

Radio Shack®

**GETTING
STARTED
WITH
COLOR
BASIC**

TRS-80® COLOR COMPUTER 2



TERMS AND CONDITIONS OF SALE AND LICENSE OF RADIO SHACK COMPUTER EQUIPMENT AND SOFTWARE PURCHASED FROM A RADIO SHACK COMPANY-OWNED
COMPUTER CENTER, RETAIL STORE OR FROM A RADIO SHACK FRANCHISEE OR DEALER AT ITS AUTHORIZED LOCATION

LIMITED WARRANTY

I. CUSTOMER OBLIGATIONS

- A. CUSTOMER assumes full responsibility that this Radio Shack computer hardware purchased (the "Equipment"), and any copies of Radio Shack software included with the Equipment or licensed separately (the "Software") meets the specifications, capacity, capabilities, versatility, and other requirements of CUSTOMER.
- B. CUSTOMER assumes full responsibility for the condition and effectiveness of the operating environment in which the Equipment and Software are to function, and for its installation.

II. RADIO SHACK LIMITED WARRANTIES AND CONDITIONS OF SALE

- A. For a period of ninety (90) calendar days from the date of the Radio Shack sales document received upon purchase of the Equipment, RADIO SHACK warrants to the original CUSTOMER that the Equipment and the medium upon which the Software is stored is free from manufacturing defects. THIS WARRANTY IS ONLY APPLICABLE TO PURCHASES OF RADIO SHACK EQUIPMENT BY THE ORIGINAL CUSTOMER FROM RADIO SHACK COMPANY-OWNED COMPUTER CENTERS, RETAIL STORES AND FROM RADIO SHACK FRANCHISEES AND DEALERS AT ITS AUTHORIZED LOCATION. The warranty is void if the Equipment's case or cabinet has been opened, or if the Equipment or Software has been subjected to improper or abnormal use. If a manufacturing defect is discovered during the stated warranty period, the defective Equipment must be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer for repair, along with a copy of the sales document or lease agreement. The original CUSTOMER'S sole and exclusive remedy in the event of a defect is limited to the correction of the defect by repair, replacement, or refund of the purchase price, at RADIO SHACK'S election and sole expense. RADIO SHACK has no obligation to replace or repair expendable items.
- B. RADIO SHACK makes no warranty as to the design, capability, capacity, or suitability for use of the Software, except as provided in this paragraph. Software is licensed on an "AS IS" basis, without warranty. The original CUSTOMER'S exclusive remedy, in the event of a Software manufacturing defect, is its repair or replacement within thirty (30) calendar days of the date of the Radio Shack sales document received upon license of the Software. The defective Software shall be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer along with the sales document.
- C. Except as provided herein no employee, agent, franchisee, dealer or other person is authorized to give any warranties of any nature on behalf of RADIO SHACK.
- D. Except as provided herein, RADIO SHACK MAKES NO WARRANTIES, INCLUDING WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
- E. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation(s) may not apply to CUSTOMER.

III. LIMITATION OF LIABILITY

- A. EXCEPT AS PROVIDED HEREIN, RADIO SHACK SHALL HAVE NO LIABILITY OR RESPONSIBILITY TO CUSTOMER OR ANY OTHER PERSON OR ENTITY WITH RESPECT TO ANY LIABILITY, LOSS OR DAMAGE CAUSED OR ALLEGED TO BE CAUSED DIRECTLY OR INDIRECTLY BY "EQUIPMENT" OR "SOFTWARE" SOLD, LEASED, LICENSED OR FURNISHED BY RADIO SHACK, INCLUDING, BUT NOT LIMITED TO, ANY INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR OPERATION OF THE "EQUIPMENT" OR "SOFTWARE". IN NO EVENT SHALL RADIO SHACK BE LIABLE FOR LOSS OF PROFITS, OR ANY INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY BREACH OF THIS WARRANTY OR IN ANY MANNER ARISING OUT OF OR CONNECTED WITH THE SALE, LEASE, LICENSE, USE OR ANTICIPATED USE OF THE "EQUIPMENT" OR "SOFTWARE".
- NOTWITHSTANDING THE ABOVE LIMITATIONS AND WARRANTIES, RADIO SHACK'S LIABILITY HEREUNDER FOR DAMAGES INCURRED BY CUSTOMER OR OTHERS SHALL NOT EXCEED THE AMOUNT PAID BY CUSTOMER FOR THE PARTICULAR "EQUIPMENT" OR "SOFTWARE" INVOLVED.
- B. RADIO SHACK shall not be liable for any damages caused by delay in delivering or furnishing Equipment and/or Software.
- C. No action arising out of any claimed breach of this Warranty or transactions under this Warranty may be brought more than two (2) years after the cause of action has accrued or more than four (4) years after the date of the Radio Shack sales document for the Equipment or Software, whichever first occurs.
- D. Some states do not allow the limitation or exclusion of incidental or consequential damages, so the above limitation(s) or exclusion(s) may not apply to CUSTOMER.

IV. RADIO SHACK SOFTWARE LICENSE

RADIO SHACK grants to CUSTOMER a non-exclusive, paid-up license to use the RADIO SHACK Software on one computer, subject to the following provisions:

- A. Except as otherwise provided in this Software License, applicable copyright laws shall apply to the Software.
- B. Title to the medium on which the Software is recorded (cassette and/or diskette) or stored (ROM) is transferred to CUSTOMER, but not title to the Software.
- C. CUSTOMER may use Software on one host computer and access that Software through one or more terminals if the Software permits this function.
- D. CUSTOMER shall not use, make, manufacture, or reproduce copies of Software except for use on one computer and as is specifically provided in this Software License. Customer is expressly prohibited from disassembling the Software.
- E. CUSTOMER is permitted to make additional copies of the Software only for backup or archival purposes or if additional copies are required in the operation of one computer with the Software, but only to the extent the Software allows a backup copy to be made. However, for TRSDOS Software, CUSTOMER is permitted to make a limited number of additional copies for CUSTOMER'S own use.
- F. CUSTOMER may resell or distribute unmodified copies of the Software provided CUSTOMER has purchased one copy of the Software for each one sold or distributed. The provisions of this Software License shall also be applicable to third parties receiving copies of the Software from CUSTOMER.
- G. All copyright notices shall be retained on all copies of the Software.

V. APPLICABILITY OF WARRANTY

- A. The terms and conditions of this Warranty are applicable as between RADIO SHACK and CUSTOMER to either a sale of the Equipment and/or Software License to CUSTOMER or to a transaction whereby RADIO SHACK sells or conveys such Equipment to a third party for lease to CUSTOMER.
- B. The limitations of liability and Warranty provisions herein shall inure to the benefit of RADIO SHACK, the author, owner and/or licensor of the Software and any manufacturer of the Equipment sold by RADIO SHACK.

VI. STATE LAW RIGHTS

The warranties granted herein give the original CUSTOMER specific legal rights, and the original CUSTOMER may have other rights which vary from state to state.

Getting Started with Color BASIC:
Copyright © 1981 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 the 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 obtained herein.

TRS-80 Color BASIC System Software: Copyright ©
1980 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 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.

Printed in the United States of America

10 9 8 7 6 5 4 3

WELCOME NEWCOMERS!

If you don't know anything about Computers and would like to be spared the long, technical explanations, relax — this book is for you!

Using this as your guide, you'll be able to interact and enjoy your Computer *right away*. The first section is all you need to get going. The rest of the book is frills.

You'll find that , especially at first — we'll have you doing a lot of games, songs, and other fun-type programs. Don't worry — if you want to do “practical” programs, you'll find plenty of that later. We start you off with the fun programs because that's the quickest way for you to feel at ease with your Computer. Once you feel it's truly an extension of yourself, you'll be able to make it do most anything you want.

So sit down and spend a couple of hours with it. Type away at it. Play with it. Try to make it do strange things. In other words . . . get to feeling comfortable with it. There's an endless number of things it can do for you.

. . . *AND HELLO OLD-TIMERS*

We haven't forgotten about you. If you already know how to program, turn to Appendix J. There, you'll find a summary of COLOR BASIC with page numbers you can refer to for the things you want to know more about. Then, if you want to learn more about your Computer, go straight to Section IV. It'll show you how to program high resolution graphics and call machine-language programs.

TO GET STARTED . . .

Connect your Computer by referring to the Chapters on "Installation" and "Operation", and to Figure 1 in your *TRS-80 Color Computer Operation Manual*.

Then power up your Computer:

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

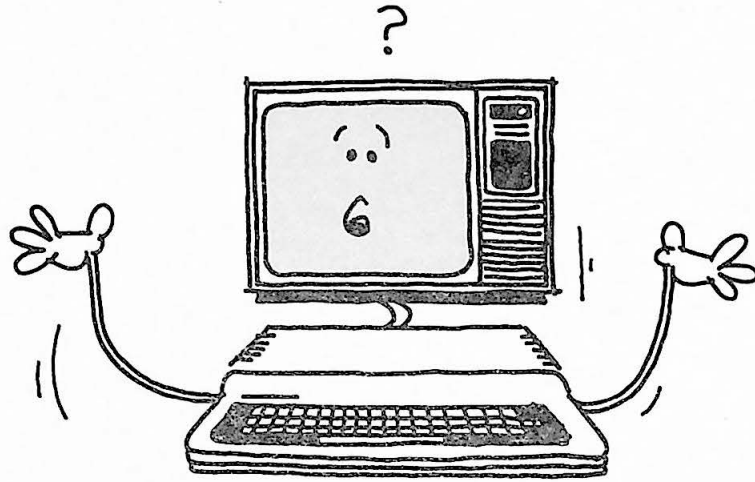
This message should appear:

```
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 T.V. set. Check all the connections. If you still don't get this message, refer to "Troubleshooting and Maintenance" in the *TRS-80 Color Computer Operations Manual*.

Once you do get this message, you're ready to begin.



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 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 mid-way in the book, you might forget what one of the words means. If this happens, simply look up the word in the back of the book or use your "Quick Reference Card" to find its meaning.

TABLE OF CONTENTS

SECTION I — GETTING THE HANG OF IT

CHAPTER 1: MEET YOUR COMPUTER	6
CHAPTER 2: YOUR COMPUTER NEVER FORGETS (. . . unless you turn it off . . .)	16
CHAPTER 3: SEE HOW EASY IT IS	24
CHAPTER 4: COUNT THE BEAT	34
CHAPTER 5: SING OUT THE TIME	42
CHAPTER 6: DECISIONS, DECISIONS	54
CHAPTER 7: GAMES OF CHANCE	60
CHAPTER 8: SAVE IT ON TAPE	70
CHAPTER 9: COLOR YOUR SCREEN	76
CHAPTER 10: ONE FANTASTIC TEACHER	90
CHAPTER 11: HELP WITH MATH	102
CHAPTER 12: A GIFT WITH WORDS	112
CHAPTER 13: BEAT THE COMPUTER	124
CHAPTER 14: POLISH IT OFF	134

SECTION II — GRAPHICS WITH PIZZAZZ

CHAPTER 15: MOVING PICTURES	148
CHAPTER 16: THE TALKING COMPUTER TEACHER	158
CHAPTER 17: GAMES OF MOTION	166
CHAPTER 18: FASTER THAN MOTION	174
CHAPTER 19: LET'S DANCE	186

SECTION III — GETTING DOWN TO BUSINESS

CHAPTER 20: KEEPING TABS ON EVERYTHING	196
CHAPTER 21: PUT POWER IN YOUR WRITING	208
CHAPTER 22: TAPE YOUR BOOK COLLECTION (. . . or your records, Christmas list, tax receipts, inventory . . .)	218
CHAPTER 23: FILING — AS EASY AS ABC	232
CHAPTER 24: GETTING ANALYTICAL	238

SECTION IV — DON'T BYTE OFF MORE THAN YOU CAN CHEW

PART A: HIGH RESOLUTION GRAPHICS	252
PART B: USING MACHINE-LANGUAGE SUBROUTINES	267
PART C: MEMORY MAP	271

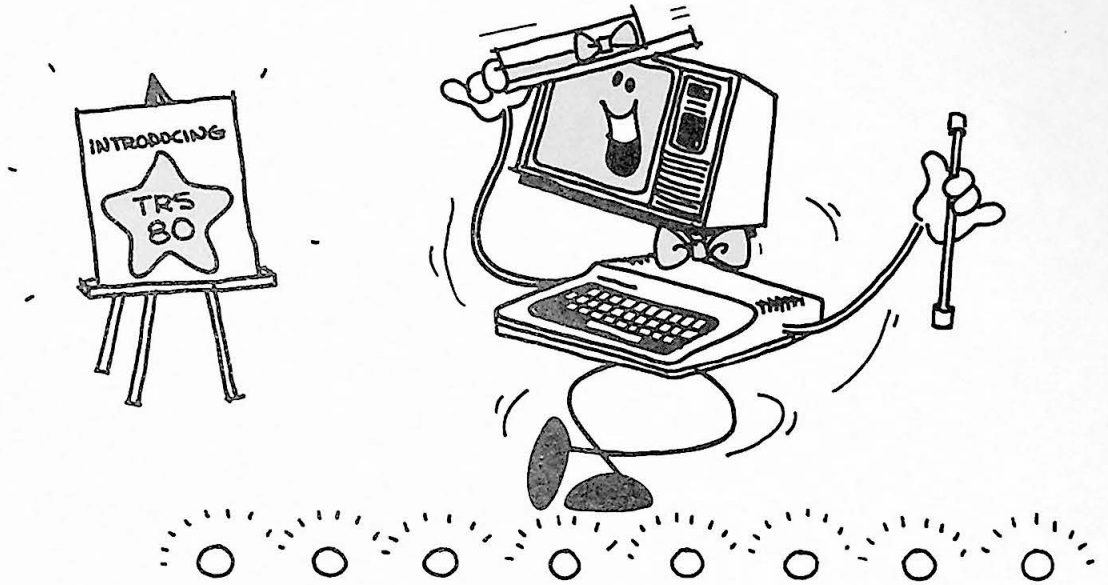
APPENDIXES

A / MUSICAL NOTES	274
B / BASIC COLORS AND GRAPHICS	276
C / PRINT @ SCREEN LOCATIONS	277
D / GRAPHICS SCREEN LOCATIONS	278
E / ASCII CHARACTER CODES	280
F / ANSWERS TO EXERCISES	282
G / SUBROUTINES	287
H / SAMPLE PROGRAMS	291
I / ERROR MESSAGES	298
J / BASIC SUMMARY	300

SECTION I

**GETTING THE
HANG OF IT**

CHAPTER 1



MEET YOUR COMPUTER



MEET YOUR COMPUTER

Is your Computer connected? Turned on? Ready to give it a first work-out?

In these first two Chapters, we're going to introduce you to your Computer — the way it thinks, some of its many talents, and even a couple of little quirks it has. By the time you finish these chapters, you'll be ready to program . . . *promise!*

Type away on the keyboard and then press the **(ENTER)** key.

Don't worry about anything but the last line of type on your screen. It should say:


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* are the Master — you can tell the Computer to do anything you wish.

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

PRINT "HI, I'M YOUR COLOR COMPUTER"

When you reach the end of the line on your screen, keep on typing. The last part of the message will appear on the next line.



*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.*

See the blinking light? You can type something wherever you see it.

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

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 had in quotes. Give it another message to print. 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 a lot more than just an electronic mimic . . . like maybe



"Hi, I'm Your Color Computer!"

some answers! Well, try it without the quotation marks. Type:

PRINT 2 + 2 (ENTER)

Much better. This time the Computer prints the answer:

4

These quotation marks obviously must mean something. Try experimenting some more with them. 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)

Have you come up with any conclusions on what the quotes do?



The Computer thinks of quotes like a journalist does. If the message is 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.

RULES ON STRINGS VS 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 will figure it out like a numerical problem.

A COLOR CALCULATOR, NO LESS!

Any arithmetic problem is a snap for your Computer. Let it do some long division. Type:

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

Let's do a multiplication problem:

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

Notice that the Computer's multiplication sign is an asterisk, rather than the X sign which you've always used in math. This is because the Computer is such a precise and literal creature 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)
```

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

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

DO-IT-YOURSELF COMMAND LINES

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

If you used "correct" command lines, this is what the Computer should have printed 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*:

```
PRIINT "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". Anytime you get a SN error, it's probably because you made some kind of typographical error.

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

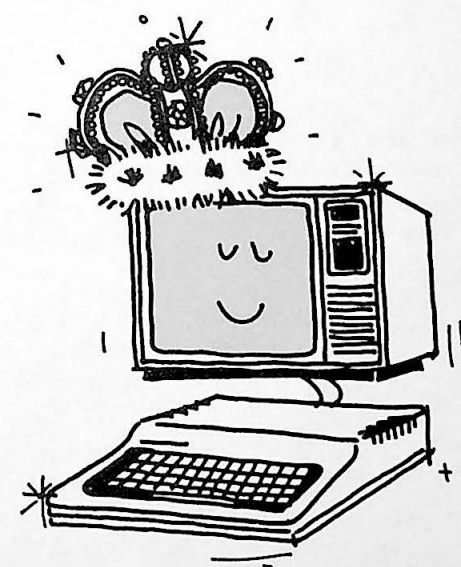
```
PRINT 5/0 ENTER
```

The Computer prints:

