CBUFAD, located at 7E, contains the buffer address.

BLKTYP, located at 7C, contains the block type.

BLKLEN, located at 7D, contains the number of data bytes.

Exit Conditions

Interrupts are masked.

X = CBUFAD + BLKLEN.

All registers are modified.

WRTLDR = [A00C]

Turns the Cassette On and Writes a Leader

Entry Conditions

None

Exit Conditions

None

CHROUT = [A002]

Outputs a Character to Device

CCHROUT 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 15D

Right/Left 15C

Right Joystick

Up/Down 15B
Right/Left 15A

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.

MEMORY CONTENTS

This table shows the contents of the Color Computer's memory. The first column shows the memory address in decimal notation; the second, in hexadecimal notation.

Decimal	Hex	Memory Contents
0-105	0-69	Direct page RAM (can be used by machine language programs)
112-255	70-FF	Direct page RAM (cannot be used by machine language programs using any of BASIC's subroutines)
256-273	100-111	Internal Use (Interrupt Vector's)
274-276	112-114	USR/JMP - Jump to BASIC's USR routine
277-281	115-119	Can be used by machine language programs
282	11A	Keyboard Alpha lock — O = not locked, FF = locked
283-284	11B-11C	Keyboard delay constant
285-337	11D-151	Can be used by machine language programs
338-345	152-159	Keyboard rollover table
346-349	15A-15D	Joystick pot values
350-1023	15E-3FF	Internal Use
1024-1535	400-5FF	Video Memory
1536-4095	600-0FFF	Program and Variable Storage (4K RAM)
1536-16383	600-3FFF	Program and variable storage (16K RAM)
16384-32767	4000-7FF	Not Used
32768-40959	8000-9FFF	Extended Color BASIC
40960-49151	A000-BFFF	COLOR BASIC (8K ROM)
49152-65279	C000-FEFF	Program Pak Memory
65280-65535	FF00-FFFF	Input/Output

10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

APPENDIXES



APPENDIX A

Musical Tones

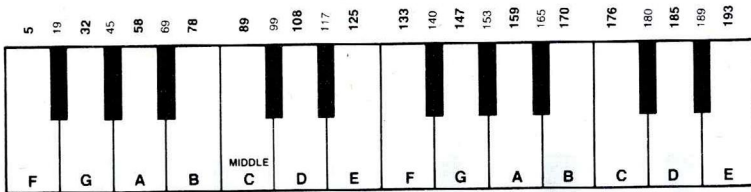
Your computer can come fairly close to matching (although it can't exactly match) the musical tones shown below. You may use either the piano keyboard or the musical staff to determine the numeric code that represents the note you want.

If you're using the piano keyboard, the numeric code for each key is directly over the key. For example, the numeric code for middle C is 89.

If you're using the musical staff, the numeric code for each note is below the note. For example, the numeric code for



is 108.



If the note is a flat, select the numeric code immediately preceding the note. For example:



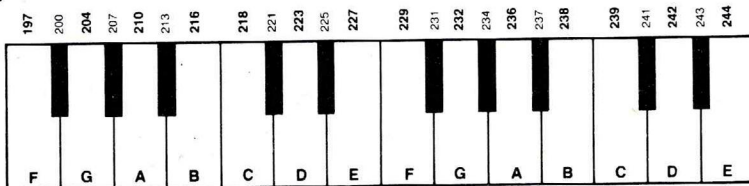
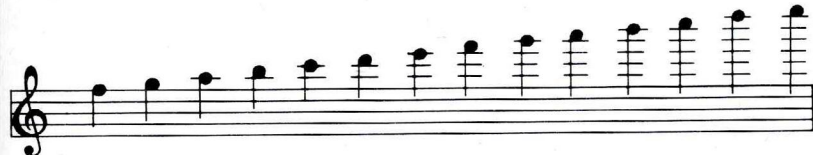
is 99.

If the note is a sharp, select the numeric code immediately following the note. For example:



is 117.

Chapter 5 shows how to program the computer to play a song.



APPENDIX B

BASIC Colors and Graphics Characters

These are the codes for the colors you can create on your screen.

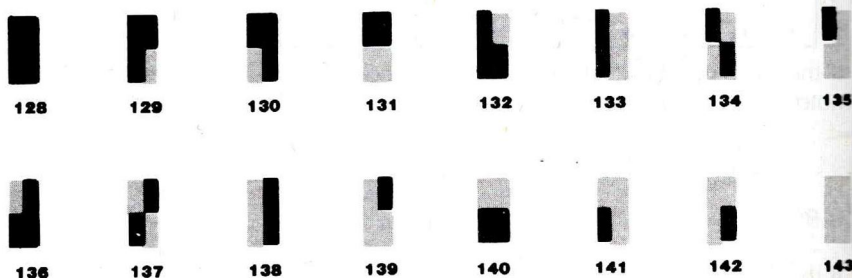
BASIC COLORS

- | | |
|------------------------------|-------------|
| 0 — black (absence of color) | 5 — buff |
| 1 — green | 6 — cyan |
| 2 — yellow | 7 — magenta |
| 3 — blue | 8 — orange |
| 4 — red | |

When using SET, color 0 will leave a dot's color unchanged.

GRAPHICS CHARACTERS

These are the codes for the Color Computer's graphics characters. To produce them, use CHR\$ with the character's code. For example, PRINT CHR\$ (129) produces character 129.



To print all these graphics characters, type and run this program:

To create these characters in one of the colors below, add the appropriate number to the code. For example, PRINT CHR\$ (129 + 16) produces character 129, except the green area is yellow.

- | | | |
|---------------|-------------|----------------|
| + 16 — yellow | + 64 — buff | + 96 — magenta |
| + 32 — blue | + 80 — cyan | + 112 — orange |
| + 48 — red | | |

Chapter 16 explains how to use graphics characters.

APPENDIX C

PRINT @ SCREEN LOCATIONS

APPENDIX D





GRAPHICS SCREEN LOCATIONS





APPENDIX E

ASCII Character Codes

These are the ASCII codes for each of the characters on your keyboard. The first column is the character; the second is the code in decimal notation; and the third converts the code to a hexadecimal (16-based number).