```
?/0 ERROR
```



Actually there is 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!



Which means “Don’t ask me to divide by 0 — that’s impossible!!”

If you get a strange error message you don’t understand, flip back 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 typed command told the Computer to clear the screen and print color number 3 — blue.

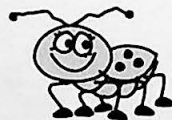
But why the green stripe? The Computer cannot type on a blue background. Anytime it types something on the screen, it must type it on a green background. Try typing some more characters. Notice that the Computer gives these characters a green background also.

Colors other than green are for printing graphics illustrations. We’ll spend lots more time with this color capability later.

Press **(ENTER)** so that you get the OK prompt on your screen. Type:

CLS(7) **(ENTER)**

Now you should have magenta (pinkish purple) on your screen with a green stripe at the top. Try some more colors if you like. Use any number from 0 to 8. Your color Computer has nine colors. Each color has a numeric code.



BUG: If you get a message saying MICROSOFT or an ?FC Error message, it’s because you are using a number other than 0 through 8.

If you don’t get the right colors, refer to the color test in your Owner’s Manual.

Type CLS without a number code:

CLS **ENTER**

If you don't use a number code, the Computer assumes you just 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 are 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's got quite a hum-range . . . hope you're suitably impressed. Try some other numbers. Hope you like the Computer's voice (it's the only one it's got).

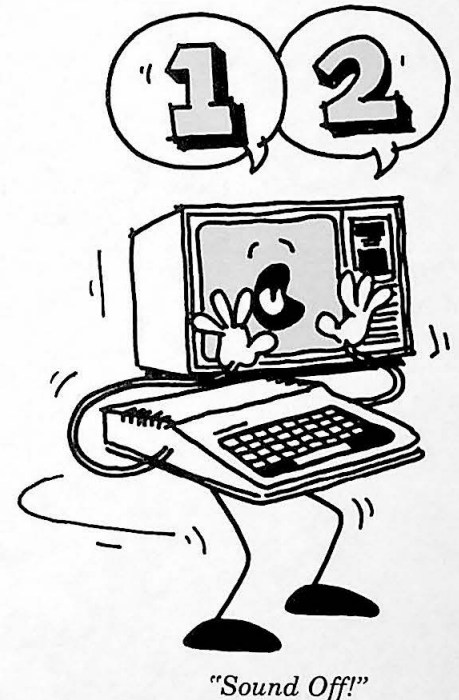
You want to know what the other number is for? (Or maybe you've already found out). The second number tells the Computer *how long* to hum the tone. You can use any number from 1 to 255. Try 1:

SOUND 128, 1 **ENTER**

and the Computer will hum 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 stick to numbers between 1 and 255.





BUG: Again, if you get an ?FC Error message, it's because you are using a number other than 1 through 255.

ONE MORE THING . . .

Curious about the reversed colors? They're for people with a printer. The printer will print everything typed in reversed colors as lower case letters.

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

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


PRINT "HI" **(ENTER)**

The Computer gives you an ?SN ERROR. It doesn't understand the command.

Press the **(SHIFT)** and **(0)** characters again and 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'll work.

We just wanted to show you this in case you ever press **(SHIFT)** and **(0)** by a mistake. The computer can't understand any commands you type with reversed colors. If you ever find you're typing with these reversed colors, press the **(SHIFT)** and **(0)** keys to get the colors back to normal.

LEARNED IN CHAPTER 1

BASIC WORDS	KEYBOARD CHARACTERS	CONCEPTS
PRINT SOUND CLS		string vs. numbers error messages

We'll put a list like this at the end of each chapter. It'll help you make sure you didn't miss anything.

NOTES:

? = Print

* = mult.

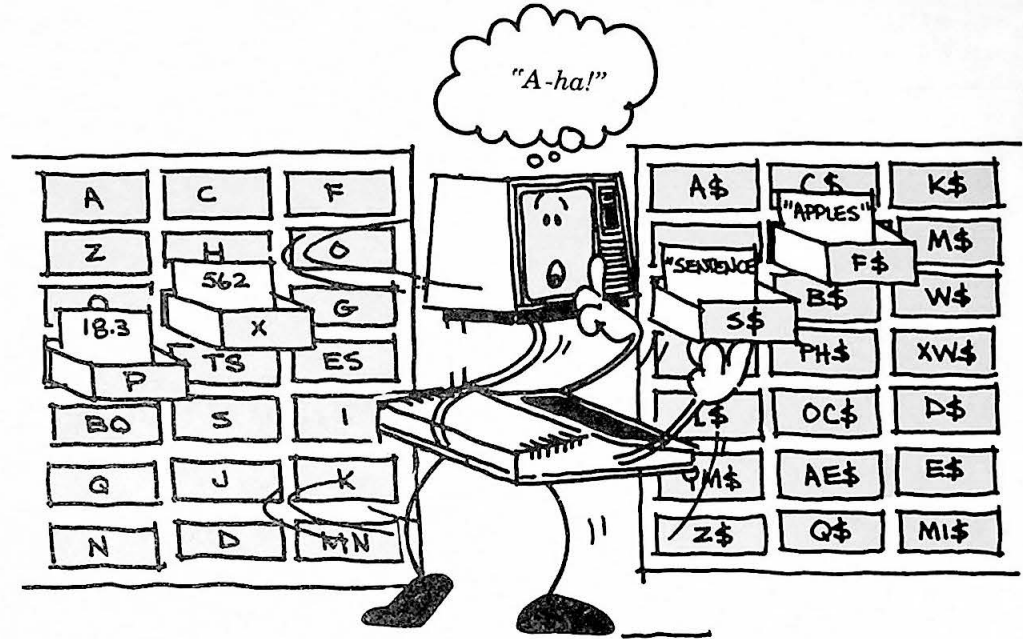
/ = div.

- = sub.

+ = add.

Sn. = Syntax

CHAPTER 2



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



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

One of the things that makes your Computer so powerful is its ability to remember anything you ask it to. To make the Computer remember the number 13, type this:

A = 13 **(ENTER)**

Now type anything you want to confuse the Computer. When you're done, press **(ENTER)**. To see if the Computer remembers what A stands for, type:

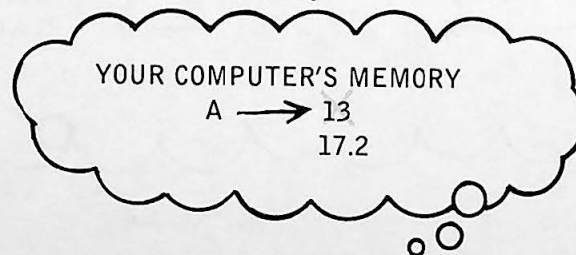
PRINT A **(ENTER)**

Your Computer will remember 13 as long as you have it *on* . . . or until you do what we're going to do next. Type:

A = 17.2 **(ENTER)**

Now if you ask it to PRINT A, it will print the number 17.2.

This is what just happened in your Computer's memory:



*Did it get confused?
or forget?*

If you already know BASIC, you might be accustomed to using the word LET before these command lines. Your Color Computer thinks that word is unnecessary and is confused when you use it.



You don't have to use the letter A. You may use any letters from A to Z. (As a matter of fact, you can use any *two* letters). Try typing this:

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

Have it print all your numbers. Type:

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

To get it to remember a *string* of letters or numbers, put a dollar sign next to the letter. Type:

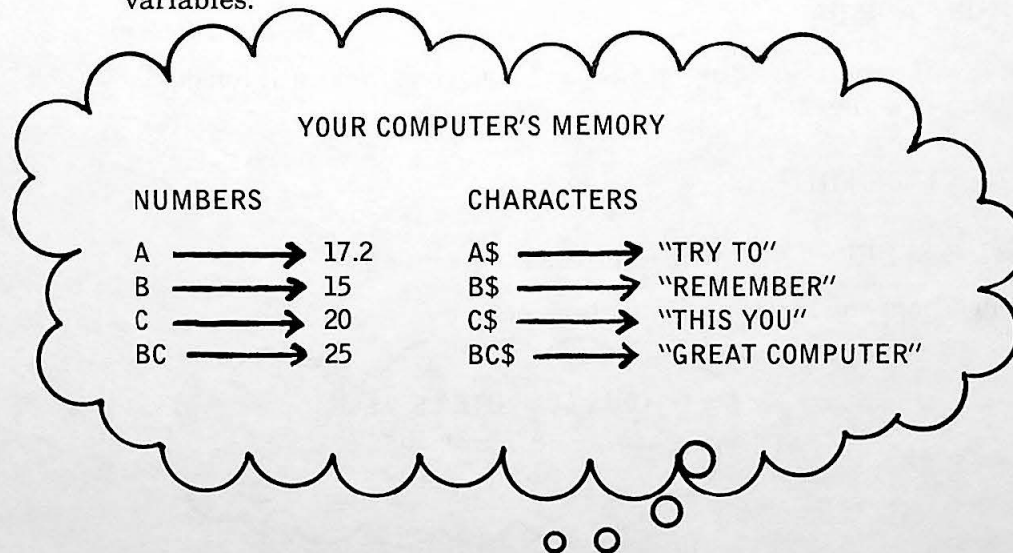
```
A$ = "TRY TO"
B$ = "REMEMBER"
C$ = "THIS YOU"
BC$ = "GREAT COMPUTER"
```

To the Computer, a dollar sign means it's a string.

Let's see how sharp your Computer is. Type:

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

Computer types call all these letters *variables*. So far, we've used these variables:



Try spot checking these variables to see if the Computer has remembered your information properly. For instance, type:

```
PRINT BC (ENTER)
```

To see if BC still contains 25.

You can think of these variables as little boxes where you can store your information. One set of boxes is for *strings*; the other set's for *numbers*. You use these variables to label each box.

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

THE COMPUTER IS FUSSY ABOUT ITS RULES

Do you think the Computer will accept these lines:

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

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

With both of these lines, the Computer responds with ?TM ERROR. It's telling you you've got to play according to *its rules*.

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

These are the rules you ignored:

RULES ON STRING DATA

- (1) Any data in quotes is *STRING DATA*
- (2) *STRING DATA* may only be assigned to variables *WITH A \$ SIGN*

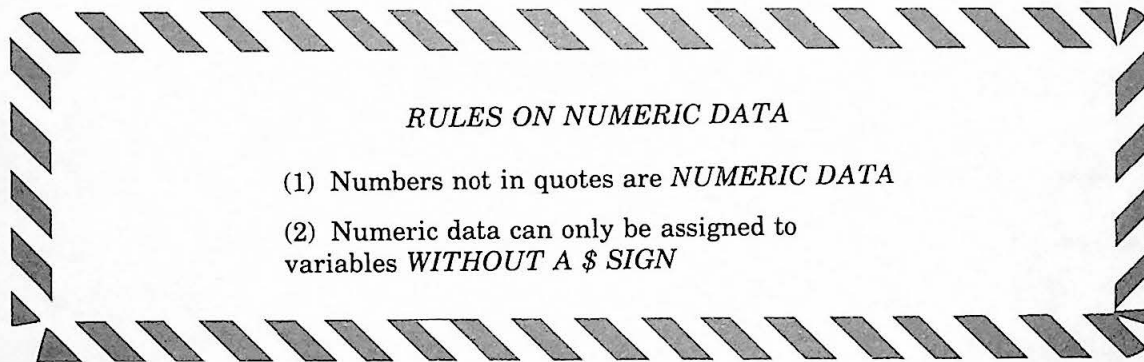
To obey the Computer's rules, we have to put a dollar sign after D and Z. Type:

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

which the Computer accepts.
Do you think the Computer will accept this?

D\$ = 6 (ENTER)

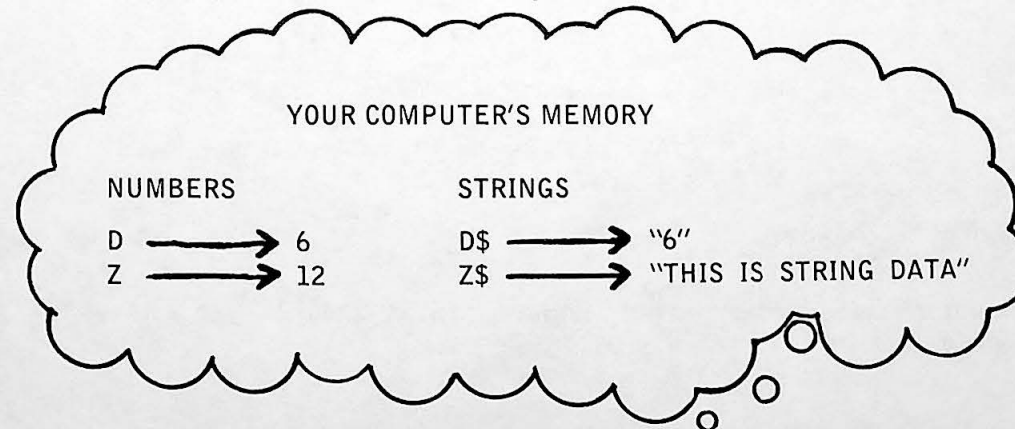
These are the rules that this command ignored:



Type this, which the Computer will accept:

D = 6 (ENTER)
Z = 12 (ENTER)

You have now added this to your Computer's memory.



Now you can do something interesting with these letters. Type:

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

The Computer prints the product of D times 2.

Try this line:

```
PRINT Z/D
```

The Computer prints the quotient of Z divided by D.

Would this work:

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

Did you try it? This makes the Computer print the same ?TM ERROR. It cannot multiply string data.

Cross out the commands that the Computer will reject:



The computer remembers that D = 6.

EXERCISE WITH VARIABLES

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

Finished? This is what the Computer will *accept*.

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

RULES ON VARIABLES

You may use any two characters from A-Z for a variable. The first character must be a letter from A-Z; however, the second may be either a numeral or letter. If you want to assign it string data, put a dollar sign after it. Otherwise, it can only hold numeric data.

LEARNED IN CHAPTER 2

CONCEPTS

Variables
String vs. Numeric Variables

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

NOTES:

Lined writing area with horizontal lines on both sides of the page.



CHAPTER 3



SEE HOW EASY IT IS?



SEE HOW EASY IT IS?

Type:

```
NEW (ENTER)
```

This is just to erase anything that might be in the Computer's "memory".

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

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

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

```
RUN (ENTER)
```

The Computer obediently runs your program. Type RUN again and again to your heart's content. The magic machine will run your program anytime you wish, as many times as you wish.

Since that worked so well, let's add another line to the program. Type:

```
20 PRINT "WHAT IS YOUR NAME?"
```

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 that SN Error. The Computer didn't understand what you meant when you typed your name. In fact, the Computer can't understand anything unless you talk to it in its own way.

So let's 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 will label 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:

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

```
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:

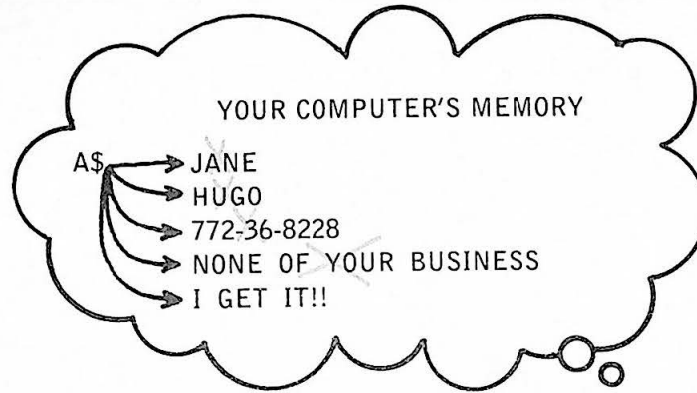
```
HI, I'M YOUR COLOR COMPUTER
WHAT IS YOUR NAME?
? HUGO
HI, HUGO
```

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

```
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 want to 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):



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 told the Computer to go back up 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 will now run perpetually, because every time it hits line 50, the Computer goes up to line 10 again. 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 so we can give your name the kind of attention it deserves. How do we change a program line? Simply by typing it over again, using the same line number. Type:

```
50 GOTO 40
```



To delete a program line, simply type and **(ENTER)** the line number. For example:

50 **(ENTER)**
 erases line 50 from the program.

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. Press the **(BREAK)** key when you've seen enough.

There's a big change we 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:

```
40 PRINT A$;
```

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

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

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

RULES ON PRINT PUNCTUATION

This is how punctuation at the end of a PRINT line makes the Computer PRINT:

- (1) a *COMMA* makes the Computer *PRINT* in columns.
- (2) a *SEMICOLON* makes the Computer cram the *PRINTing* together.
- (3) *NO PUNCTUATION* makes the Computer *PRINT* in rows.

COLOR/SOUND DEMONSTRATION

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

In this program we are 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.

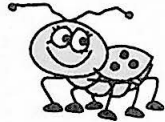


Let's play around some more with your Computer's sound and color abilities. First clean out its memory. Remember how?

Now 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 this program to get a sampling of some of the Computer's tones.



BUG: If you get a ?FC Error when you run this program, it's because you used a number other than 1 through 255. This error, like all errors, will make the Computer stop RUNning the program.

What would happen if we changed line 40 to:

```
40 SOUND 50, T
```

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

Did you figure it out? By making this change, the Computer hums the same tone every time, but hums it for a different length of time, depending on the number you type in.

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 color you ask for. Remember, there are 9 colors, 0 through 8.

DO-IT-YOURSELF PROGRAM

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

This is our program:

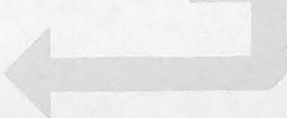
```
10 PRINT "TO MAKE ME CHANGE MY COLOR"  
20 PRINT "TYPE A NUMBER BETWEEN 0 AND 8"  
30 INPUT T  
40 CLS(T)  
50 GOTO 10
```

ADD POLISH TO THE PROGRAM

Professional programmers would think that pressing the **(BREAK)** key was a rather sloppy way of getting the program to stop running. Why not get the Computer to politely ask us if we are ready to end it? Change Line 50 in the above program to:

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

Press **(BREAK)** before typing the line.



and add these lines:

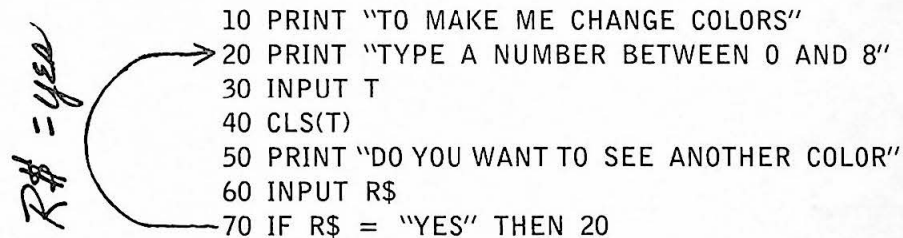
```
60 INPUT R$
70 IF R$ = "YES" THEN 20
```

and RUN the program . . . Type YES and the program will keep on running. Type anything else and the program will stop.

This is what the program looks like:

```
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
```

R\$ = YES



Let's look at what these new lines did:

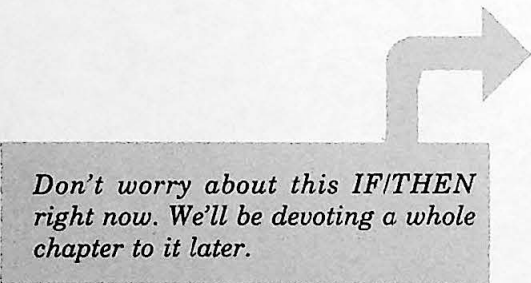
Line 50 simply printed a question.

Line 60 told the Computer to stop and wait for our answer -- R\$.

Line 70 told the Computer to go back to line 20 *IF* (and only *IF*) your answer (R\$) was *YES*. If not, the program simply ended since there are no more lines in the program.

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

Don't worry if you don't understand everything perfectly yet. Just enjoy using your Computer.



Don't worry about this IF/THEN right now. We'll be devoting a whole chapter to it later.

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

NOTES:

Pg. 27- neatt!

CHAPTER 4



COUNT THE BEAT



COUNT THE BEAT

In this Chapter we are going to do some experimenting with Computer sound effects. To do this, we have to first teach the Computer how to count.

Type this:

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

RUN the program.

RUN the program several times, each time replacing 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 are making the Computer do? They are making it count. Let's study the last program we suggested you try:

The logic of this will become clear later.

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


```

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

```

Line 10 tells the Computer that the first number should be 20 and the last number should be 24. It uses X to label these numbers.

Line 30 tells the Computer to keep going back up to line 10 for the next Number—the NEXT X—until it reaches the last number (24).

Look at line 20. Since line 20 is between the FOR and NEXT lines, the Computer must PRINT the value of X every time it counts:

```

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

```

Add another line between FOR and NEXT:

```

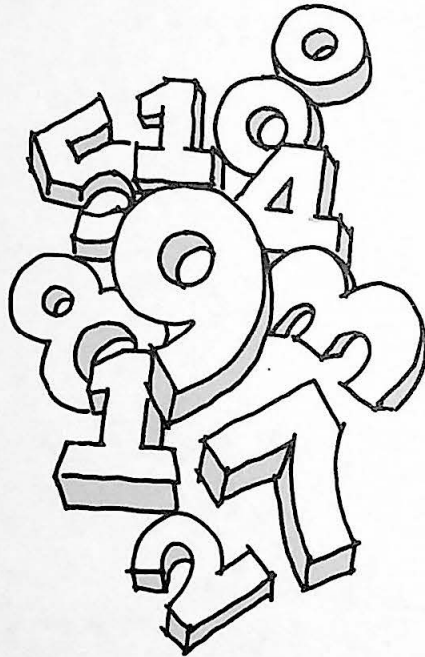
15 PRINT "... COUNTING ..."

```

and RUN it. With every count, your Computer executes any lines you choose to insert between FOR and NEXT.

Write a program which will make the Computer print your name 10 times.

DO-IT-YOURSELF PROGRAM 4/A



HINT: The program must count to 10.

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

DO-IT-YOURSELF PROGRAM 4/B

*HINT: PRINT 9*X is a perfectly legitimate program line.*

Write a program which will print the multiplication tables for 9*1 through 9*25.

DO-IT-YOURSELF PROGRAM 4/C

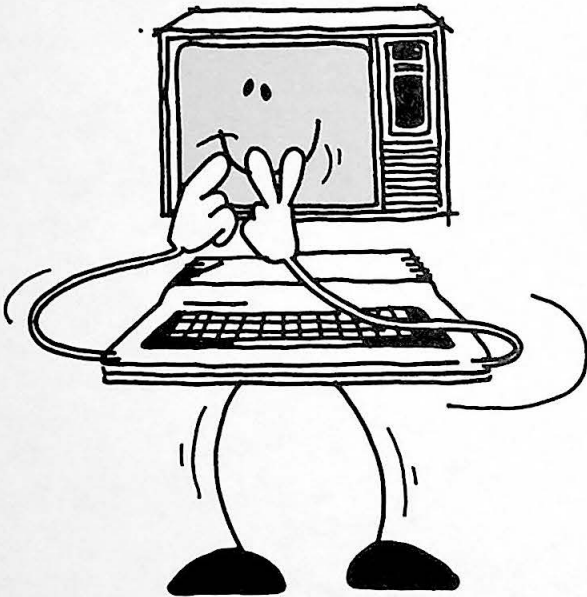
HINT: By adding a comma in the PRINT line, you can get all the problems and results on your screen at once.

Finished? These are our programs:

```
Program 4/A
10 FOR X = 1 TO 10
20 PRINT "THOMAS"
30 NEXT X
```

```
Program 4/B
10 FOR X = 1 TO 10
20 PRINT "9**X"="9*X
30 NEXT X
```

```
Program 4/C
10 FOR X = 1 TO 25
20 PRINT "9**X"="9*X,
30 NEXT X
```



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

COUNTING BY TWOS

Now we'll make it count a little differently. Erase your program by typing `NEW` and then type our 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"
```

RUN the program . . . Do you see what the `STEP 2` did? It makes the Computer count by 2's. Line 10 tells the Computer that:

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

To make the Computer count by 3's, make all the Xs 3 apart. Try this for line 10:

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

RUN the program. It should print this on your screen:

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


It passed up the last X (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:

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



This program is making the Computer count from 1 to 255 (by ones). Each time it counts it does what lines 20 and 30 tell it to do:

- It PRINTs X, the current count (Line 20)
- It SOUNDS X's particular tone (Line 30)

For example:

- the first time the Computer got to FOR, in line 10, it made X equal to 1.
- then it went to line 20 and printed 1, the value of X.
- then, line 30 had it SOUND tone #1.
- then it went back up to line 10 and made 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? Using STEP, change line 10 so the Computer will sound tones from:

You might be wondering about the programs we ran at the first of this Chapter where we didn't use STEP. If we leave out STEP, the Computer assumes we mean STEP 1.

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

- (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.

PROGRAMMING EXERCISE

10 _____
10 _____
10 _____

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

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
```

Now see if you can write a program which 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

DO-IT-YOURSELF PROGRAM

BUT CAN IT SING?

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



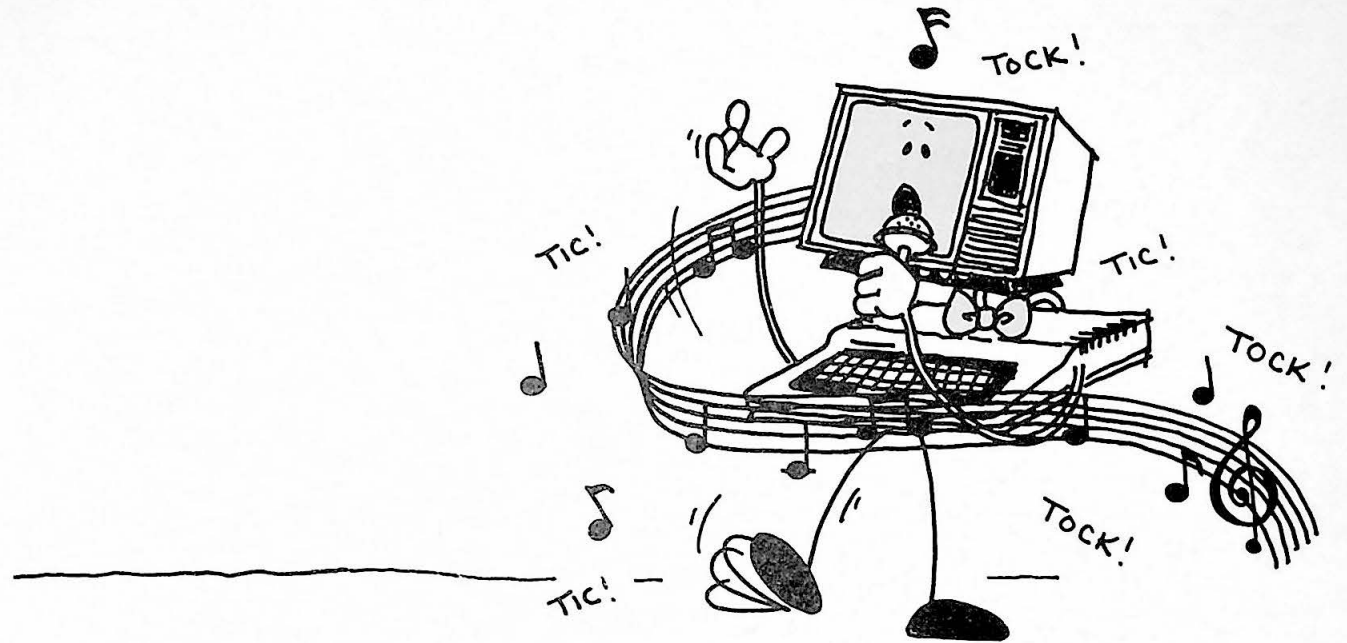
LEARNED IN CHAPTER 4

BASIC WORDS	KEYBOARD CHARACTER
FOR ... TO ... STEP NEXT	SHIFT @

NOTES:

Pg. 35 - neato!

CHAPTER 5



SING OUT THE TIME



SING OUT THE TIME

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

Begin 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 2 seconds to count to 920.

Lines 10 and 20 set a *timer pause* in your program. By making the Computer count to 920, it keeps the Computer busy for 2 seconds.

As you can see, this gives us the makings of a stopwatch. Erase the program, and type this:

```
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, inputting the number of seconds you want timed on your stopwatch.
It would be nice if the stopwatch could sound some kind of alarm. Add some lines to the end of the program to make it sound an alarm.

DO IT YOURSELF PROGRAM

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 we added at the end of the program. This is so the message would print and the alarm would keep ringing over and over again until the nervous programmer must press the **BREAK** or **SHIFT @** keys to turn it off.

COUNTING WITHIN THE TIME

Before we go any further on the clock, we're going to have the Computer keep count *within* the time. This concept will become very 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
```

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. *Every time* it counts X, it does these things:
 - A. It *PRINTs* the value of X

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




B. It counts Y from 1 to 2. *Every time* it counts Y, it does this:

(1) It *PRINTs* the value of Y

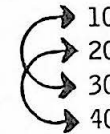
Whenever you put a loop inside another loop, you *must close* the inner loop before closing the outer loop:

Right



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

Wrong

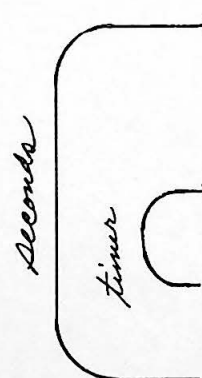


```
10 FOR X = 1 TO 3
20 FOR Y = 1 TO 2
30 NEXT X
40 NEXT Y
```

RELATING THIS TO A CLOCK

With these tools, we can make the Computer do a lot more. Type this:

*Notice that we changed the **TIMER PAUSE** in line 40 to 390. (in our previous program it was 460.) Because of all the extra things the program is doing, we had to adjust the timer to a lower number.*



```
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. *every time.*

it counts one second —

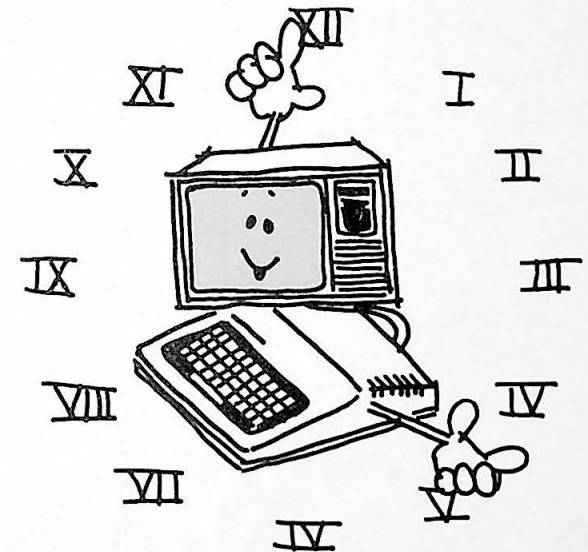
- A. It *PRINTs* the second
 - B. It *SOUNDs* a tone
 - C. It pauses long enough for one second to pass.
- II. When it finishes counting all the seconds from 0 to 59, it *PRINTs* a message that one minute is up.

There is a way we can make this program look a little better. Add this line which Clears the Screen:

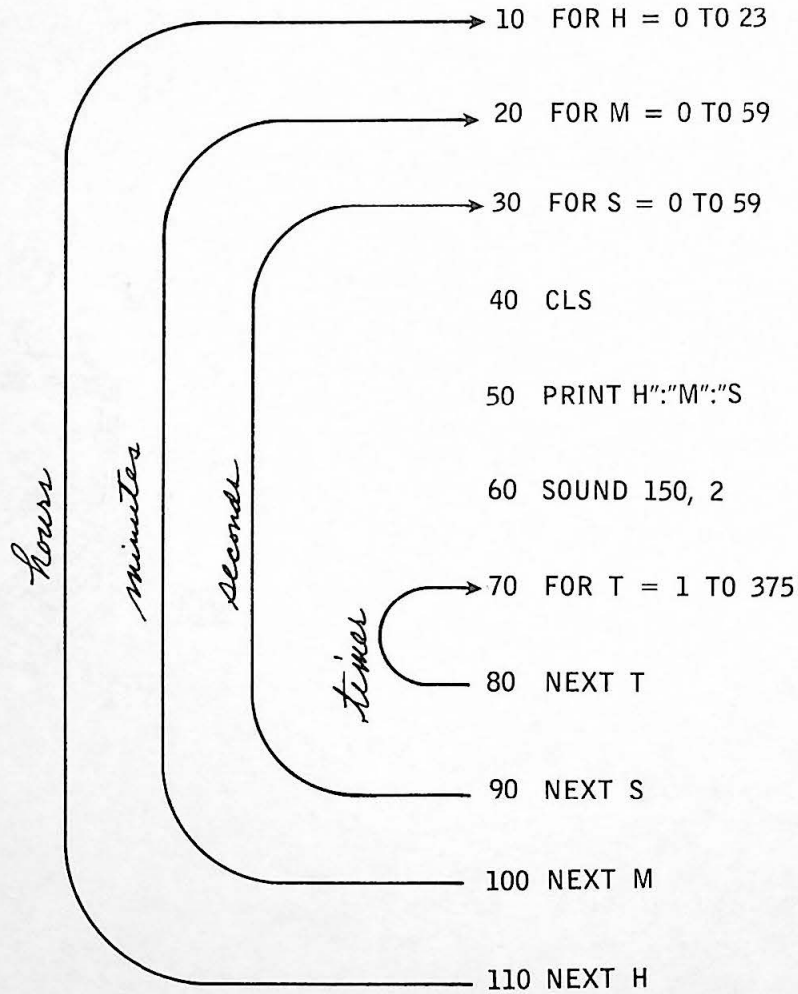
```
15 CLS
```

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).
Every time it counts the seconds.
 - 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).



With this groundwork, it is easy to make a full fledged clock:



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

I. It counts the hours from 0 to 23. (Line 10)

Every time it counts a new hour:

A. It counts the minutes from 0 to 59. (Line 20)

Every time it counts a new minute:

1. It counts the seconds from 0 to 59. (Lines 30 and 90)

Every time it counts a new second:

a. It *CL*ears the Screen. (Line 40)

b. It *PRINT*s the hour, minute, and second. (Line 50)

c. It *SOUND*s 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 up to line 20 for the next minute. (Line 100)

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

II. When it finishes counting all hours (0-23), the program ends.

Between lines 90 and 100 you can add some tones which will sound every minute. Write a program which does this.

*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

Write a program which makes your Computer show each of its nine colors for 1 second each:

DO-IT-YOURSELF PROGRAM

The answers to both of these programs are in the back.

But who said this Computer could make the Opera?

If you're a real music lover, you will probably want to purchase RADIO SHACK's "MUSIC" — Catalog number 26-3151. Then you will be able to compose songs on your Computer with perfect pitch.

FOR A COMPUTER, IT SINGS GREAT!

Now back to teaching your Computer how to sing. Flip back to the Appendix. We have a table, "Musical Tones", which shows the Computer's tone number for each note on the musical keyboard. For example, the Computer's tone number 89 corresponds to "middle C".

Unfortunately, your Computer can't exactly match most of the musical tones. That's why the Computer sings a little off key. . . But to those without perfect pitch, it can still sound very close to music.

Type this:

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

RUN the program. It is the first three notes of . . .well you know that, great piece!

To get these first three notes to play over again, we can 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? It's easy enough to put a *timer pause* in the program. Add these lines:

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

and RUN it again. Now it's beginning to sound like the real thing!

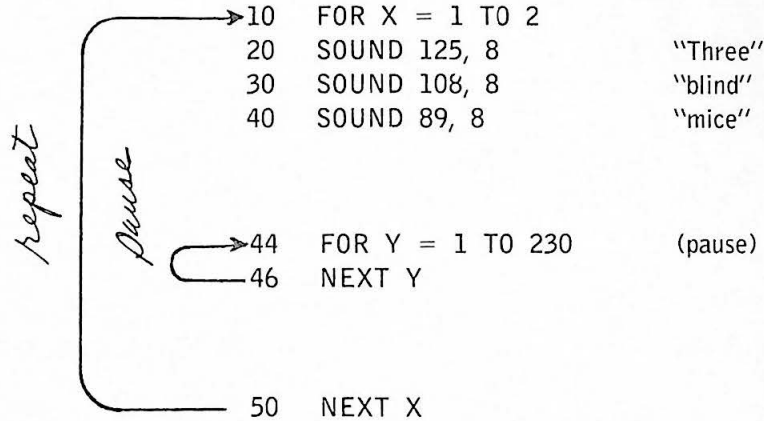


Here is a program that gets through the first two phrases:

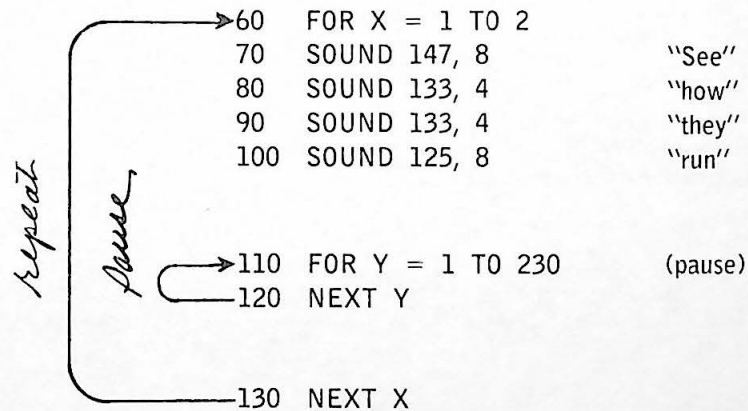
THREE BLIND MICE



Three blind mice



See how they run



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. Your Computer songs can certainly jazz up any program.

LEARNED IN CHAPTER 5

BASIC WORD

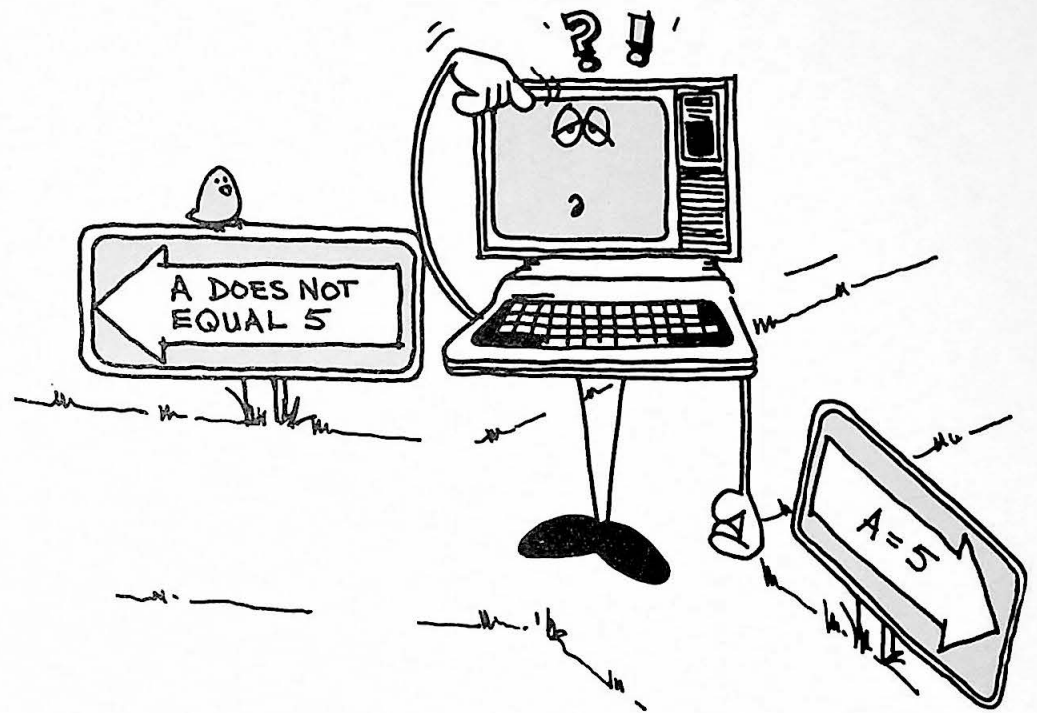
CLS

PROGRAMMING CONCEPT

Nested Loops

NOTES:

CHAPTER 6



DECISIONS, DECISIONS...



DECISIONS, DECISIONS. . .

Here's an easy decision for the Computer:

(1) IF you type RED . . . *THEN* make the screen red

. . . or

(2) IF you type BLUE . . . *THEN* make the screen blue

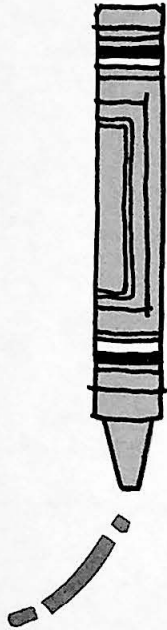
Easy enough? Let's make the Computer do it. Type this program:

```
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)
```

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 several times, typing both RED and BLUE.

Let's see what the program is doing:



IF you type RED ... *THEN* ...

Line 30 sends your program down to line 100. Line 100 makes your screen red. At this point, we have to stop the Computer from going on to line 200.

Line 110 does just that. It ends your program right there... Once the Computer gets to line 110, it will never make it to 200.

... On the other hand ...

IF you type BLUE ... *THEN* ...

Line 40 sends your Computer down to line 200, which makes your screen blue. We do not have to put END on the next line. Since line 200 is the last line in the program, the Computer will end there anyway.

What happens if you type something other than RED or BLUE? Try running the program, typing GREEN in response to the Computer's question.

It makes the screen RED, right? Do you know why?

HINT: IF the condition is not true, the THEN part of the line is ignored and the Computer proceeds to the next program line.

There are two lines you could add to make the Computer ask you to type your answer again if you don't type RED or BLUE. We will give you the two lines, and let you figure out where to put them in the program:
them in the program:

PROGRAMMING EXERCISE

```
.... 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.

Did you figure out where the two lines should go in the program? They must come after line 40 and before line 100:

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

See if you can make one more change to the program:

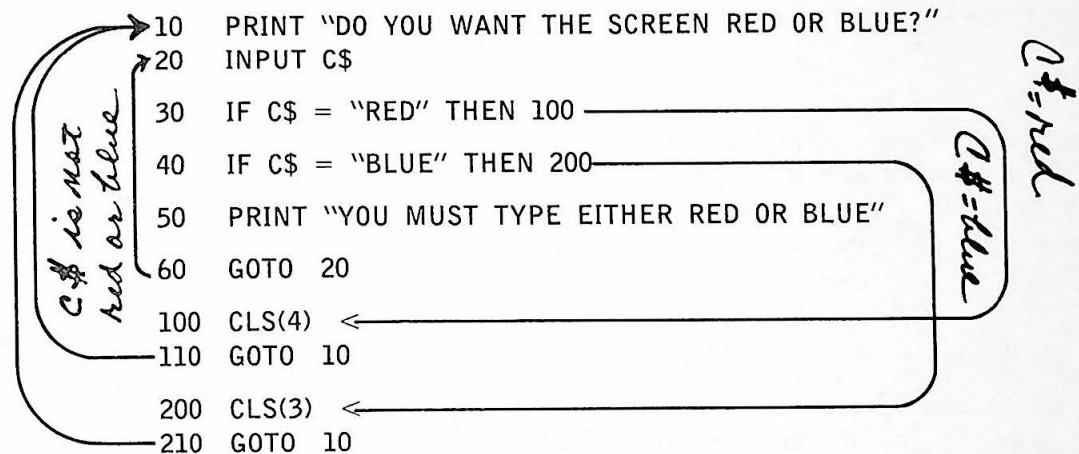
Instead of having the Computer end the program after it makes the screen red or blue, have it go back and ask you to type RED or BLUE again.



DO IT YOURSELF PROGRAM

HINT: You will need to change line 110 and add line 210.

Have you got a program written? Look on the next page for a diagram of ours.



To trace the path the Computer takes down this program, simply go down, from one line to the next, following the arrows when told to. 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.
 You only follow these arrows if the condition
 (C\$ = "RED" or C\$ = "BLUE") is true.

GOTO is unconditional.
 You follow these arrows whenever
 you arrive at a GOTO line.

Although this chapter was short, you've learned one of the most important programming concepts. We will be getting the Computer to make decisions all through the rest of this book.

LEARNED IN CHAPTER 6

BASIC WORDS

IF/THEN
END

NOTES:



CHAPTER 7



GAMES OF CHANCE



GAMES OF CHANCE

Thanks to a BASIC word called RND, your Computer can play almost any kind of game involving chance or luck. Even if you don't plan to play games with your Computer, you'll want to know how to use RND and PRINT @ — the words we're introducing in this Chapter. We'll also show you some more uses for IF/THEN.

Type this:

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

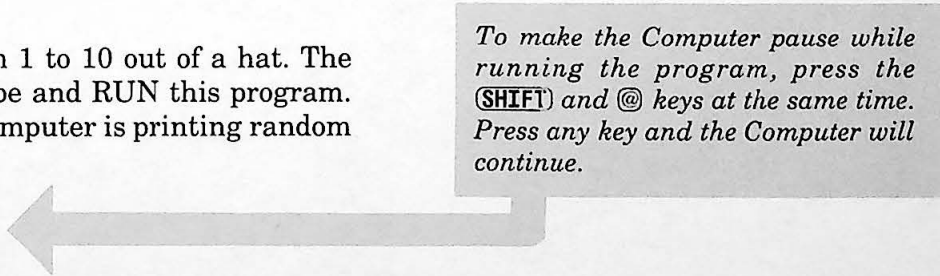
RUN it. The Computer just picked a random number from 1 to 10. RUN it some more times. . .

It's as if the Computer is drawing a number from 1 to 10 out of a hat. The number the Computer picks is unpredictable. Type and RUN this program. Press **(BREAK)** when you satisfy yourself that the Computer is printing random numbers:

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

To have the Computer pick random numbers from 1 to 100, change line 10 to this:

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



*To make the Computer pause while running the program, press the **(SHIFT)** and @ keys at the same time. Press any key and the Computer will continue.*

and RUN. How would you change this program to have the Computer pick a random number from 1 to 255?

.....

The answer is:

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

A COMPLETELY RANDOM SHOW

Just for the fun of it, let's have the Computer compose a song made up of random tones. Type this:

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

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


To add a random visual presentation to this program, add a couple of lines to make the Computer show a random color (1-8) just before it sounds each random tone.

DO IT YOURSELF PROGRAM



Here's our program:

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




We'll show you a couple of simple games in this Chapter. Feel free to use your imagination to add interest to them — or invent your own games.

RUSSIAN ROULETTE

In our “Russian Roulette” game, the gun has 10 chambers. The Computer picks, at random, which of the 10 chambers will have 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.

Now look at the arrows we drew:

IF X is equal to RND(10), THEN the Computer goes down to 100.

IF X is not equal to RND(10), THEN the Computer “clicks” and goes back up to line 10 where you get another chance. . .

Let's make the dead routine in line 100 better. Type:

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

Remember how to list a portion of a program?

LIST 50-130

lists the middle portion of the program.

This will also help you in listing a long program. Press the (SHIFT) and @ keys when the Computer first starts listing the program. The Computer will pause the "scrolling" on your display. Press any key to continue the listing.

```
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 happens in this program:

Lines 100 through 130 makes the Computer produce a sound of descending tones and print BANG!!!! over and over again on the screen.

Line 140 CLears the Screen. Since we did not choose a color number code, the Computer assumes we want the screen green.

Look at lines 150 and 170. Both of these lines use PRINT @. Here's the way PRINT @ works:

Notice the grid we have below, showing each of the 511 positions on your video screen. When writing the program, we wrote the two messages "SORRY, YOU'RE DEAD" and "NEXT VICTIM PLEASE" on this grid, positioning them where we wanted them on the screen.

SORRY, YOU'RE DEAD begins at location 230 (224 + 6). NEXT VICTIM PLEASE begins at location 390 (384 + 6). Using these numbers in the PRINT @ line, simply tells the Computer where we want the message printed.

```

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0
32
64
96
128
160
192
224      S O R R Y ,   Y O U ' R E   D E A D
256
288
320
352
384      N E X T   V I C T I M ,   P L E A S E
416
448
480

```

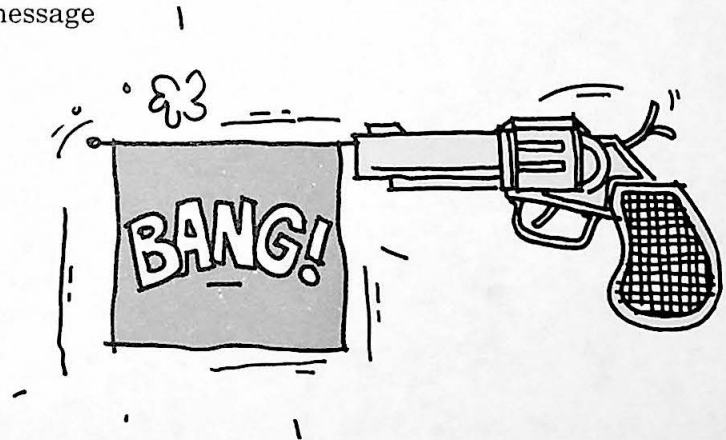
We put this grid in the Appendix of this book "PRINT @ Screen Locations". Use it in planning your programs, since good screen formatting can add a great deal of interest to your programs.

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:

```

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0
32
64
96
128
160
192
224      C O N G R A T U L A T I O N S ! ! !
256      Y O U   M A N A G E D
288      T O   S T A Y   A L I V E
320
352
384
416
448
480

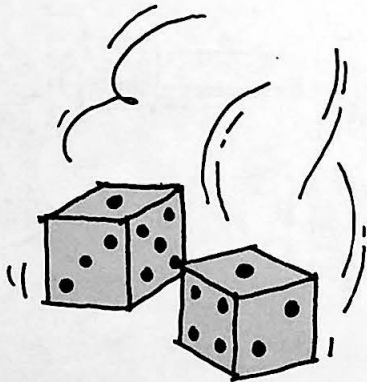
```



DO IT YOURSELF PROGRAM

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

Our answer to this is in Appendix F.



"Loser!"

ROLLING THE DICE

For our next game, we'll first have to teach the Computer to roll the dice. To do this, the Computer must roll *two* dice; that is, 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. Let's look at it:

Line 10 tells the Computer to CLeAr the Screen.

Line 20 has the Computer pick a random number from 1 to 6 for one of the die.

Line 30 has the Computer pick a random number for the other die.

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

Lines 50-70 PRINT the results of the roll on the screen.

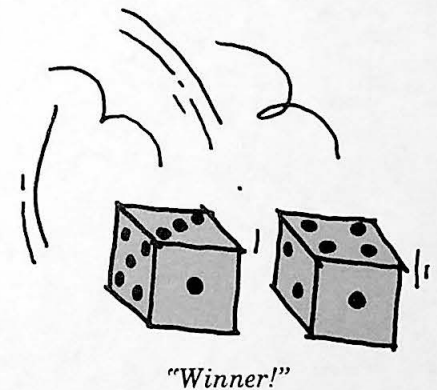
In line 90, you are able to INPUT whether you want the program to RUN again. IF you type YES, the Computer goes back to line 10 and runs the program again. Otherwise, since this is the last line in the program, the program ends.

CRAPS

Now that you know how to get the Computer to roll the dice, it should be fairly easy for you to write a Craps program. These are the rules of the game (in its simplest form):

1. The player rolls the two dice. If he rolls a sum of 2 ("snake eyes"), a 3 ("cock-eyes"), or a 12 ("boxcars") on the first roll, the player loses and the game is over.
2. If the player rolls a 7 or 11 on the first throw, ("a natural"), the player wins and the game is over.
3. If any other number is rolled on the first roll, it becomes the player's "point". He must keep rolling until he either "makes his point" by getting the same number again to win, or rolls a 7, and loses.

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 on what is happening. It may take you awhile to finish, but give it your best. *Good luck!*



DO-IT-YOURSELF PROGRAM

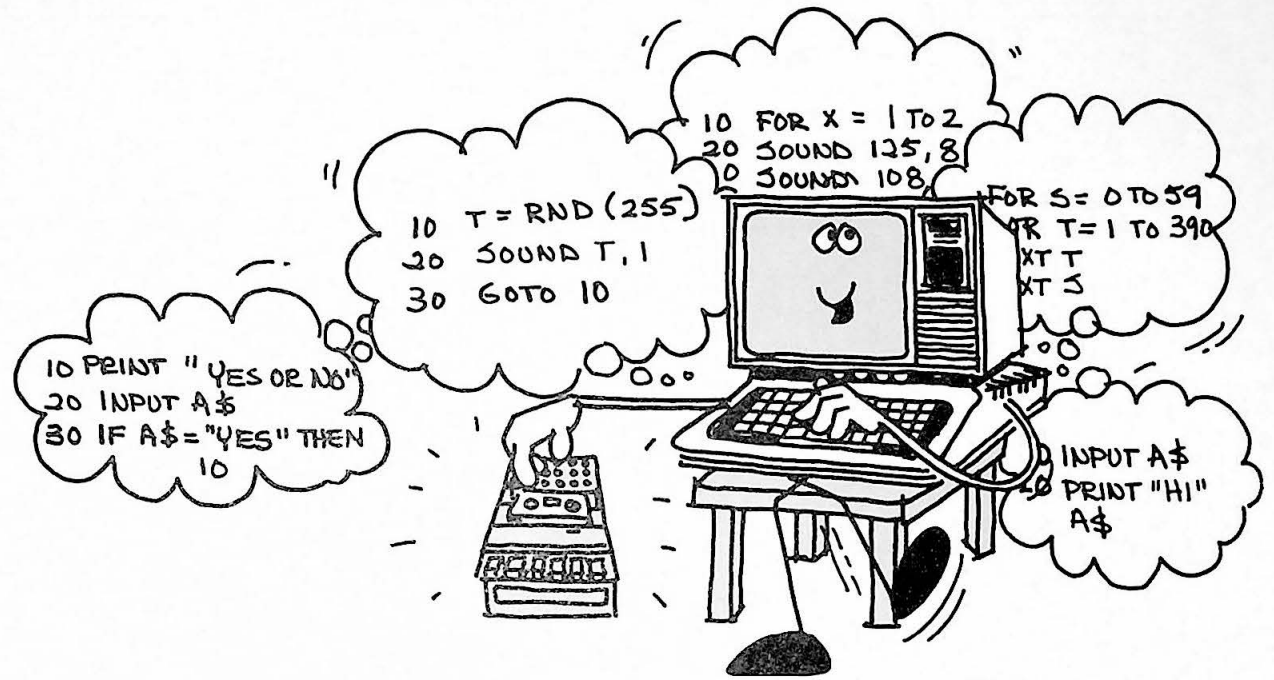
Our answer to this is in the back.

LEARNED IN CHAPTER 7

BASIC WORD

RND
PRINT @

CHAPTER 8



SAVE IT ON TAPE



SAVE IT ON TAPE

You'll soon be writing longer and more powerful programs. Perhaps you already are. It certainly cramps your style to have the program disappear everytime you turn the Computer off!

You can "save" (make a copy of) any of your programs on cassette tape. Once the program's on tape, you'll be able to "load" the program back into your Computer's memory anytime you want. We recommend that you use Radio Shack's CTR-80A cassette recorder (catalog number 26-1206) along with Radio Shack's Computer Tapes (catalog number 26-301).