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
SPACEBAR	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
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

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
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
 *	94	5E
 *	10	0A
 *	8	08
 *	9	09
BREAK	03	03
CLEAR	12	0C
ENTER	13	0D

*If shifted, the codes for these characters are as follows: **CLEAR** is 92 (hex 5C);  is 95 (hex 5F);  is 91 (hex 5B);  is 21 (hex 15); and  is 93 (hex 5D).

Lowercase Codes

These are the ASCII codes for lowercase letters. You can produce these characters by pressing the **SHIFT** and **O** keys simultaneously to get into an upper- lowercase mode. The lowercase letters will appear on your screen in reversed colors (green with a black background).

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
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

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
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

APPENDIX F

Answers to Exercises

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

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 5-2

Program that shows 9 colors for one second each:

```

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 7-1

Russian Roulette:

```
5 FOR N = 1 TO 10
10 PRINT "CHOOSE YOUR CHAMBER(1-10)"
20 INPUT X
30 IF X = RND(10) THEN 100
40 SOUND 200, 1
50 PRINT "--CLICK--"
60 NEXT N
65 CLS
70 PRINT @ 230, "CONGRATULATIONS!!!"
80 PRINT @ 265, "YOU MANAGED"
90 PRINT @ 296, "TO STAY ALIVE"
95 END
100 FOR T = 133 TO 1 STEP -5
110 PRINT "BANG!!!!!"
120 SOUND T, 1
130 NEXT T
140 CLS
150 PRINT @ 230, "SORRY, YOU'RE DEAD"
160 SOUND 1, 50
170 PRINT @ 290, "NEXT VICTIM PLEASE"
```

Do-It-Yourself Program 7-2

Craps game:

```
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 INPUT A$
200 X = 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 = 1 TO 1000
610 NEXT X
615 CLS
620 PRINT @ 264, "SORRY, YOU LOSE"
630 PRINT @ 458, "GAME'S OVER"

```

Do-It-Yourself Program 8-1

Test Your Arithmetic

```

5 CLS
6 PRINT @ 230, "YOUR NAME";
8 INPUT N$
10 CLS
15 T = T + 1
20 X = RND(100)
30 Y = RND(100)
40 PRINT @ 228, "WHAT IS" X "+" Y;
45 INPUT A
50 IF A = X + Y THEN 82
60 PRINT @ 326, "THE ANSWER IS" X + Y
70 PRINT @ 385, "BETTER LUCK NEXT TIME," N$
80 GOTO 100
82 CLS(7)
83 FOR M = 1 TO 4
84 SOUND 175, 1
85 SOUND 200, 1
86 NEXT M
87 CLS
90 PRINT @ 232, "CORRECT," N$ "!!!"
95 C = C + 1
97 PRINT @ 299, "THAT IS"
98 PRINT @ 322, C "OUT OF" T "CORRECT
  ANSWERS"
99 PRINT @ 362, C/T*100 "% CORRECT"
100 PRINT @ 420, "PRESS <ENTER> WHEN READY"
102 PRINT @ 458, "FOR ANOTHER"
105 INPUT A$
110 GOTO 10

```

Do-It-Yourself Program 9-1

Table of squares:

```

5 CLS
7 PRINT @ 38, "TABLE OF SQUARES"

```

```

8 PRINT
10 P = 2
20 FOR N = 2 TO 10
25 GOSUB 2000
30 PRINT N "*" N "=" E,
40 NEXT N
50 END
2000 REM FORMULA FOR RAISING A NUMBER TO A
    POWER
2010 E = 1
2020 FOR X = 1 TO P
2030 E = E * N
2040 NEXT X
2050 IF P = 0 THEN E = 1
2060 RETURN

```

Do-It-Yourself Program 10-1

Challenger Program:

```

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$)
110 NS$ = LEFT$(S$,X-1) + R$ +
    RIGHT$(S$,LEN(S$) - E)
120 PRINT "NEW SENTENCE IS : "
130 PRINT NS$

```

Do-It-Yourself Program 11-1

Computer typing test:

```

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
142 FOR X = 1 TO 32
144 IF MID$("NOW IS THE TIME FOR ALL GOOD
    MEN",X,1) <> MID$(B$,X,1) THEN E = E + 1
146 NEXT X
150 PRINT
160 PRINT "YOU TYPED AT--" R "--WDS/MIN"
170 PRINT "WITH" E "ERRORS"

```

Do-It-Yourself Program 16-1

Forward spacing dot:

```

10 CLS(0)
20 H = 63
25 SET(H,14,3)
30 A$ = INKEY$
40 IF A$ = CHR$(8) THEN 60
45 IF A$ = CHR$(9) THEN 100
50 GOTO 30
60 H = H - 1
65 IF H < 0 THEN H=0: GOTO 30
70 SET(H,14,3)
75 RESET(H + 1, 14)
80 GOTO 30
100 H = H + 1
110 IF H > 63 THEN H=63: GOTO 30
120 SET(H,14,3)
130 RESET(H-1,14)
140 GOTO 30

```

Do-It-Yourself Program 16-2

Table and chairs:

```

10 LC$ = CHR$(139 + 16) + CHR$(130 + 16)
20 TA$ = CHR$(142 + 112) + CHR$(140 + 112) +
    CHR$(141 + 112)
30 RC$ = CHR$(129 + 16) + CHR$(135 + 16)
40 CLS(0)
50 PRINT @ 236, LC$ + TA$ + RC$;
60 GOTO 60

```

Do-It-Yourself Program 18-1:

Checkbook program:

```

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 19-1

Inventory program:

```

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 19-2

Dealing a hand:

```

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 20-1

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 20-2

Lines that change the song lyrics:

```
110 PRINT
120 INPUT "WHICH LINE DO YOU WANT TO
REVISE"; L
125 IF L > 4 THEN 120
130 PRINT "TYPE THE REPLACEMENT LINE"
140 INPUT A$(L)
150 GOTO 50
```

Do-It-Yourself Program 20-3

Word processor challenger:

```
1 CLEAR 1000
5 DIM A$(50)
7 CLS
10 PRINT "TYPE A PARAGRAPH"
16 :
20 PRINT "PRESS </> WHEN FINISHED"
30 X = 1
40 A$ = 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 20-4

Printing on the printer:

```

150 PRINT #-2, A$(Y);

```

Do-It-Yourself Program 21-1

Alphabetizing book collection:

```

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 "0", #-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$(X) THEN X = Y
4065 Z(T) = X
4080 NEXT Y
4085 T = T + 1
4090 M$(X) = "ZZ"
4100 GOTO 4010

```

Do-It-Yourself Program 22-1

Deal two-dimensional cards:

```
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
```

APPENDIX G

Subroutines

These subroutines let you run programs that require advanced math functions not directly available in Color BASIC.

Each subroutine listing has a set of instructions in the margin. Study them closely. You'll see that some subroutines require other subroutines for internal calculations. You must enter these "auxiliary subroutines" when the instructions call for them.

Note: Subroutines are not as accurate as Color BASIC's math operators and functions. This is because:

- The subroutines contain many chain calculations, which tend to magnify the small error of individual operations.
- The subroutines are only approximations of the functions they replace.

In general, the subroutines are accurate to 5 or 6 decimal places over much of their allowable range, with a decrease in accuracy as the input approaches the upper or lower limits for input values.

Square Root

Computes: $\text{SWR}(X)$, X

Input: X , must be greater than or equal to zero

Output: Y

Also uses: W,Z internally
Other subroutines required: None
How to call: GOSUB 30030

```
30000 END
30010 REM *SQUARE ROOT* INPUT X, OUTPUT Y
30020 REM ALSO USES W & Z INTERNALLY
30030 IF X = 0 THEN Y = 0: RETURN
30040 IF X > 0 THEN 30060
30050 PRINT "ROOT OF NEGATIVE NUMBER?": STOP
30060 Y = X * .5: Z = 0
30070 W = (X/Y - Y) * .5
30080 IF (W = 0) + (W = Z) THEN RETURN
30090 Y = Y + W : Z = W: GOTO 30070
```

Exponentiation

Computes: X^Y (X to the Y power)
Input: X, Y. If X is less than zero, Y must be an odd integer
Output: P
Also uses: E, L, A, B, C internally. Value of X is changed.
Other subroutines required: Log and Exponential
How to call: 30120

```
30000 END
30100 REM *EXPONENTIATION* INPUT X,Y; OUTPUT
P
30110 REM ALSO USES E,L,A,B,C INTERNALLY
30120 P=1: E=0: IF Y=0 THEN RETURN
30130 IF (X<0)AND(INT(Y)=Y) THEN P=1-
2*Y+4*INT(Y/2): X=-X
30140 IF X<>0 THEN GOSUB 30190: X=Y*L: GOSUB
30250
30150 P=P*E: RETURN
```

Logarithms (Natural and Common)

Computes: LOG(X) base e, and LOG(X) base 10
Input: X greater than or equal to zero
Output: L is natural log (base e), X is common log (base 10)
Also uses: A, B, C internally. Value of X is changed.
Other subroutines required: None
How to call: GOSUB 30190

```
30000 END
30170 REM *NATURAL & COMMON LOG: INPUT X,
OUTPUT L,X
30175 REM OUTPUT L IS NATURAL LOG, OUTPUT X
IS COMMON LOG
30180 REM ALSO USES A,B,C INTERNALLY
30190 E=0: IF X<0 THEN PRINT "LOG UNDEFINED
AT"; X: STOP
30195 A=1: B=2: C=.5
30200 IF X>=A THEN X=C*X: E=E+A: GOTO 30200
30205 IF X<C THEN X=B*X: E=E-A: GOTO 30205
30210 X=(X-.707107)/(X+.707107): L=X*X
```

```

30215 L=(((.598979*L+.961471)*L+2.88539)
      *X+E-.5)*.693147
30220 IF ABS(L)<1E-6 THEN L=0
30225 X=L*.4342945: RETURN

```

Exponential

Computes: EXP (X) (e to the X power)

Input: X

Output: E

Also uses: L,A internally. Value of X is changed.

Other subroutines required: None

How to call: GOSUB 30250

```

30000 END
30240 REM *EXPONENTIAL* INPUT X, OUTPUT E
30245 REM ALSO USES L,A INTERNALLY
30250 L=INT(1.4427*X)+1: IF L<127 THEN 30265
30255 IF X>0 THEN PRINT "OVERFLOW": STOP
30260 E=0: RETURN
30265 E=.693147*L-X: A=1.32988E-3-
      1.41316E-4*E
30270 A=((A*E-8.30136E-3)*E+4.16574E-2)*E
30275 E=((A-.166665)*E+.5)*E-1)*E+1: A=2
30280 IF L<=0 THEN A=.5: L=-L: IF L=0 THEN
      RETURN
30285 FOR X=1 TO L: E=A*E: NEXT X: RETURN

```

Tangent

Computes: TAN(X)

Input: X in degrees

Output: Y

Other subroutines required: Cosine

How to call: GOSUB 30310

```

30000 END
30300 REM *TANGENT* INPUT X IN DEGREES,
      OUTPUT Y
30310 IF ABS(SIN((90-X)/57.29577951)) <1E-7
      THEN PRINT "UNDEFINED": STOP
30320 Y=SIN(X/57.29577951)/SIN((90-X) /
      57.29577951)
30330 RETURN

```

Cosine

Computes: COS(X)

Input: X in degrees

Output: Y

Other subroutines required: None

How to call: GOSUB 30360

```

30000 END
30350 REM *COSINE* INPUT X IN DEGREES,
      OUTPUT Y

```



```
30360 Y=SIN((90-X)/57.29577951)
30365 RETURN
```

Arc Cosine

Computes: $\text{Arccos}(S)$, angle whose cosine is S

Input: S , $0 \leq S \leq 1$

Output: Y in degrees, W is in radians

Also uses: X, Z internally

Other subroutines required: ArcSine

How to call: GOSUB 30500

```
30000 END
30500 REM *ARCCOS* INPUT S, OUTPUT Y,W
30510 REM Y IS IN DEGREES, W IS IN RADIANS
30520 GOSUB 30550: Y=90-Y: W=1.570796-W:
RETURN
```

Arc Sine

Computes: $\text{ArcSin}(S)$, angle whose sine is S

Input: S , $0 \leq S \leq 1$

Output: Y in degrees, W in radians

Also uses: X, Y internally

Other subroutines required: None

How to call: 30550