This chapter is only for those of you that have this type of cassette recorder and want to use it. If you don't, you'll probably want to skip this chapter for now, remembering that the information's here whenever you need it.

Once you're used to it, you'll find cassette tape easy to use. Simply follow these steps:

A. Connect the Tape Recorder

1. Locate the CTR-80A Cassette Recorder, Interconnecting Cable and Radio Shack Computer Recording Tape cassette.
2. Connect the short cable between the TAPE jack on the back of the TRS-80 and your Cassette Tape Recorder

If you are using a different type of cassette recorder, the connections might be different from the explanation in this chapter.

If you are using a tape other than Radio Shack's, you need to position it after the plastic "leader" at the beginning of the tape.

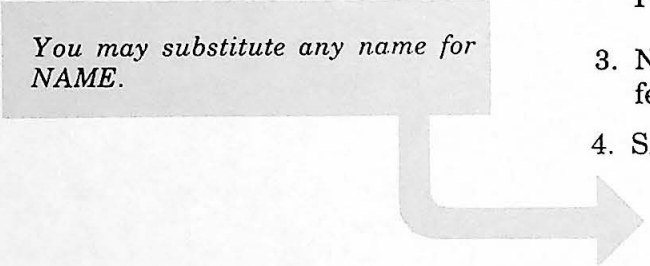
-
- The small grey plug goes into the REM jack on the Recorder.
 - The large grey plug goes into the AUX jack.
 - The black plug goes into the EAR jack.

3. Plug the Recorder into the wall outlet

B. Save a Program

1. Type any program into your Computer. RUN it to make sure it works.
2. Load the cassette tape, positioning it to the beginning of the tape. Press the PLAY and RECORD buttons *at the same time* until they lock.
3. Name the program you want to SAVE. You may use any name with 8 or fewer letters. For our example, we'll use "NAME".
4. SAVE on tape by typing this command:

You may substitute any name for NAME.



CSAVE "NAME" **(ENTER)**

The motor on the Recorder will start and you'll be recording the Computer's program on tape. Watch the screen. When:

OK

returns and the motor stops, your program is recorded on tape. It is also still in the Computer's memory. It has only been copied.

LOADING

Reversing the process and loading (copying) the program from tape into the Computer is just as easy:

1. Be sure the tape is fully rewound and the plugs are all in place.

2. Push the PLAY button down until it locks. Set the Volume Control to your CTR-80A's "Recommended Volume Level". Your CTR-80A Manual gives this recommended volume.
3. Type NEW to clear out any existing program.
4. Type the CLOAD command with the name of your program. For example:

CLOAD "NAME" **(ENTER)**

The Tape Recorder's motor will start. Watch your screen. The letter:

S

will appear at the top left hand corner. This means the Computer is Searching for your program. When the Computer has Found your program, it will print the letter F and the name of your program. For example, if your program name is NAME:

F NAME

will appear at the top of your screen. When the Computer prints:

OK

and the recorder motor stops, the program is "loaded" in memory. You may now RUN the program.

SAVING MORE THAN ONE PROGRAM

To SAVE more than one program on the same tape, you must make sure you are not recording on top of another program. This is an easy way to position the tape to the end of your last program:

1. Rewind the tape to the beginning.
2. Press the PLAY button until it locks

If you have several programs on tape, the Computer will print the name of each program it Finds on the tape preceding the one you want loaded.

If you try to load a program that's not on the tape, the Computer will not stop searching for it. Press the RESET button to stop searching.

-
3. Type SKIPF and the name of the last program on your tape. For example, if your last program is named "NAME", type:

SKIPF "NAME"

The Computer will notify you when it finds your program called NAME. When it reaches the end of NAME, the recorder's motor will stop and:

OK

will appear on your screen.

4. Once you've positioned the tape to the end of the last program, press the RECORD and PLAY buttons, name your program, and CSAVE it.

You may replace the name X with any name you know is NOT on the tape.

If you can't remember the name of your last program, type:

SKIPF "X"

and watch the screen. The Computer will give you the name of each program it encounters on the tape. It will print an I/O ERROR when it reaches the end of the tape, but don't worry about it. You've found what you were looking for — the name of the last program on the tape.

Now you can type the SKIPF command with the name of this last program. (Don't forget to rewind the tape first).

TIPS ON MAKING GOOD RECORDINGS

Here are some tips for making good recordings:

- When you're not using the Recorder for saving or loading, do not leave the RECORD or PLAY keys down. Press STOP.
- Don't attempt to re-record on a pre-recorded Computer tape. Even though the recording process erases the old recording, just enough information may

be left to confuse the new recording. If you want to use the same tape a second or third time, use a high-quality bulk tape eraser to be sure you erase everything.

- If you want to save a taped program permanently, break off the Erase Protect tab on the Cassette (see Tape Recorder's Manual). When the tab(s) has been broken off, you can't press the RECORD key on your Recorder. This will keep you from accidentally erasing that tape.

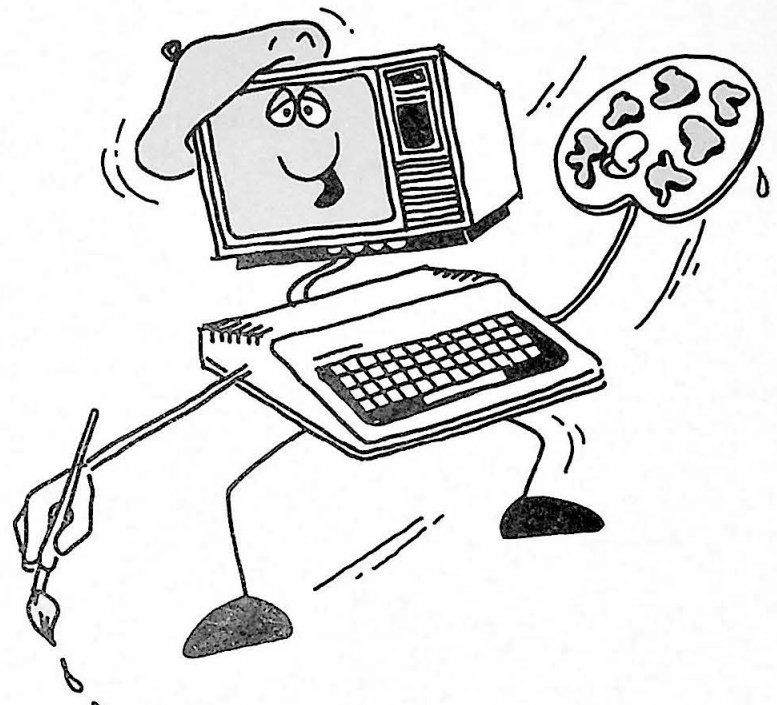
Now type as long of programs as you want, knowing you can make a permanent copy of them on tape. Happy recording!

LEARNED IN CHAPTER 8

BASIC WORDS

CLOAD
CSAVE
SKIPF

CHAPTER 9



COLOR THE SCREEN



COLOR THE SCREEN

You've learned enough now to really start using the colors. Since color graphics ideas usually come very quickly to people — and the good graphics programs usually end up long — this Chapter just shows you how to get started. While going through this Chapter, you'll probably want to stop from time to time and add on to our programs or build your own. We hope you do. That's a fast way to learn.

To get started, type:

```
10 CLS(0)
```

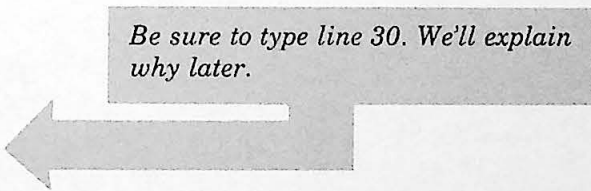
to make the screen black. Add these two lines and RUN the program:

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

Do you see the blue dot? It's at the top left-hand corner of your screen. To put the dot at the bottom right-hand corner, change line 20 and RUN the program:

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

Want to put it in the middle of the screen? RUN the program using this for line 20:



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

20 SET(31,14,3)

SET tells the Computer to SET a dot on your screen at a certain horizontal and vertical location.

- The first number you type is the horizontal location. This may be a number from 0 to 63.
- The second number is the vertical location. It may be a number between 0 and 31.

In the Appendix, there's a grid on your screen, "Graphics Screen Locations". The grid divides your screen into the 64 (0 to 63) horizontal locations and 32 (0 to 31) vertical locations. Use this grid in planning your graphics illustrations.

All of this explains what the first two numbers are for, but what about 3, the third number? Try using some numbers other than 3 for the third number. Type each of these lines and RUN the program:

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

Got it figured out? With number 4, you get a red dot, and with number 1 you get a green dot. The number codes are the same as the CLS number codes — 0 to 8. These are listed in your Appendix, "BASIC Colors".

Now, what's the GOTO line for? Try deleting the GOTO line from your program and RUN it:

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

It looks like no dot was SET this time. Actually the dot was SET, but when the program ended, the Computer printed its OK message on top of the dot.

To avoid this, type the GOTO line at the end of the program. It sets up an infinite loop (going to itself over and over again) so that the program will never end.

The computer uses different screen locations for SET than PRINT @. That's why we have two grids in the Appendix. Be sure to use the one we call "Graphics Screen Locations".

SETTING TWO DOTS

To SET more than one dot, you need to do a little planning. Erase your program and RUN this program:

```
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.

Now change the color of the right dot so you'll have one blue and one red dot. Type:

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

and RUN the program. . . *Both dots are red.*

Look again at the "Graphics Screen Locations" grid in your Appendix. Notice the darker lines group the dots together into blocks of four. For instance, the block in the middle of the grid contains these 4 dots:

	Horizontal	Vertical
Location	32	14
Location	33	14
Location	32	15
Location	33	15

Each dot within the block must either be:

1. the same color (colors 1-8)
- or
2. black

In our program, we tried to get the Computer to SET two dots with different colors — blue and red — within the same block. Since the Computer can't do that, it SETs both dots the second color — red.

Type this and RUN the program:



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

Since the dot in location 34, 14 is in a different block, the Computer can SET the two dots in different colors.

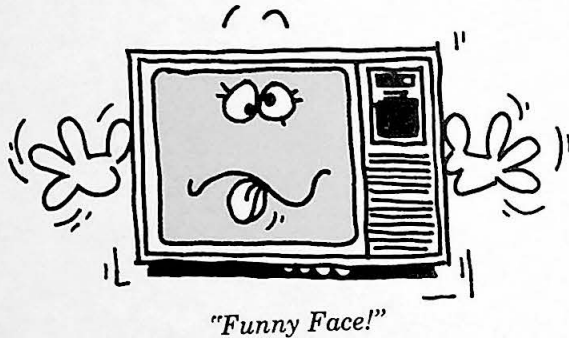
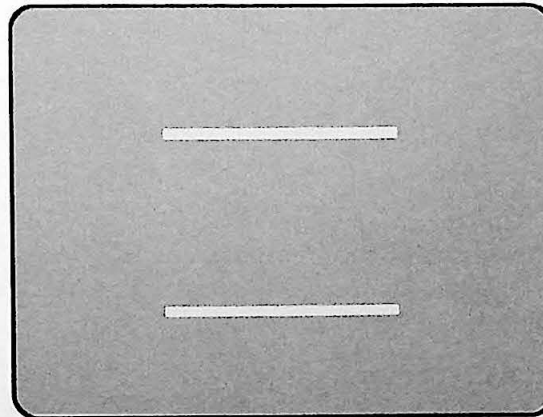
THE COMPUTER'S FACE

With this groundwork, 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
```

and RUN.

This is what you should have 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 locations 15 through 48 for the top and the bottom lines.

Line 20 SETs the top line. The horizontal location is 15 through 48 and the vertical location is 5.

Line 30 SETs the bottom line. The horizontal location, again, is 15 through 48 and the vertical location 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.

We'll make the nose orange. Type:

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

and the mouth red. Type:

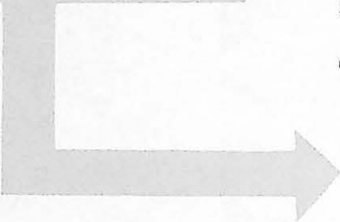
```
100 FOR H = 28 TO 36
110 SET(H,16,4)
120 NEXT H
```

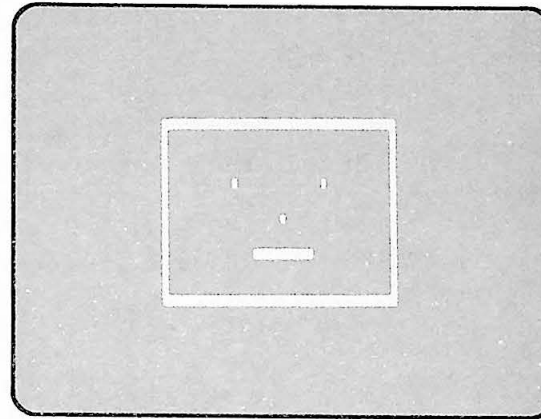
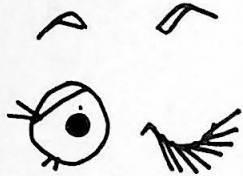
and blue eyes. Type:

```
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:

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





A BLINKING COMPUTER

*You don't need to tell the Computer the color of the dot to **RESET** (erase) it.*

By adding a couple of lines, we can make the Computer blink. Type:

```
150  RESET(38,10)
```

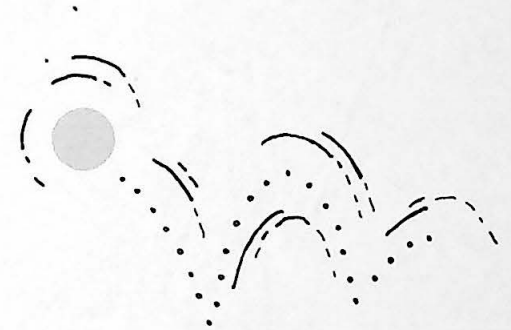
and RUN the program. What you should have on your screen now is the same face as above, except the right eye is missing. **RESET** tells the Computer to erase the dot in the horizontal location 38 and the vertical location 10. That's the right eye.

To make it blink, we'll simply **SET** and **RESET** the right eye over and over again, by adding line 160:

```
160  GOTO 140
```

LIST your program to see if it still looks like mine:

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



and RUN it . . . Try your hand at some pictures. I'm sure you have better artistic skills than we do.

THE BOUNCING DOT

By using SET and RESET, we can make a moving picture. Type and RUN these lines to make the dot go down:

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

Remember to always erase your program before typing a NEW one.

Every dot that is SET on line 20 is RESET (erased) on line 30. Add these lines to make the dot go back up:

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

and this line to make the dot go up and down, over and over again:

```
90 GOTO 10
```

and RUN it. To slow the dot down — it will look a little better — change lines 30 and 70:

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

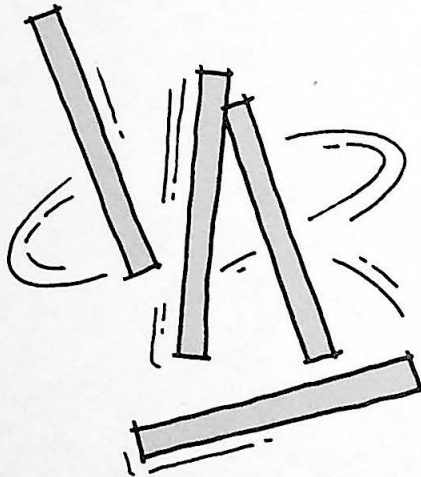
The > sign means the same as it does in math — *greater than*. The < sign means *less than*.

SET and RESET opens up all sorts of possibilities — moving targets, animated pictures, etc. Use your imagination in experimenting with this combination.

IF YOU HAVE THE JOYSTICKS...

If you have joysticks with your Computer, you have many more options open to you. If you haven't connected them yet, do it. Simply plug them in to the back of your Computer. They only fit in the correct slot, so don't worry about connecting them to the wrong one.

Now, type this short program which demonstrates how they 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

```

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

RUN the program. See the four numbers on your screen. These numbers tell the Computer the horizontal and vertical coordinates of your two joysticks' "floating switches".

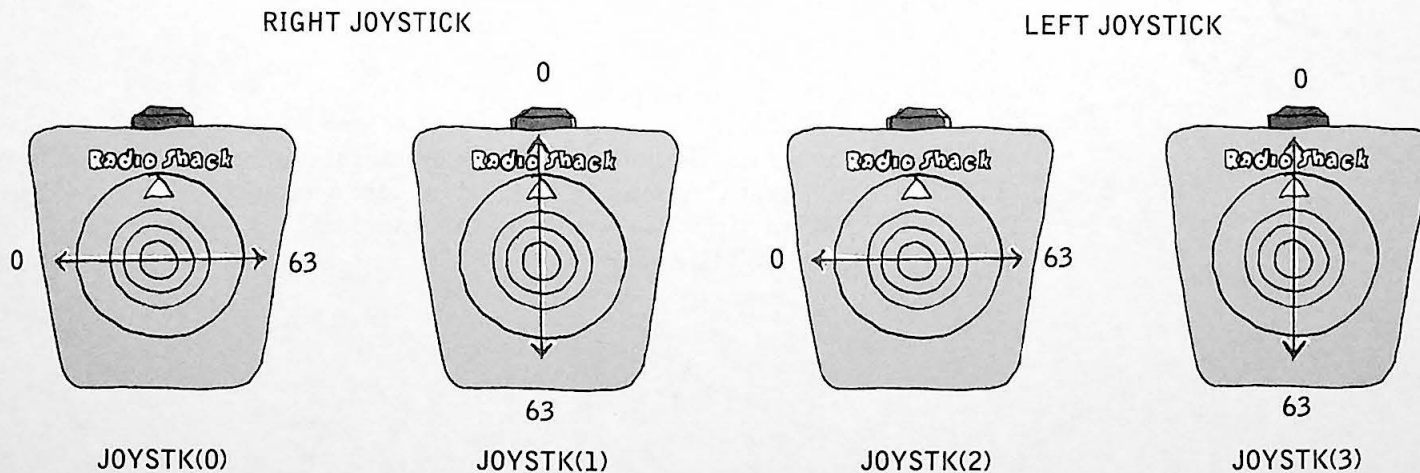
Grab the "floating switch" of your right joystick (the one connected to the jack marked RIGHT JOYSTICK on the back of the Computer). Keeping it in the center, move it from left to right. The first number on your screen will change, going through all the numbers from 0 to 63.

Move the "floating switch" of your left joystick from left to right. It will change the third number on your screen.

Now move the floating switches up and down, keeping them in the center. Moving the *right* joystick up and down makes the *second* number change from 0 to 63. Moving the *left* joystick up and down makes the *fourth* number change from 0 to 63.

The second or fourth number might change also, but NOT from 0 to 63.

This is how the Computer reads the position of your joysticks:



JOYSTK (0) and JOYSTK (1) tell the Computer to read the position of your *right* joystick:

- JOYSTK(0) makes it read the horizontal (left to right) coordinate.
- JOYSTK(1) makes it read the vertical (up and down) coordinate.

JOYSTK (2) and JOYSTK (3) tell the Computer to read the position of your *left* joystick:

- JOYSTK(2) makes it read the horizontal coordinate.
- JOYSTK(3) makes it read the vertical coordinate.

One more thing. 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.

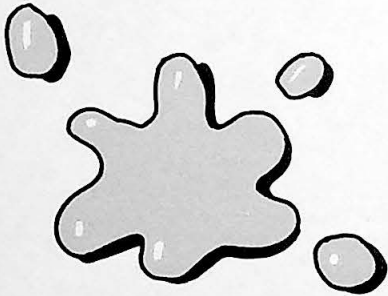
Now delete line 20 and change line 60:

```
60 GOTO 30
```

RUN the program. Move all the switches around. This time it doesn't work at all. The Computer will not read any of the coordinates unless you first have it read JOYSTK(0). Type these lines:

```
20 A = JOYSTK(0)
60 GOTO 20
```

and RUN the program. Even though the Computer is not printing the location of JOYSTK(0), it is still reading it. Everything else works like it's supposed to. Remember that anytime you're having the Computer read to coordinates of JOYSTK(1), JOYSTK(2), or JOYSTK(3), you must first have it read JOYSTK(0).



MAKE PAINT BRUSHES OUT OF JOYSTICKS:

Type this:

```
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
```

RUN it . . . 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 could be a number from 0 to 63.

Line 30 reads V — its vertical position. This also could be a number from 0 to 63. Since the highest vertical position on your screen is 31, we had to add line 40 to the program. Line 40 makes V always equal to 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.

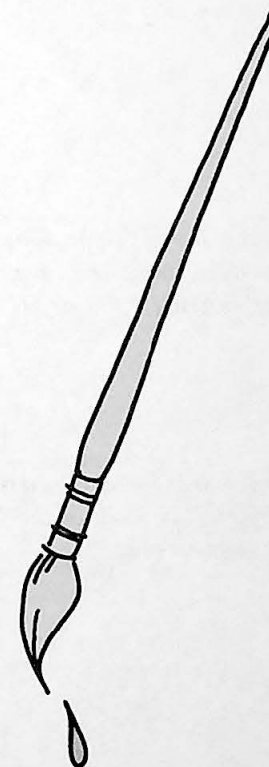
We haven't even used the left joystick. Perhaps we could use it for color. Add these lines:

```
50 C = JOYSTK(2)
60 IF C < 31 THEN C = 3
70 IF C > = 31 THEN C = 4
80 SET(H,V,C)
```

RUN the program. Move your left joystick to the right and the Computer makes C = 3. It SETs red dots. Move it to the left and the Computer makes C = 4 and SETs blue dots.

Want to make the buttons on your joysticks do something? Add these lines to the end of your program:

$> =$ means greater than or equal to



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

Now type:

```
RUN 100 ENTER
```

This tells the Computer to only RUN lines 100 through the end of the program. Your computer should be printing either 255 or 127 over and over again.

PEEK tells the Computer to look at a certain spot in its memory to see what number's there. We had it look at the number in location 65280. As long as you're not pressing either of the buttons, this spot contains the number 255 or 127.

If you press the buttons when you're not RUNNING the program you will get @ABCDEFGH or IJKLMNOP.

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.

Some of the joysticks will not read 6 "blocks" in each of the four corners of your screen.

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. Start your paintings. Press the right button when you want to clear the screen and start over again.

LEARNED IN CHAPTER 9

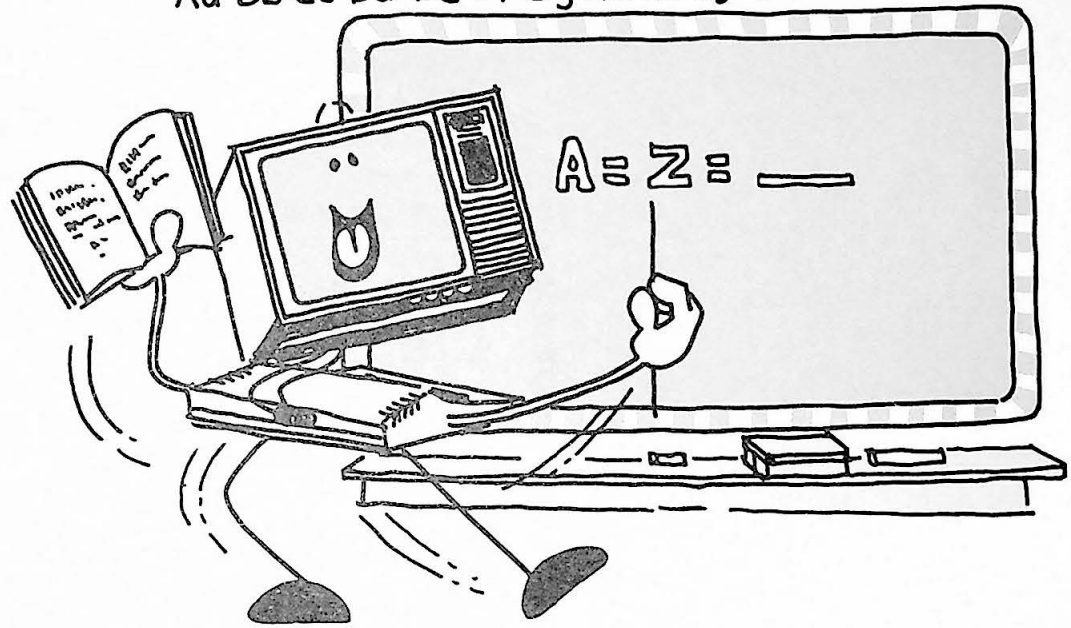
BASIC WORDS

SET
RESET
JOYSTK
PEEK

NOTES:

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

CHAPTER 10



ONE FANTASTIC TEACHER

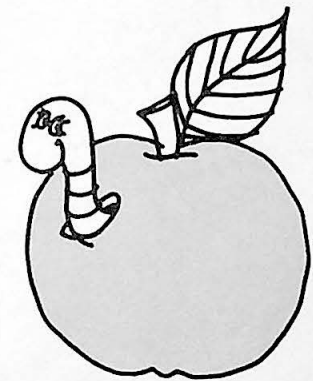


ONE FANTASTIC TEACHER

Your Computer has all the attributes of a natural born teacher. After all, it's patient, tireless, and detail conscious (. . . perhaps a bit nit-picky . . .). Depending on the programmer — we're talking about you, of course — it can be imaginative, consoling, and quite enthusiastic.

So lets get on with it! We can use RND to get the Computer to drill us on one math problem after the next. 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
```



This program will drill you on your multiplication tables, from 1 to 15, and check your answers.

How would you change this program to get the Computer to drill you on addition problems from 1 to 100:

DO-IT-YOURSELF PROGRAM

Here's 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
```

To make the program a little more interesting we can have the Computer keep a running total of all the correct answers. Type:

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

T keeps a count of all the questions the Computer asks you. When you first

RUN the program T equals zero. Then, everytime the Computer gets to line 15, it adds 1 to T.

C does just about the same thing. It keeps a count of the number of correct answers. Since it is on line 95, the Computer will not increase C unless you get a correct answer.

There are many ways to make this program more entertaining. Add some lines to the program which will get the Computer to 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 percent of correct answers.
5. End the program if you get 10 answers in a row correct.

Use your imagination on this one. We have a program in back which does all five of the above.



When you first turn on the Computer, all numeric variables equal 0. Also, when you type NEW (ENTER), all numeric variables equal 0.

There are many variations you could try with this program. For instance, the Computer could test you with business questions.

DO-IT-YOURSELF PROGRAM

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
```

So what happened? Nothing? Nothing that you can see, that is. To see what the Computer is doing, add this line and RUN it:

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

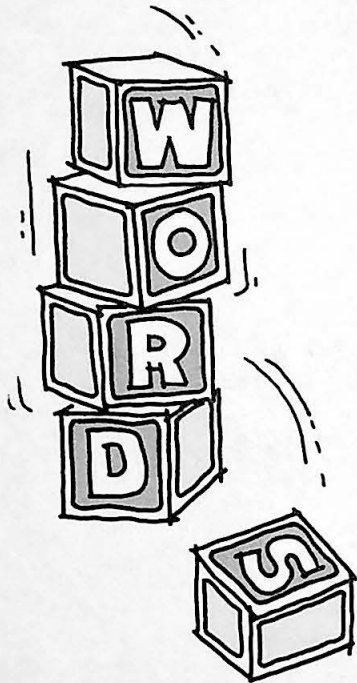
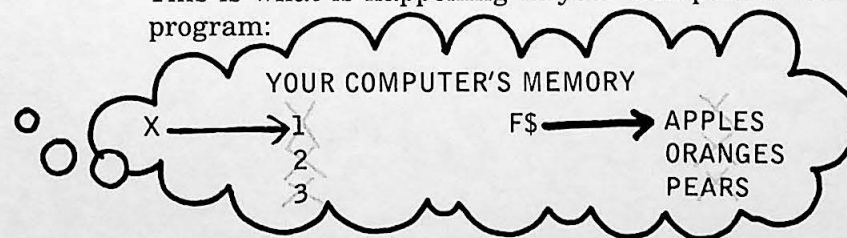
Line 30 tells the Computer to:

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 things:

1. Look for a DATA line
2. READ the first item — this time it is ORANGES
3. Give ORANGES the F\$ label
4. "Cross out" ORANGES

This is what is happening in your Computer's memory when you RUN the program:



What if you want the Computer to READ the same list over again? It's already crossed everything out . . . Type:

```
60 GOTO 10
```

and RUN the program. It prints ?OD ERROR IN 30. OD means Out of Data. The Computer has already crossed out its Data.

Type this line and RUN the program:

```
50 RESTORE
```

Now it's as if the Computer never crossed anything out. The Computer will READ the list over and over again.

The nice thing about DATA lines is that you can put them anywhere you want in the program. RUN each of these programs:

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

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

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

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

*Remember how to make the Computer pause while RUNning a program? Press **SHIFT** @ Press any key to get it to continue.*

They all work the same, don't they? This knowledge should be handy for something . . .

...NOW HAVE IT BUILD YOUR VOCABULARY

Here's some words and definitions you might want to be tested on:



"Cataclysmic!"

Words	Definitions
-------	-------------

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 to get the Computer to pick out a word at random from the list. Hmm . . . there are ten items in the list. Maybe this will work:

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

RUN it a couple of times to see if it works.

It doesn't quite work like we want it to. The Computer is just as likely to stop at a definition as a word. What we really want the Computer to do is to pick a random word from items 1, 3, 5, 7, or 9.

Fortunately, there is a word which will explain this to the Computer. Type:

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

RUN the program a few times. It should work now.

INT tells the Computer to only look at the whole portion of the number and

ignore the decimal part. For instance, the Computer sees INT(3.9) as 3.

Here's how line 65 works. Say the random number the Computer picks is 10. The Computer does this calculation:

$$\begin{aligned}\text{INT}(10/2) &= 10/2 \\ \text{INT}(5) &= 5 \\ 5 &= 5\end{aligned}$$

Since this is true, 5 *does* equal 5, the Computer completes the THEN portion of the line and makes N equal to 9 (10 - 1).

However, if the Computer picks 9, it does this:

$$\begin{aligned}\text{INT}(9/2) &= 9/2 \\ \text{INT}(4.5) &= 4.5 \\ 4 &= 4.5\end{aligned}$$

Since this is not true, 4 *does not* equal 4.5, the Computer doesn't subtract 1 from N. 9 remains 9.

Now that the Computer is able to READ a random word, it must also READ the word's definition. You can do this simply by adding these lines to the end of the program:

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

RUN it several times now. To get the Computer to print one random word and definition after the next, add this line to the beginning of the program:

```
5 CLEAR 100
```

to give the Computer plenty of string space. And add these lines to the end of the program:

```
130 RESTORE
140 GOTO 60
```

So that the Computer can pick out a new random word and its definition from a

clean slate of data items.

Here is the way the entire program looks now:

```
5      CLEAR
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
```

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

For variations on this program, you might try states and capitols, cities and countries, foreign words and meanings. Got more ideas?

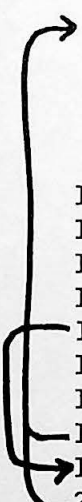
Want to complete this program? Try it before turning the page to see ours. Program it so that the Computer will:

1. PRINT the definition ONLY
2. Ask you for the word
3. Compare the word with the correct random word
4. Tell you if your answer is correct. If your answer is incorrect, have it PRINT the correct word.

DO-IT-YOURSELF PROGRAM

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 dress the program up with a good looking screen format, sound, etc.

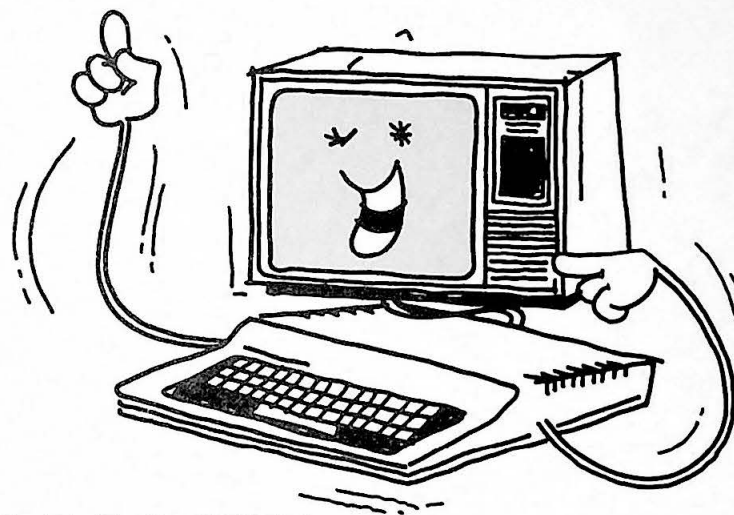
LEARNED IN CHAPTER 10

BASIC WORDS

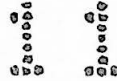
DATA
READ
RESTORE
INT
CLEAR

CHAPTER 11

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



HELP WITH MATH

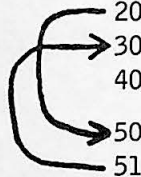


HELP WITH MATH

Solving complicated math formulas with super speed and precision is an area where your Computer shines. But before driving yourself crazy typing a bunch of math formulas, there are some shortcuts and hints you'll probably want to know about.

One easy way to handle complicated math formulas is by using 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
```



Line 20 tells the Computer to GO to the SUBroutine beginning at line 500. RETURN tells the Computer to return back to the BASIC word immediately following GOSUB.

Delete line 40 and see what happens when you RUN the program.

.....

Did you delete it?



If you did, this is what's on your screen:

```
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. Can you see why deleting END in line 40 would cause this error?

At first, the Computer went through the program just like it did before you deleted the END line. It was sent to the subroutine in line 500 by GOSUB and it returned to the BASIC word immediately following GOSUB.

Then, since you deleted END, it went to the next line in the program which was the subroutine. When it got to RETURN, it did not know where to return to, since this time it had not been sent to the subroutine by a GOSUB.

Here's a subroutine which raises a number to any power you want:

See something different about INPUT? We can have the Computer PRINT a message before waiting for us to INPUT something.

```
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 " P " POWER 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
```

Notice we introduced a couple of new things in the program.

Look at line 40. If you find it easier, you can combine two or more program lines into one, using a colon to separate the two lines. Line 40 contains the two lines:

```
PRINT  
and  
PRINT N " TO THE " P " POWER IS " E
```

Line 2000 has something else new — REM.

REM doesn't mean anything to the Computer. The Computer ignores any line beginning with REM. You can put REM lines anywhere you want in your program, so that you can remember what the program does. These REM lines will make *no difference to the way the program works*.

If you don't believe us, add these lines and RUN the program to see if they make any difference:

```
5 REM THIS IS A PECULIAR PROGRAM,  
17 REM THIS LINE SHOULD MESS UP THE PROGRAM  
45 REM THE NEXT LINE KEEPS THE SUBPROGRAM SEPARATED
```

Satisfied? . . . *Good*.

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

PRINT by itself tells the Computer to skip a line.

DO-IT-YOURSELF PROGRAM

The answer is in the back of the book.

GIVE THE COMPUTER A LITTLE HELP

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

Divide the sum of $13 + 3$ by 8

You might want the Computer to solve the problem like this:


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

But your Computer solves it differently. Type this command line:

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

What the Computer did was logical according to its rules:

The word "operation" means something you're getting the Computer to do. Here, we're talking about the "operations" of adding, subtracting, multiplying, and dividing.



RULES ON ARITHMETIC

The Computer solves arithmetic problems in this order

1. any multiplication and division operations are solved first
2. addition and subtraction operations are solved last
3. in case of a tie (that is, more than one multiplication/division or addition/subtraction operation) the operations are performed from left to right

In the problem above the Computer followed its rules. It performed the division operation first ($3/8 = .375$) and then the addition ($13 + .375 = 13.375$). To get the Computer to solve the problem differently, you can use parenthesis. Type this line:

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

Whenever the Computer sees an operation in parenthesis, it solves that before solving anything else.

What do you think the Computer will PRINT as the answers to each of these problems:

COMPUTER MATH EXERCISE

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

That is what you're actually asking the Computer to do:

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

When the Computer sees a problem with more than one set of parenthesis, it

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

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

↘ $5 - 1 = 4$

$$(10 - 4) / 2$$

↘ $10 - 4 = 6$

$$6 / 2$$

↘ $6 / 2 = 3$

RULES ON PARENTHESIS

1. When the Computer sees a problem containing parenthesis, it solves the operation inside the parenthesis **BEFORE** solving the rest of the operations.
2. If there are parenthesis inside parenthesis, the Computer solves the innermost parenthesis first and works its way out.

Insert parenthesis in the problem below so that the Computer will PRINT 28 as the answer:

COMPUTER MATH EXERCISE

PRINT 30 - 9 - 8 - 7 - 6

Answer:

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

IS SAVING WORTH IT?

With what you've learned in this chapter, you can let the Computer do all the math by putting complicated math formulas in your subroutines. 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
```



"A PENNY SAVED..."