```
30000 END
30530 REM *ARCSIN SUBROUTINE* INPUT S,
OUTPUT Y,W
30535 REM Y IS IN DEGREES, W IS IN RADIANS
30540 REM ALSO USES VARIABLES X,Z INTERNALLY
30550 X=S: IF ABS(S)<=.707107 THEN 30610
30560 X=1-S*S: IF X<0 THEN PRINT S;"IS OUT OF
RANGE": STOP
30565 IF X=0 THEN W=90/57.29577951: GOTO
30630
30570 W=X/2: Z=0
30580 Y=(X/W-W)/2: IF (ABS(Y)<.1E-
8)AND(Y=Z) THEN X=W: GOTO 30610
30600 W=W+Y: Z=Y: GOTO 30580
30610 Y=X+X*X*X/6+X*X*X*X*X*.075+X*X*X*X
*X*X*X*4.464286E-2
30620 W=Y+X*X*X*X*X*X*X*X*X*3.038194E-2
30625 IF ABS(S)>.707107 THEN W=1.570796-W
30630 Y=W*57.29577951: RETURN
```

Arc Tangent

Computes: $\text{ATN}(X)$, angle whose tangent is X

Input: X

Output: C in degrees, A in radians

Also uses: B, T internally. Value of X is changed.

Other subroutines required: None

How to call: GOSUB 30690

```
30000 END
30660 REM *ARCTANGENT* INPUT X, OUTPUT C,A
30670 REM C IS IN DEGREES, A IS IN RADIANS
30680 REM ALSO USES B,T INTERNALLY
30690 T=SGN(X): X=ABS(X): C=0
30700 IF X>1 THEN C=1: X=1/X
30710 A=X*X
30720 B-((2.86623E-3*A-1.61657E-2)*A+
4.29096E-2)*A
30730 B=(((B-7.5289E-2)*A+.106563)*A-
.142089)*A+.199936)*A
30740 A=((B-.333332)*A+1)*X
30750 IF C=1 THEN A=1.570796-A
30760 A=T*A: C=A*57.29577951: RETURN
```

APPENDIX H

Sample Programs

Space Guns

```
10 CLEAR 1000
20 FOR Y = 0 TO 1
30 C = (Y+1)*16
40 S$(Y) = CHR$(131+C)+CHR$(139+C)+CHR$(
130+C)
50 S2$(Y) = CHR$(128+C)+CHR$(136+C)
60 NEXT Y
100 FOR Y = 0 TO 1
105 C = JOYSTK(0)
110 A(Y) = JOYSTK(0+Y*2)
120 B(Y) = JOYSTK(1+(Y*2))
130 IF A(Y) > 59 THEN A(Y) = 59
140 B(Y) = INT(B(Y)/4) * 4
150 L(Y) = B(Y) * 8 + INT(A(Y)/2)
160 IF L(Y) >= 480 THEN L(Y) = L(Y) - 32
170 NEXT Y
180 CLS(0)
190 FOR Y = 0 TO 1
200 PRINT @ L(Y), S$(Y);
210 PRINT @ L(Y)+32, S2$(Y);
220 NEXT Y
500 P = PEEK(65280)
510 IF P = 125 OR P = 253 THEN GOSUB 1000
530 GOTO 100
800 REM
900 REM FIRE GUN ROUTINE
1000 V1 = INT(B(1)/2)+1
1010 H1 = A(1) + 2
1020 IF A(1) > A(0) THEN 1100
```

```

1030 FOR H = H1 + 3 TO 63
1040 IF POINT(H,V1) = 2 THEN SOUND 100,2
1050 SET(H,V1,4)
1060 IF H <= H1 + 4 THEN 1080
1070 RESET (H-2, V1)
1080 NEXT H
1090 RETURN
1100 FOR H = H1 TO 4 STEP -1
1110 IF H = H1 THEN 1160
1120 IF POINT(H-4,V1)=2 THEN SOUND 100,2
1130 SET(H-4,V1,4)
1140 IF H >= H1 - 2 THEN 1160
1150 RESET(H-2,V1)
1160 NEXT H
1170 RETURN

```

Bouncing Ball

```

5 CLEAR 12
8 INPUT "BACKGROUND COLOR(1-8)"; C
9 CLS(C)
10 X=13: Y=13
15 XM = 20: YM = 15
400 F=0
410 XT = X: YT = Y
420 X = X + XM: Y = Y + YM
430 TX = X: TY = Y: T1 = XM: T2 = YM
440 GOSUB 1000
450 X = TX: Y = TY: XM = T1: YM = T2
455 H = INT(XT/2)*2: V = INT(YT/2)*2
460 SET(H,V,C): SET(H+1,V,C)
462 SET(H,V+1,C): SET(H+1,V+1,C)
470 RESET(X,Y)
480 GOTO 400
499 REM
1000 REM CHECK BOUNDARIES
1010 IF TX > 63 THEN TX = 63: T1 = -T1
1020 IF TX < 0 THEN TX = 0: T1 = -T1
1030 IF TY > 31 THEN TY = 31: T2 = -T2
1040 IF TY < 0 THEN TY = 0: T2 = -T2
1099 RETURN

```

Blackjack

```

5 REM BUILD ARRAYS
7 DIM S$(5), N$(13), D(52), P(5), C(5)
10 DATA 16, 32, 48, 96, 1
20 DATA *ACE**, *TWO**, *THREE**, *FOUR**,
*FIVE**, *SIX**, SEVEN*, EIGHT*, *NINE**,
*TEN**, *JACK**, QUEEN*, *KING*
30 FOR X = 1 TO 5: READ S: S$(X) =
CHR$(143+S): NEXT X
40 FOR X = 1 TO 13: READ N$: N$(X) = N$: NEXT
X
45 CLS(6)

```