Notice that we have one subprogram "calling" another subroutine. This is perfectly OK with the Computer as long as you have a GOSUB to send the Computer to each subroutine, and a RETURN in each subroutine to return the

Computer to the BASIC word following each GOSUB.

One more thing we think you'll like. Flip back to the Appendix, "Subroutines". We've put a lot of math formulas into subroutines. You'll probably want to add some of these to your programs.

LEARNED IN CHAPTER 11

BASIC WORDS

GOSUB
RETURN
REM

BASIC SYMBOLS

()
:

BASIC CONCEPTS

Order of operations

NOTES:

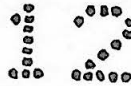
Two columns of horizontal lines for writing notes.



CHAPTER 12



A GIFT WITH WORDS



A GIFT WITH WORDS

One of your Computer's greatest skills is its gift with words. It can tirelessly twist, combine, or separate words to anything you want. Because of this gift, you can teach it to read, write, and even carry on a halfway decent conversation.

For starters, see what you think of this:

```
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\$) tells the Computer to compute the LENGTH of the string S\$ — your sentence. Your obedient Computer counts every single character in the sentence, including spaces and punctuation marks.

Here's another skill it has. Erase your program and type this to make it compose 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$
```

Not impressed? Well, later we'll talk about some practical ways to use this unusual skill.

Here the Computer combines strings. The plus sign tells it to do this. D\$ combines B\$ and C\$ to get "IS A ROSE", and you can see what strings are combined to form F\$.

A word of caution about combining strings — add this line to your program and RUN it:

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

When you RUN this program, the Computer prints ?OS ERROR IN 80. OS means Out of String Space. The Computer only reserves about 200 characters for working with strings. Add this line to the beginning of the program for reserving plenty of string space:

```
5 CLEAR 500
```

RUN the program again. This takes care of the first problem, but there's still another.

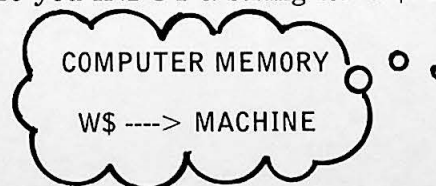
This time the Computer prints ?LS ERROR IN 80. LS means string too long. Line 80 asks the Computer to form a string — G\$ — with more than 255 characters. Your Computer simply can't manage to remember a string with that many characters.

Now that the Computer has combined strings, let's get it to take one apart. Type and RUN this program:

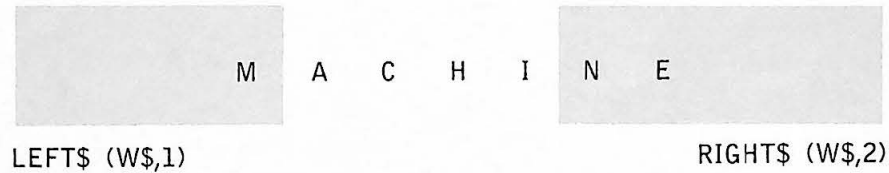
```
10 INPUT "TYPE A WORD"; W$
20 PRINT "THE FIRST LETTER IS : " LEFT$(W$,1)
30 PRINT "THE LAST 2 LETTERS ARE : " RIGHT$(W$,2)
40 GOTO 10
```

Here's what your Computer is doing:

In line 10 you INPUT a string for W\$. Let's say the string is MACHINE:



Your Computer then performs several calculations in lines 20 and 30 to get the first LEFT letter and the last 2 RIGHT letters of the string:



Try RUNning the program a few times until you get the hang of it.

Add this line to the program:

```
5 CLEAR 500
```

So that your Computer will set aside plenty of space for working with strings. Now INPUT a sentence rather than a word.

.....

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?

PROGRAMMING EXERCISE

```
20 .....  
30 .....
```

Answers:

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

.....

Erase your program and type this one:

Remember how to erase a program?

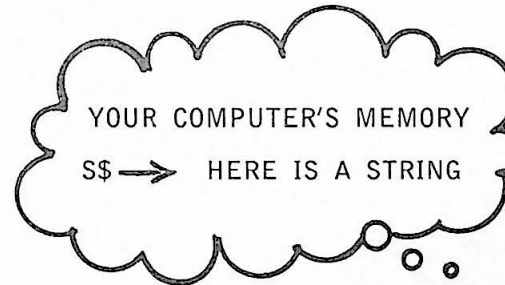
Type:

NEW **ENTER**

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

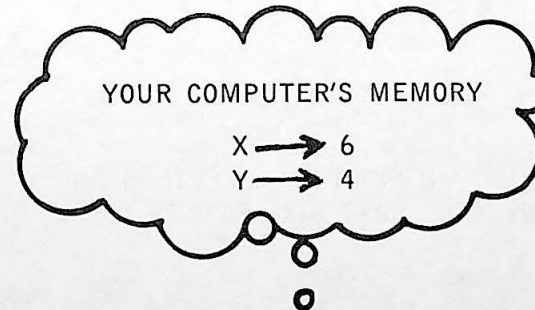
RUN this program a few times to see if you can figure out how MID\$ works.

Here's how the program works. Say you INPUT "HERE IS A STRING" for your sentence:

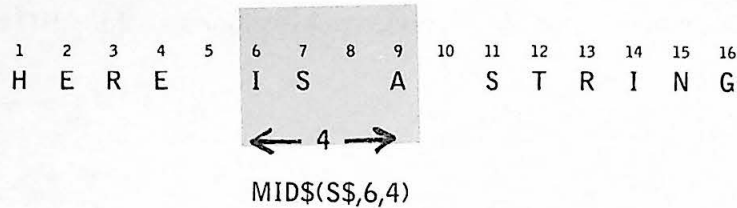


In line 30, the computer first calculates the LENGTH of S\$ — 16 characters. It then asks you to choose a number from 1 to 16. Let's say you pick the number 6.

The Computer then, in line 60, asks you to pick another number from 1 to 11 (16-6) plus 1. Say you pick the number 4.



In line 90, the Computer gives you a MID string of S\$ which begins at character number 6 and is 4 characters long:



Here's something else you can do with MID\$. Erase your program and type:

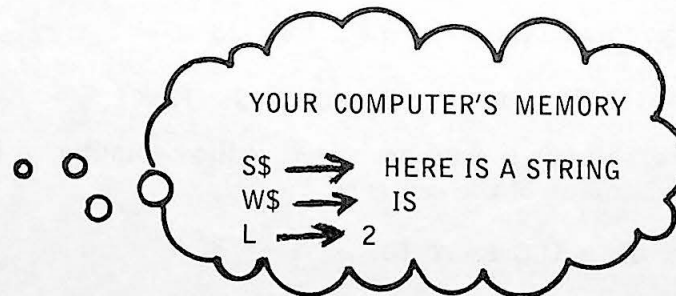
```

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

```

RUN this program a few times. Here's how it works.

Say you type in the above sentence, and the word you INPUT for W\$ is "IS". In line 30, the Computer then calculates the LENgth of W\$ — 2 characters.



The Computer then goes through the FOR/NEXT loop (lines 40-90) counting each character in S\$ beginning with character 1 and ending with character number LEN(S\$) — 16.

These types of programs can be used to sort through large files of information. For instance, by separating strings, you could look through a mailing list for TEXAS addresses.

Every time it counts a new character, the Computer looks at a new MID string. Each MID string begins at character X and is L or 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:

```

      4
  H E R E I S A S T R I N G
      ←2 →
MID$(S$,4,2)

```

It finally finds the MID string it is looking for when X equals 6.

YOUR COMPUTER CAN BE A TOUGH EDITOR

Are you beginning to get a picture of the Computer hacking away at your sentences with a red pen? You can program it to help refine your writing and save you hours of typing and rewriting.

Say you had a phrase you want to add to:

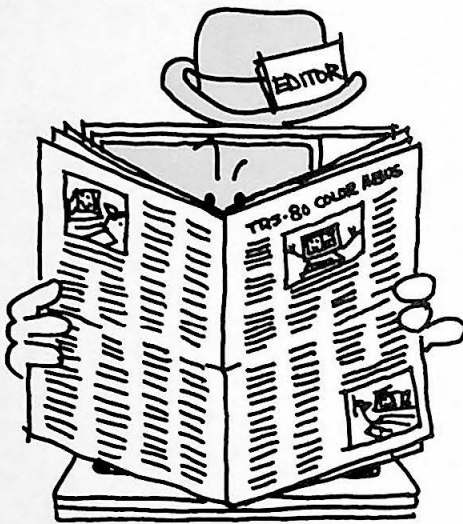
```
10 A$ = "CHANGE A SENTENCE."
```

Add to this one-line program so that the Computer will insert these words at the beginning of the sentence:

```
IT'S EASY TO
```

and PRINT the new sentence:

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



"#@%l&\$!"

DO-IT-YOURSELF PROGRAM

This is our program:

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

Now see if you can add to the program to get the Computer to:

1. Find the beginning location of the MID string:

A SENTENCE

2. Delete the words A SENTENCE, forming a new string:

IT'S EASY TO CHANGE

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

ANYTHING YOU WANT

4. And PRINT the newly formed string:

IT'S EASY TO CHANGE ANYTHING YOU WANT

DO-IT-YOURSELF PROGRAM

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 type of program is the basis of a "word processing" program — a popular program for cutting down on office typing expenses.

IF YOU LIKE A CHALLENGE, TRY THIS. . .

Write a program in which:

1. The Computer asks you to INPUT:
 - a. a sentence
 - b. a phrase within the sentence to delete
 - c. a new phrase to replace the deleted phrase
2. The Computer then PRINTs a new sentence with your change intact.

This may take a while, but you have everything you need to write it. Our answer is in the back of the book:

DO-IT-YOURSELF CHALLENGER PROGRAM

LEARNED IN CHAPTER 12

BASIC WORDS

LEN
LEFT\$
RIGHT\$
MID\$

BASIC String OPERATOR

+

NOTES:

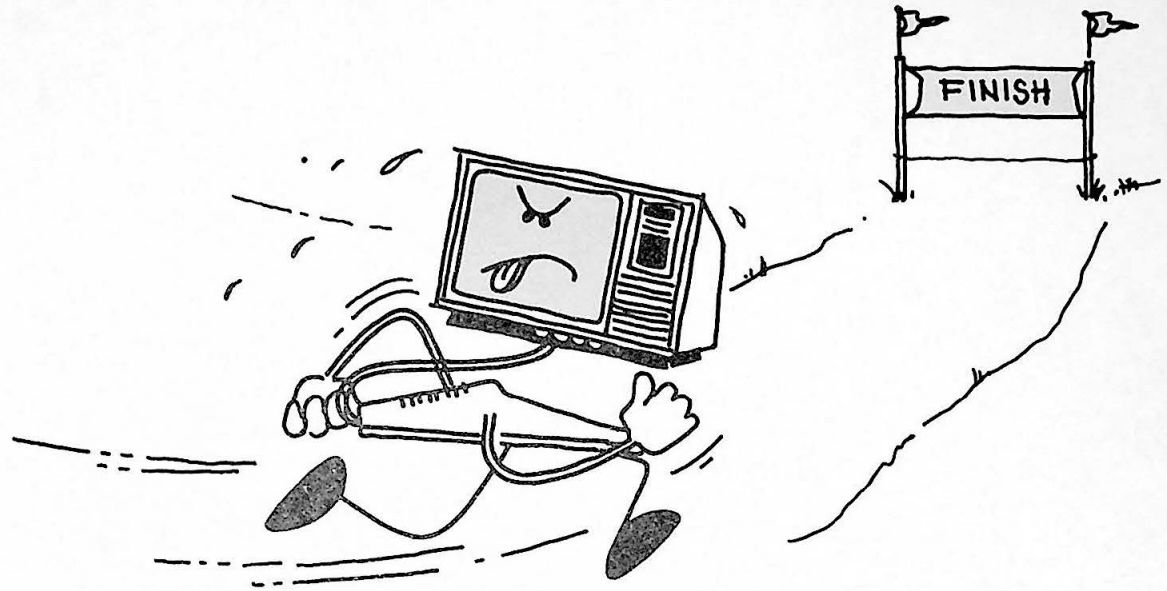
Handwritten notes area with horizontal lines.

NOTES:

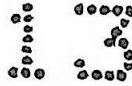
Two columns of horizontal lines for writing notes.



CHAPTER 13



BEAT THE COMPUTER



BEAT THE COMPUTER

You'll find the Computer much more adept by getting it to constantly watch and react to everything you do. By "watching you", we mean watching the keyboard to see if you are pressing something. The word INKEY\$ makes this possible.

Type this:

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

Remember what <> means? (It means "not equal to")

"" means an empty string — nothing

Press a key while RUNning this program.

INKEY\$ tells the Computer to look at the keyboard to see if you have pressed anything. The Computer does this with super-speed. You will have pressed absolutely nothing for at least the first 20 times the Computer checks.

The Computer labels this key, or this non-key (""), A\$. Then the Computer makes its decision:

If A\$ equals "" — *nothing* — the Computer prints "YOU PRESSED NOTHING" and goes back up to line 10 to check the keyboard again.

However, if A\$ equals *something*, the Computer goes to line 50 and prints the key.

For a constant look-out, type this and RUN the program:

```
60 GOTO 10
```

No matter how quick you are, the Computer is much faster! Erase line 30 to see what keys you're pressing.

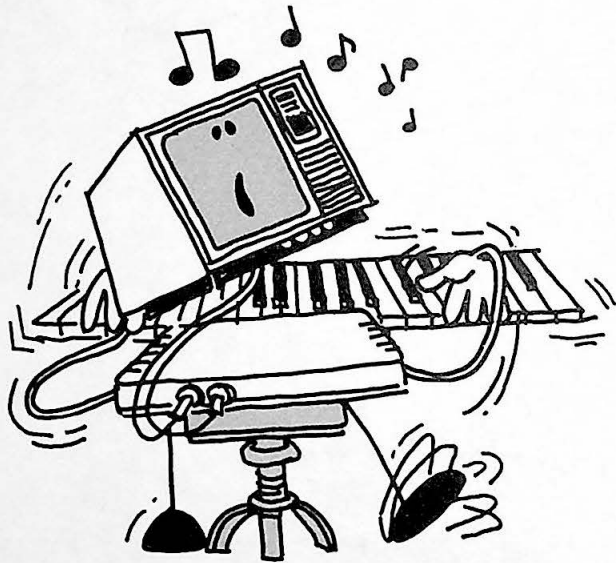
AN ELECTRONIC PIANO

Try using INKEY\$ to make a piano out of your keyboard. Look at that table in the Appendix, "Musical Tone Numbers". 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

We can tell the Computer that if you press a certain key it should SOUND one of these tones. Erase your program and type:

```
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
```



RUN the program . . . 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.

*How would this change the program?
120 SOUND T, 1*

Why wouldn't the program work right if you use INPUT rather than INKEY\$?

.....

If you use INPUT the Computer will wait until you press **ENTER** before acknowledging 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 would be done?

.....

This is what we came up with:

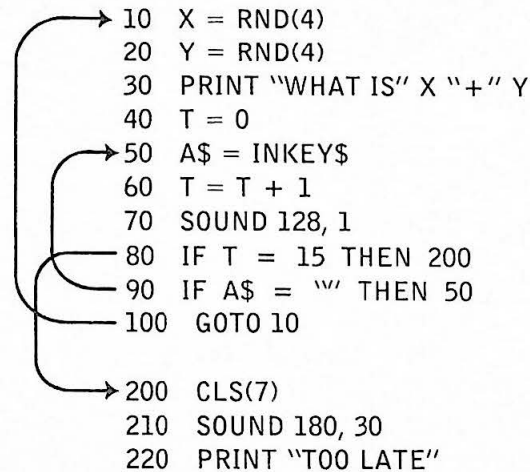
```
10 A$ = INKEY$
20 FOR X = 1 TO 8
30 READ B$, T
40 IF A$ = B$ THEN SOUND T, 5
50 NEXT X
60 RESTORE
70 GOTO 10
80 DATA A, 89, S, 108
90 DATA D, 125, F, 133
100 DATA G, 147, H, 159
110 DATA J, 170, K, 176
```

Using these DATA and READ lines will make it easier for you to add more tones to your Computer's repertoire.

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 what the program tells the Computer to do:

Lines 10, 20, and 30 gets the Computer to pick two random numbers and ask you what their sum is.

Line 40 sets T to 0. We will use T as a timer.

Line 50 gives you your first chance to answer the question — in one minute split second.

Line 60 adds 1 to 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. Everytime the Computer executes line 60 it will add 1 to T.

Line 70's just there to make you nervous.

Line 80 tells the Computer you have 15 chances to answer. Once T equals 15, time's up. The Computer will insult you with lines 200, 210, and 220.

Line 90 says if you haven't answered yet to go back and give you another chance.

The Computer only gets to line 100 if you do answer. It will go back for another problem.

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

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

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

```
?TM ERROR IN 100
```

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

Fortunately, your Computer has a way of doing this. Change line 100 by typing:

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

VAL(A\$) turns A\$ into its numeric VALue. If A\$ equals the string "5"; VAL(A\$) equals the number 5. (However, if VAL(A\$) equals the string "C"; VAL(A\$) equals 0 since "C" has no numeric value.)