```

46 PT = 0: CT = 0
47 FOR X = 1 TO 5: P(X) = 0: C(X) = 0: NEXT
50 FOR X = 1 TO 52: D(X) = X: NEXT X
60 FOR X = 1 TO 5: GOSUB 1000: P(X) = Z: NEXT
  X
70 FOR X = 1 TO 3: GOSUB 1000: C(X) = Z: NEXT
  X
72 REM
75 REM      PRINT PLAYER'S HAND
80 L = 257
90 FOR M = 1 TO 2: C = P(M): GOSUB 500: PT =
  PT + T: NEXT
100 FOR M = 1 TO 3: S = 5: GOSUB 2000: NEXT
102 REM
105 REM      PRINT COMPUTER'S HAND
110 L = 10
120 S = 5: GOSUB 2000
130 C = C(2): GOSUB 500: CT = CT + T
150 PRINT @ 8, "COMPUTER'S HAND";
160 PRINT @ 267, "YOUR HAND";
200 L = 269: K = 3
205 PRINT @ 230, "ANOTHER CARD(Y/N)?";
210 R$=INKEY$: IF R$=" " THEN 210
220 IF R$ = "N" THEN 255
230 C = P(K): GOSUB 500
240 PT = PT + T
242 FOR X = 1 TO K
244 IF PT > 21 AND (P(X)-1)/13 = INT((P(X)-
  1)/13) THEN PT = PT - 10
246 NEXT X
247 IF PT > 21 THEN PRINT @ 488, "YOU
  BUSTED!!!"; GOTO 400
250 K = K + 1: IF K < 6 THEN 205
255 L=10
260 C = C(1): GOSUB 500: CT = CT + T
360 IF PT <=CT THEN 380
370 PRINT @ 484, "CONGRATULATIONS WINNER!";
375 GOTO 390
380 PRINT @ 487, "TOUGH LUCK, KID";
390 REM
400 PRINT @ 230, "ANOTHER GAME(Y/N)?";
410 R$=INKEY$: IF R$=" " THEN 410
420 IF R$ = "Y" THEN 45 ELSE END
430 IF N = 1 THEN T = 11
500 GOSUB 4000: GOSUB 2000
510 GOSUB 3000: RETURN
900 REM
1000 REM      DEAL THE CARDS
1005 Z = RND(52)
1010 IF D(Z) = 0 THEN 1000
1020 D(Z) = 0
1030 RETURN
1900 REM
2000 REM      PRINT THE SUITS
2005 L1 = L
2010 FOR X = 1 TO 6
2015 L1 = L1 + 32

```

```

2020 FOR Y = 1 TO 5
2030 PRINT @ L1 + (Y-1), S$(S);
2040 NEXT Y,X
2045 L1 = 0: L = L + 6
2050 RETURN
2900 REM
3000 REM      PRINT THE NUMBERS
3005 L1 = L - 6
3010 FOR X = 1 TO 6
3020 L1 = L1 + 32
3030 PRINT @ L1+2, MID$(N$(N), X, 1);
3040 NEXT X
3045 L1 = 0
3050 RETURN
3900 REM
4000 REM      COMPUTE NUMBER AND SUIT
4005 S = INT((C-1)/13)+1
4010 N = C-(S*13-13)
4015 REM      COMPUTE POINT VALUE
4020 IF N = 11 OR N = 12 OR N = 13 THEN T = 10
      ELSE T=N
4030 IF N = 1 THEN T = 11
4040 RETURN

```

Kaleidoscope

```

10 CLS0
20 X=RND(32)-1
30 Y=RND(16)-1
40 Z=RND(9)-1
50 GOSUB90
60 GOTO20
90 IFZ=0 OR RND(7)=3THEN150
100 SET(31-X,16+Y,Z)
110 SET(31-X,15-Y,Z)
120 SET(32+X,16+Y,Z)
130 SET(32+X,15-Y,Z)
140 RETURN
150 RESET(31-X,16+Y)
160 RESET(31-X,15-Y)
170 RESET(32+X,16+Y)
180 RESET(32+X,15-Y)
190 RETURN

```

Electronic Dice

```

4 CLEAR 2000
5 CLS(3)
6 DIM DB$(6)
8 DIM DF(21), P(6), D$(6)
10 REM      FACES IF DIE
20 FOR X = 1 TO 21
30 READ DF(X)
40 NEXT X

```

```

50 DATA 39
60 DATA 14, 64
70 DATA 14, 39, 64
80 DATA 14, 20, 58, 64
90 DATA 14, 20, 39, 58, 64
100 DATA 14, 20, 36, 42, 58, 64
105 FOR X = 1 TO 7
110 REM
120 REM      PLACE IN ARRAY DF
130 FOR X = 1 TO 6
140 READ P(X)
150 NEXT X
160 DATA 1, 2, 4, 7, 11, 16
165 REM
170 REM      BUILD DIE STRING
175 FOR X = 1 TO 6
180 M = P(X)
185 FOR Y = 1 TO 7
190 FOR Z = 1 TO 11
192 IF (Y-1)*11+Z <> DF(M) THEN 200
194 D$(X) = D$(X) + CHR$(128)
196 M = M + 1
197 IF M = 22 THEN M = 0
198 IF M = X THEN M = 0
199 GOTO 230
200 D$(X) = D$(X) + CHR$(143+96)
230 NEXT Z
240 FOR Z = 0 TO 31-11
250 D$(X) = D$(X) + CHR$(143+32)
260 NEXT Z
270 NEXT Y, X
480 REM
490 REM      ROLL DICE
500 FOR T = 1 TO 10
510 A=RND(6): B = RND(6)
520 PRINT @ 35, D$(A);
530 PRINT @ 273, D$(B);
540 NEXT T
550 PRINT @ 113, "PRESS ANY KEY";
560 PRINT @ 145, "FOR NEXT ROLL";
570 K$=INKEY$: IF K$=" " THEN 570
580 GOTO 500

```

Play Back Your Tune

```

5 DIM A(25), S$(13), B(200): Y=1
10 FOR X = 1 TO 25: READ A(X): NEXT X
20 DATA 89, 99, 108, 117, 125
30 DATA 133, 140, 147, 153, 159
40 DATA 165, 170, 176, 180, 185
50 DATA 189, 193, 197, 200, 204
60 207, 210, 213, 216, 218
70 FOR X = 1 TO 13: READ S$(X): NEXT X
80 DATA A,W,S,E,D,F,T,G,Y,H,U,J,K
90 CLS

```

```

92 PRINT @ 167, "COMPOSE YOUR SONG"
94 PRINT @ 227, "USE KEYS ON 2ND & 3RD ROWS"
96 PRINT @ 292, "PRESS <X> WHEN FINISHED"
100 P$ = INKEY$
110 IF P$ = " " THEN 100
115 FOR X = 1 TO 13
120 IF P$ <> S$(X) THEN 150
130 SOUND A(X), 5
140 B(Y) = X
145 Y = Y + 1
150 NEXT X
160 IF P$ <> "X" THEN 100
165 CLS
170 PRINT @ 202, "SONG PLABACK"
174 PRINT @ 264, "WHICH KEY(1-11)";
176 INPUT K
180 FOR X = 1 TO Y-1
190 SOUND A(B(X)+K), 5
200 NEXT X
210 GOTO 165

```

Learn That Tune

```

10 DIM M(50), T(8)
20 FOR B = 1 TO 8
30 READ T(B)
40 NEXT B
50 X = 1
60 M(X) = RND(8)
70 FOR Y = 1 TO X
80 CLS(M(Y))
90 PRINT @ 239, M(Y);
100 SOUND T(M(Y)), 8
110 NEXT Y
120 CLS
130 PRINT @ 231, "PLAY BACK THE TUNE";
140 FOR Y = 1 TO X
150 T = 1
160 K$ = INKEY$
170 T = T + 1
180 IF T > 150 THEN 310
190 IF K$ = " " THEN 160
200 K = VAL(K$)
210 IF K <> M(Y) THEN 310
220 CLS(K)
230 PRINT @ 239, K;
240 SOUND T(K), 3
250 NEXT Y
260 X = X + 1
270 CLS: PRINT @ 230, "LISTEN TO NEXT TUNE";
280 FOR T = 1 TO 500: NEXT T
290 CLS: PRINT @ 230, "LISTEN TO NEXT TUNE";
300 GOTO 60
310 CLS(0)
320 PRINT @ 235, "YOU LOSE";

```

330 SOUND 1, 25
340 DATA 89, 108, 125, 133, 147, 159, 170,
176

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$(X)
5090 NEXT X
5100 CLOSE #-1: RETURN
5900 REM
6000 REM      LOAD ITEMS FROM TAPE
6010 CLS: PRINT @ 136, "LOAD ITEMS FROM TAPE"
6020 PRINT @ 235, "REWIND TAPE"
6030 PRINT @ 300, "PRESS PLAY"
6040 PRINT @ 388, "PRESS <ENTER> WHEN READY"
6050 INPUT R$
6060 OPEN "I", #-1, "LIST"
6070 Y = 1
6080 IF EOF(-1) THEN 6120
6090 INPUT #-1, S$(Y)
6095 PRINT S$(Y)
6100 Y = Y + 1
6110 GOTO 6080
6120 CLOSE #-1: RETURN

```

Bar Graph

```

10 DIM A(5;3;2), A$(5)
20 DATA UTILITIES, PERSONNEL, SUPPLIES,
   RENT, TRAVEL
30 FOR X = 1 TO 5
40 READ A$(X)
50 CLS
60 PRINT @ 139, "EXPENSES"
70 PRINT @ 175 - INT(LEN(A$(X))/2), A$(X)
80 PRINT
90 FOR Y = 1 TO 3
100 PRINT "DEPT" Y
110 INPUT "BUDGETED"; A(X,Y,1)

```

```

120 INPUT "ACTUAL"; A(X,Y,2)
130 NEXT Y
140 NEXT X
150 CLS
160 PRINT @ 133, "WOULD YOU LIKE TO SEE"
170 L = 203
180 FOR X = 1 TO 5
190 PRINT @ L, X; A$(X)
200 L = L + 32
210 NEXT X
220 PRINT @ 460, "(1-5)"
230 INPUT X
235 C(1)=0:C(2)=0:LC(1)=0:LC(2)=0
240 FOR Y = 1 TO 3
250 C(1) = A(X,Y,1)+C(1)
260 C(2) = A(X,Y,2) + C(2)
270 NEXT Y
280 IF C(2) > C(1) THEN 310
290 LC(1)=30: LC(2)=INT(C(2)/C(1)*30)
300 GOTO 320
310 LC(2)=30: LC(1)=INT(C(1)/C(2)*30)
320 P = 129
330 CLS(0)
340 PRINT @ 11, "EXPENSES";
350 PRINT @ 47 - INT(LEN(A$(X))/2), A$(X);
360 PRINT @ 97, "BUDGETED";
370 PRINT @ 257, "ACTUAL";
380 PRINT @ 448, CHR$(159)+CHR$(159);
390 PRINT @ 451, "DEPT 1";
400 PRINT @ 459, CHR$(175)+CHR$(175);
410 PRINT @ 462, "DEPT 2";
420 PRINT @ 470, CHR$(191)+CHR$(191);
430 PRINT @ 473, "DEPT 3";
440 PRINT @ 480, "PRESS ANY KEY TO CONTINUE";
450 FOR M = 1 TO 2
460 FOR N = 1 TO 2
470 P1 = P + 32
480 FOR Y = 1 TO 3
490 D(Y) = INT(A(X,Y,M)/C(1)*LC(1))
500 FOR O = 1 TO D(Y)
510 PRINT @ P1, CHR$(143+16*Y);
520 P1 = P1 + 1
530 NEXT O
540 NEXT Y
550 P = P + 32
560 NEXT N
570 P = 289
580 NEXT M
590 K$ = INKEY$: IF K$=" " THEN 590
600 GOTO 150

```

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
60 READ A$: PRINT @ 256, A$
70 FOR Y = 1 TO (360/WPM) * 460 : NEXT Y
80 REM      Y LOOP SETS LINES/MIN
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 WERE, 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

```

Music Composer

```

10 INPUT "LENGTH(1-10)"; M
20 M = M*4
30 INPUT "TEMPO (1-4)"; T1
40 IF T1 = 4 THEN G0
50 T = T1 : GOTO 70
60 T = 8
70 FOR K = 1 TO M*8
80 GOSUB 1000
90 B = RND(3) * T
100 SOUND P, B
110 CLS(S)
120 NEXT K
130 IF RND(10) <= 8 THEN 150
140 SOUND 125, 16*T
145 END
150 SOUND 90, 16*T
160 END
1000 X = RND(100)
1010 IF X <= 20 AND X <= 25 THEN P = 90 : S = 1
1020 IF X > 20 AND X <= 25 THEN P = 100 : S = 2
1030 IF X > 25 AND X <= 40 THEN P = 125 : S = 3
1040 IF X > 40 AND X <= 55 THEN P = 133 : S = 4
1050 IF X > 55 AND X <= 75 THEN P = 147 : S = 5

```

```

1060 IF X > 75 AND X <= 85 THEN P = 159 : S = 6
1070 IF X > 85 AND X <= 95 THEN P = 176 : S = 7
1080 IF X > 95 THEN P = 58 : S = 8
1090 RETURN

```

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

```

APPENDIX I

Error Messages

- | | |
|----|--|
| /0 | Division by zero. You asked the computer to divide a number by 0, which is impossible. |
| AO | Attempt to open a data file that is already open. |
| BS | Bad subscript. The subscripts in an array are out of range. Use DIM to dimension the array. 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 will get this error. |
| CN | Can't continue. You are using the CONT command and are at the end of the program. |
| DD | Attempt to redimension an array. You can dimension an array only once. For example, you cannot have DIM A(12) and DIM A(50) in the same program. |
| DN | Device number error. You can use only three devices 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 can be a result of loading a program with no line numbers. |

- FC Illegal Function Call. You used a parameter (number) with a BASIC word that is out of range. For example, SOUND (260,260) or CLS(10) causes this error. Also RIGHT\$(S\$,20), when S\$ contains only 10 characters, causes the error. Other examples are a negative subscript, such as A(-1), or a USR call before the address has been poked in.
- FD Bad file data. You printed data to a file or input data from a file, using the wrong kind 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. You attempted to INPUT data from a file OPEN for OUTPUT (O), or PRINT data into a file OPEN for INPUT (I).
- ID Illegal direct statement. You can use INPUT only as a program line, not a command line.
- IE Input past end of file. Use EOF to check to see when you've reached the end of the file. When you have, close it.
- IO Input/Output error. Often this is caused by trying to input a program or a data file from a bad tape.
- LS String too long. A string may be a maximum of 255 characters.
- NF NEXT without FOR. NEXT is being used without a matching FOR statement. This error also occurs when you have the NEXT lines reversed in a nested loop.
- NO File not open. You cannot 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. Use CLEAR at the beginning of your program to reserve more string space.
- OV Overflow. The number is too large for the computer to handle.
- RG RETURN without GOSUB. A RETURN line is in your program with no matching GOSUB.
- SN Syntax error. This could result from a misspelled command, incorrect punctuation, open parenthesis, or an illegal character. Type the program line or command again.
- ST String formula too complex. A string operation was too complex to handle. Break up the operation into shorter steps.
- TM Type Mismatch. This occurs when you try to assign numeric data to a string variable (A\$ = 3) or string data to a numeric variable (A = "DATA").

UL Undefined line. You have a GOTO, GOSUB, or other branching line in the program asking the computer to go to a nonexistent line number.

APPENDIX J

BASIC Summary

Statements

BASIC statements are commands that tell your computer to do some action, such as printing a message on the screen. Use BASIC statements as lines in your program.

AUDIO Connects or disconnects cassette output to TV speaker.

CLEAR *n,h* Reserves *n* bytes of string storage space. Erases variables. *h* specifies highest BASIC address.

CLOAD Loads specified program file from cassette. If you do not specify *filename*, the first file encountered is loaded. *Filename* can be a maximum of 8 characters.

CLOADM Loads machine-language program from cassette. You may specify an offset address to add to the loading address.

CLOSE#*dev* Closes access to specified file. If you do not specify *device*, all open files are closed.

CLS *c* Clears display to specified color *c*. If you do not specify color, green is displayed.

CONT Continues program execution after you have pressed **BREAK** or used the STOP statement.

CSAVE Saves program on cassette (program name can be a maximum of 8 characters). If you specify A, program is saved in ASCII format.

DATA Stores data in your program. Use READ to assign data to variables.

DIM Dimensions one or more arrays.

END Ends program.

EXEC (*address*) Transfers control to machine-language programs at specified address. If you omit address, control is transferred to address set in last CLOADM.

FOR . . . TO STEP/NEXT Creates a loop in program that the computer must repeat from the first number to the last number you specify. Use STEP to specify how much to increment the number each time through the loop. If you omit STEP, the computer uses 1.

GOSUB Calls a subroutine beginning at specified line number.

GOTO Jumps to specified line number.

IF test THEN . . . action 1 ELSE action 2 Performs a test. If it is true, the computer executes *action 1*. If it is false, then the computer executes *action 2*.

INPUT Causes the computer to stop and await input from the keyboard.

INPUT#-1 Inputs data from cassette.

LIST Lists (displays) specified line(s) or entire program on screen.

LLIST Lists specified program line(s) or entire program to printer.

MOTOR Turns cassette ON or OFF.

NEW Erases everything in memory.

ON . . . GOSUB Multiway branch to call specified subroutines.

ON . . . GOTO Multiway branch to specified lines.

OPEN m,#dev,f Opens specified file (*f*) for data transmission (*m*) to specified device (*dev*). *m* may be I (Input) or O (Output). *dev* may be #0 (screen or keyboard), #-1 (cassette), or #-2 (printer).

POKE location, value Puts value (0-255) into specified memory location.

PRINT Prints (displays) specified message or number on TV screen.

PRINT # dev, data list Prints data list to specified buffer. (See OPEN.) To separate items within data list, use either commas or semicolons.

PRINT #-1 Writes data to cassette.

PRINT #-2 Prints an item or list of items on the printer.

PRINT TAB Moves the cursor to specified column position.

PRINT @ scr pos Prints specified message at specified text screen location.

READ Reads the next item in DATA line and assigns it to specified variable.

REM Allows insertion of comment in program line. The computer ignores everything after REM.

RESET (X, Y) Resets a point.

RESTORE Sets the computer's pointer back to first item on the first DATA line.

RETURN Returns the computer from subroutine to the BASIC word following GOSUB.

RUN Executes a program.

SET (X,Y,C) Sets a dot at specified text screen position.

SKIPF Skips to next program on cassette tape or to end of specified program.

SOUND tone, duration Sounds specified tone for specified duration.

STOP Stops execution of a program.

Functions

BASIC functions are built-in subroutines that perform some kind of computation on data, such as computing the absolute value of a number. Use BASIC functions as data within your program lines.

ABS (numeric) Computes absolute value.

ASC (str) Returns ASCII code of first character of specified string.

CHR\$ (code) Returns character for ASCII, control, or graphics code.

EOF (dev) Returns FALSE = 0 if there is more data; TRUE = -1 if end of file has been read.

INKEY\$ Checks the keyboard and returns the key being pressed (if any).

INT (numeric) Converts a number to an integer.

JOYSTK (j) Returns the horizontal or vertical coordinate (*j*) of the right or left joystick:

0 = horizontal, right joystick

1 = vertical, right joystick

2 = horizontal, left joystick

3 = vertical, left joystick

LEFT\$ (str, length) Returns left portion (length characters) of a string.

LEN (str) Returns the length of a string.

MEM Finds the amount of free memory.

MID\$ (str,pos,length) Returns a substring of another string starting at pos. If you omit length, the entire string right of position is returned.

PEEK (mem loc) Returns the contents of a specified memory location.

POINT (S,Y) Tests whether specified graphics cell is on or off. x (horizontal)=0-63; y (vertical)=0-31. The value returned is -1 if the cell is in a text character mode; 0 if it is off; or the color code if it is on. See CLS for color codes.

RIGHT\$ (str,length) Returns right portion of string.

RND(n) Generates a "random" number between 1 and *n* if *n* > 1, or between 0 and 1 if *n* = 0.

SGN (numeric) Returns sign of specified numeric expression: -1 = negative; 0 = 0; +1 = positive.

SIN (numeric) Returns sine of angle given in radians.

STR\$ (numeric) Converts a numeric expression to a string.

USR (numeric) Calls user's machine-language subroutine starting at the address 275,276 (MSB,LSB).

VAL (str) Converts a string to a number.

Operators

BASIC operators perform some kind of operation on data, such as adding two numbers.

- , +	Unary negative, positive
* , /	Multiplication, division
+ , -	Addition and concatenation, subtraction
< , > , = , <= , >= , <>	Relational tests
NOT	
AND	
OR	



INDEX

- \$ See strings
- ; See print punctuation
- , See print punctuation
- BREAK** 20
- SHIFT** **O** 11
- ?/O ERROR 10
- ?LS ERROR 54
- ?OS ERROR 53
- ?SN ERROR 10
- ?TM ERROR 14
- ABS 68
- alphabetizing See sorting
- analyzing 117
- AND 67
- answers to exercises 155
- arrays 105
- arrays, multidimensional 117
- ASC 87
- ASCII See ASC and character codes
- Asteroids, program 81
- AUDIO 83
- BASIC Summary 180
- black on green 11
- Blinking Computer, program 75
- BLKIN 143
- BLKOUT 144
- Boolean algebra 130
- Boolean operators AND, OR, NOT 130
- Card Dealing, program 121
- character codes
 - listing 153
 - use of 87
- CHR\$ See character codes
- CHROUT 144
- CLEAR 54, 132, 141
- CLOADM 141
- CLOSE 100
- CLS 10
- color codes
 - reference 150
 - use of 10
- concatenate (+) 53
- CONT 65
- correcting See error
- Craps, program 41
- Dancing Computer, program 91
 - data
 - sorting 114
 - storing on tape 99
- Deal the Cards, program 108
- deleting, program line 20
- Display Control Register 131
- Display Mode Selection, table 136
- division (/) 9
- Do-It-Yourself Programs See answers to exercises
- Drawing Board, program 124
- E notation 69
- Electronic Piano, program 59
- END 35
- error
 - messages 178
 - program line 20
 - typographical error 7
- exponents 69
- FOR 24
- functions, BASIC 182
- games 38
- general-purpose subroutines 128
- GETKEY 141
- GIVABF 142
- GOSUB 48
- graphics
 - character codes 88, 150
 - strings 89
 - modes 124
 - modes, table 136
- graphics screen location
 - grid 152
 - use of 73
- green on black 12
- IF 35
- information See data
- INKEY\$ 59
- INPUT 19
- joysticks 76
- JOYSTK 76
- LEFT\$ 54
- LEN 53
- LIST 18
- LLIST 113
- loops 30-39
- machine-language subroutines 141
- mapping functions 134
- MEM 65
- memory 13
- MID\$ 55
- mistake, correcting 7
- MOTOR 83
- multiplication (*) 9
- musical tones
 - reference 149
 - use of 33
- nested loop 31
- NEXT 24
- numbers 9
- numeric data 15
- numeric, arrays 105
- ON GOSUB 66
- ON...GOTO 67
- OPEN 99
- OR 67
- Page-Select Register 131
- Painting, program 78
- parentheses, rules on 51
- PEEK 124, 130
- pixel 129
- plus (+), addition 53
- POINT 80
- POKE 124, 130, 141
- PRINT 8

PRINT @ locations
 grid 277
 use of 40
 PRINT punctuation, rules 21
 PRINT #-1 100
 PRINT #-2 112
 printer, use of 112
 prompt 7
 RESET 75
 resolution 129
 RESTORE 44
 RETURN 48
 reversed colors 11
 reversed colors, **SHIFT****O** 113
 RIGHTS\$ 54
 RND 38
 Rolling the Dice, program 40
 ROM subroutines 143
 RUN 18
 Russian Roulette, program 39
 sample programs 168
 SET 72
 SGN 68
 singing 33
 sorting 114
 SOUND 11, 27
 statements, BASIC 180
 STEP 26
 STOP 65
 STR\$ 68
 STRING DATA 15
 string, arrays 110
 string(s) 9, 14
 subroutines
 descriptions 48
 ROM routines 143
 general purpose 128
 subscripted variables 105
 Talking-Computer Teacher, program 83
 taping 99
 technical information 123-45
 THEN 35
 Three Blind Mice, program 34
 tone 11
 Typing Test, program 62
 USR 142
 variables 19-22
 variables, subscripted 105
 VDG 131
 Video Display Generator Register 131
 VIDEO RAM 130
 VIDEO RAM Page Selection, table 137
 Vocabulary, program 43
 Voting Tabulation, program 105
 Word Processing, program 112
 Writing an Essay, program 111
 WRTLDR 144

RADIO SHACK, A DIVISION OF TANDY CORPORATION

U.S.A.
FORT WORTH, TEXAS 76102

CANADA
BARRIE, ONTARIO, L4M4W5

TANDY CORPORATION

AUSTRALIA
91 KURRAJONG AVENUE
MOUNT DRUITT, N.S.W. 2770

BELGIUM
PARC INDUSTRIEL DE NANINNE
5140 NANINNE

UNITED KINGDOM
BILSTON ROAD, WEDNESBURY
WEST MIDLANDS WS10 7JN