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



For those that want to make the program a bit 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
```

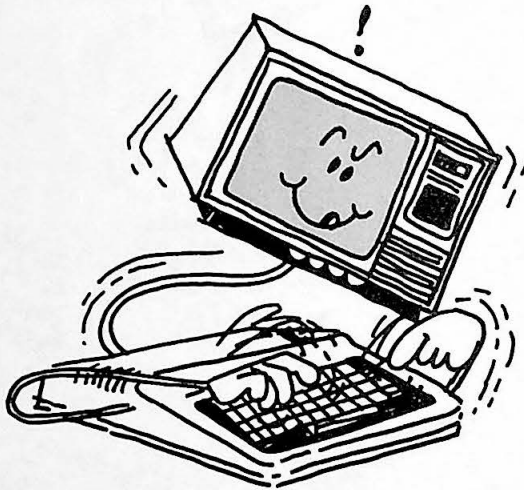
And add these lines:

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

A COMPUTER TYPING TEST

Here's a program that will get the Computer to time how fast you type:

```
10 CLS
20 INPUT "PRESS <ENTER> WHEN READY TO TYPE THIS PHRASE"; E$
30 PRINT "NOW IS THE TIME FOR ALL GOOD MEN"
40 T = 1
50 A$ = INKEY$
60 IF A$ = "" THEN 100
70 PRINT A$;
80 B$ = B$ + A$
90 IF LEN(B$) = 32 THEN 120
100 T = T + 1
110 GOTO 50
120 S = T/74
130 M = S/60
140 R = 8/M
150 PRINT
160 PRINT "YOU TYPED AT —'R'—WDS/MIN"
```



Here's how this program works:

In line 40, we set the timer — T — to 1.

Line 50 gives you your first opportunity to type a key — A\$. If you're not quick enough, line 60 sends the Computer directly 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\$), it will be added to B\$. For example, if the first key you type is "N", then:

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

```
"Now  
is  
the  
time  
for  
good  
men"
```


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

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

And if the third key you type is "W", then:

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

When the LENGTH of B\$ equals 32 characters (the length of "NOW IS THE TIME FOR ALL GOOD MEN"), the Computer assumes you've finished typing the phrase and goes to line 120 to compute your words per minute.



We could have made this calculation in one line by using parenthesis:

$$120 R = 8((T/74)/60)$$

We calculate the words per minute in lines 120, 130, and 140 by dividing T by 74 (to get the seconds), S by 60 (to get the minutes), and then dividing the 8 words by M to get the rate of words per minute.

Change this program to get the Computer to check to see if you made a typographical error.

DO-IT-YOURSELF PROGRAM

Our answer is in the back of this book.

How about trying a variation of this program — a speed reading test.

LEARNED IN CHAPTER 13

BASIC WORDS

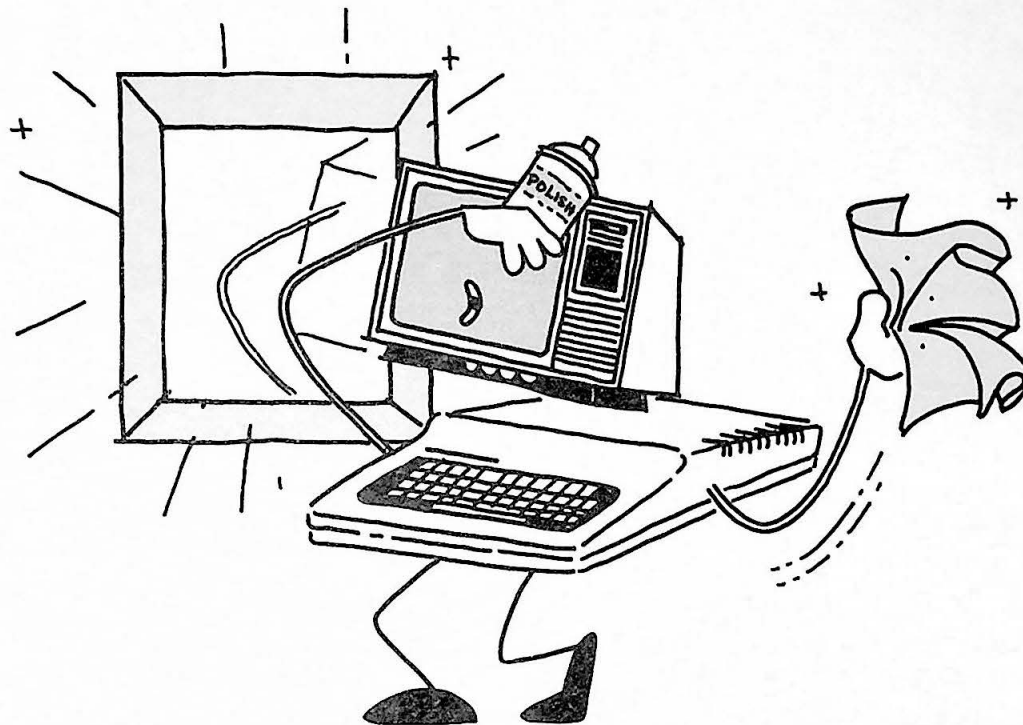
INKEY\$
VAL

NOTES:

<hr/>	<hr/>
<hr/>	<hr/>
<hr/>	<hr/>
<hr/>	<hr/>
<hr/>	<hr/>
<hr/>	<hr/>
<hr/>	<hr/>
<hr/>	<hr/>



CHAPTER 14



POLISH IT OFF



POLISH IT OFF

Before we let you finish this section, there are a few more BASIC words we want to tell you about. You don't have to know them. They'll just make programming a little easier for you.

The first word is STOP. Easy enough? 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 prints:

```
BREAK IN 30
OK
```

The Computer STOPped executing the program when it got to line 30. At this point, you can type a command line to see what your program has done so far. for example, type:

```
PRINT A (ENTER)
```

The Computer prints 2 — the value of A when it STOPped running the program.
Now type:

CONT (ENTER)

CONT

The Computer CONTinues to run the program where it STOPped. In other words it CONTinues running the program at line 40. Then it prints:

BREAK IN 50

the second place you have a STOP. Now, you may type:

PRINT A (ENTER)

again. It prints 4, the value of A at line 50. Type CONT again and the Computer breaks back up at line 30. If you have it PRINT A it will print 5, the value of A at line 30 the second time through the program.

STOP and CONT are for times when your program isn't working as you expected it to. By putting STOP lines in your program you can analyze what's going wrong. Once you fix the program, you can take the STOP lines out.

MEM

FOR AMBITIOUS PROGRAMS . . .

Type NEW to clear memory and then type:

PRINT MEM (ENTER)

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

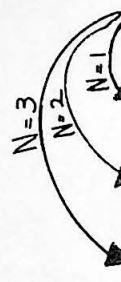
To save memory, you can omit spaces in your program before and after punctuation marks, operators, and BASIC words.

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.

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.

Line 20 could actually be replaced by these three lines:

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

It's simply fewer lines to type when you use ON . . . GOSUB.

ON . . . GOSUB tells the Computer to look at the number following ON – in this case number N. If it's a 1, the Computer goes to the subroutine beginning at the *first* line number following GOSUB. If N is 2, the Computer goes to the subroutine beginning at the *second* line number; if N is 3, the Computer counts down to the *third* line number and goes to that subroutine.

What if N is 4? Since there is no fourth line number, the Computer simply goes to the next line in the program.

ON
GOSUB

Here is a program that uses ON ... GOSUB:

ON
GOTO

```
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
```



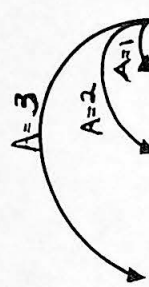
When A does not equal $X + Y$, the condition set up in line 1020 is not true.

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 $X + Y$ 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 simply sends the Computer to another line number, rather than 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
```



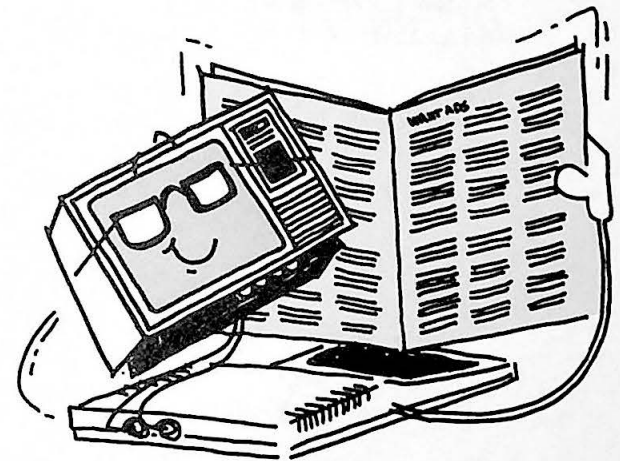
DOES THE JOB SAY "AND" OR "OR"?

Anyone who speaks English knows the difference between AND and OR — even your Computer. For example, let's say that Radio Shack has a bunch of job openings in programming. To get the job, you must have:

a degree in programming
AND
experience in programming

Here it is in a program. 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'VE GOT THE
JOB" ELSE PRINT "SORRY, WE CAN'T HIRE YOU"
50 GOTO 10
```



RUN the program. With your experience on the Color Computer, you might answer the questions this way:

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

Now let's say Radio Shack decided to be more lenient. Here are the new job qualifications:

The word "AND" is written in a large, bold, hand-drawn font. Each letter has small vertical lines extending from its top and bottom, giving it a slightly jagged or mechanical appearance.

```
a degree in programming  
OR  
experience in programming
```

All they did is change one little word. They changed AND to OR. To make this change in your program type:

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

The word "OR" is written in a large, bold, hand-drawn font. Each letter has small vertical lines extending from its top and bottom, giving it a slightly jagged or mechanical appearance.

To see the difference that one word makes, RUN the program:

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

Now that you see your Computer understands the meaning of AND and OR, you can use them in your programs. We'll be using these words in the next sections.

MORE HELP WITH MATH

There's a couple more words you might want to use to help with math programs:

SGN

SGN tells you whether a number is positive, negative, or 0. Type:

```
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. Try INPUTting some numbers like these:

```
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 numbers like 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"
```

One more thing before we let you finish this section...



Type and RUN this program:

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.)

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

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

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

Sometimes a number will get so large or so small that the Computer will cope with it by printing it in "exponential notation". The number "one billion" (1,000,000,000), for example, can be written " $1E+09$ ". This means "the number 1 followed by nine zeros."

If an answer comes out " $5E-06$ ", that means we must shift the decimal point, which comes after the 5, six places to the *left* inserting zeroes as necessary. Technically, it means $5*10^{-6}$, or 5 millionths. (.000,005). It's really pretty simple once you get the hang of it, and a lot easier to keep track of numbers without losing the decimal point.

We've run you through this Chapter pretty fast, but we think you'll appreciate knowing these odds and ends when you go through the next sections or practice your own programs.

LEARNED IN CHAPTER 14

BASIC WORDS

STOP
CONT
MEM

SGN
ABS
STR\$

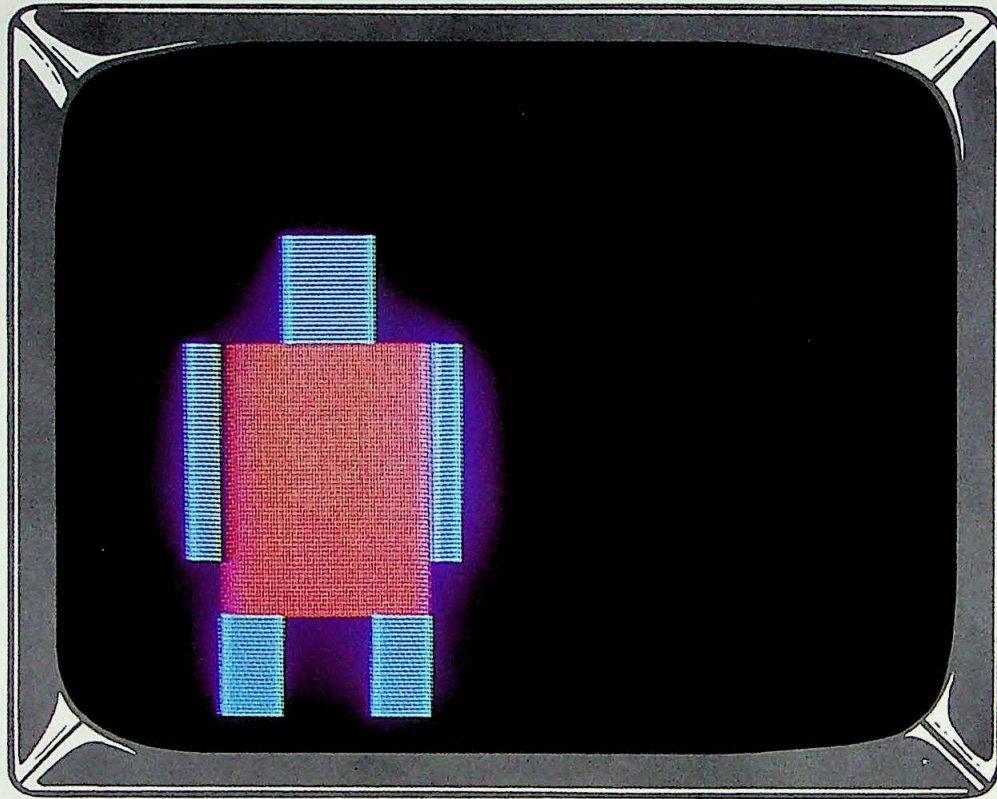
BASIC SYMBOLS

AND
OR

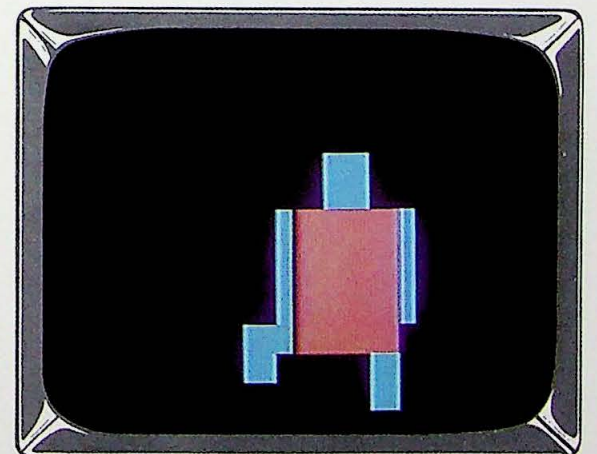
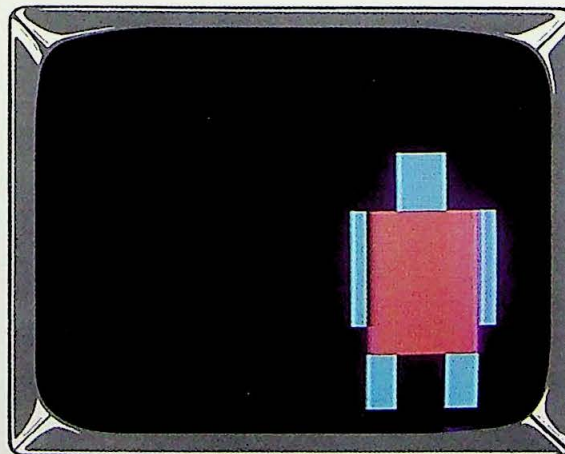
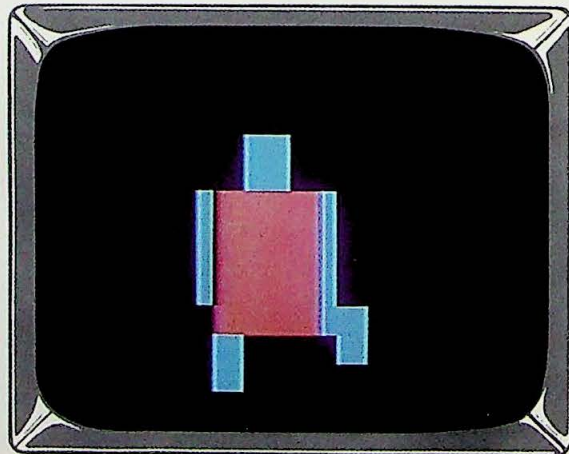
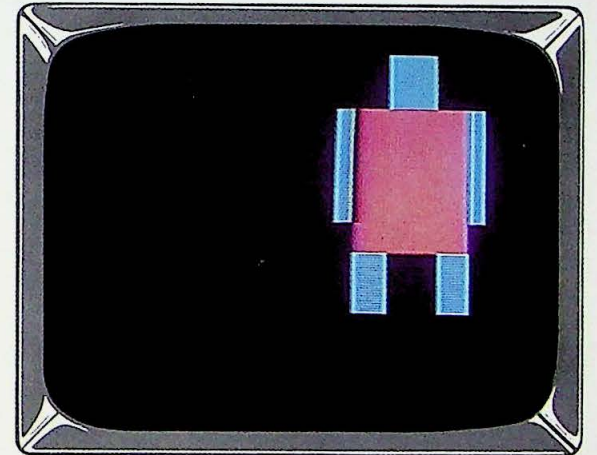
BASIC CONCEPT

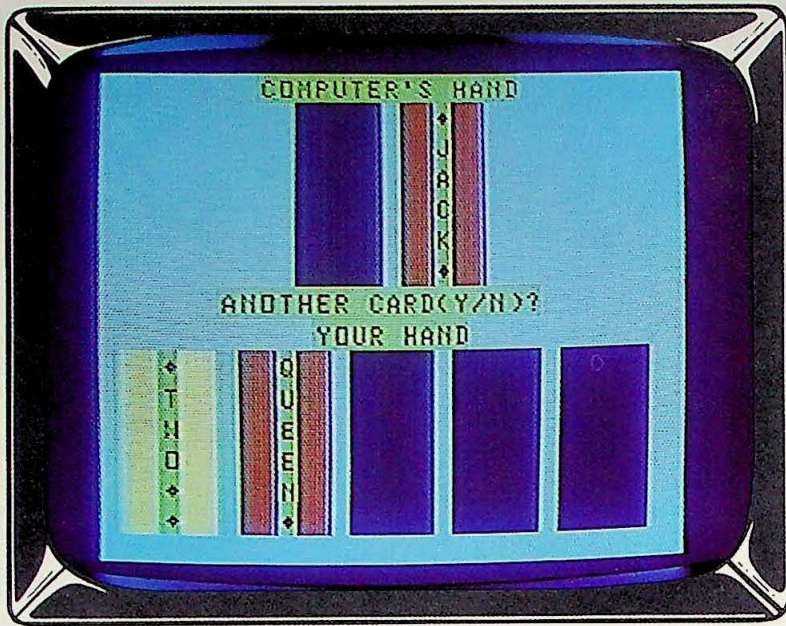
Exponential notation

PICTURE THIS

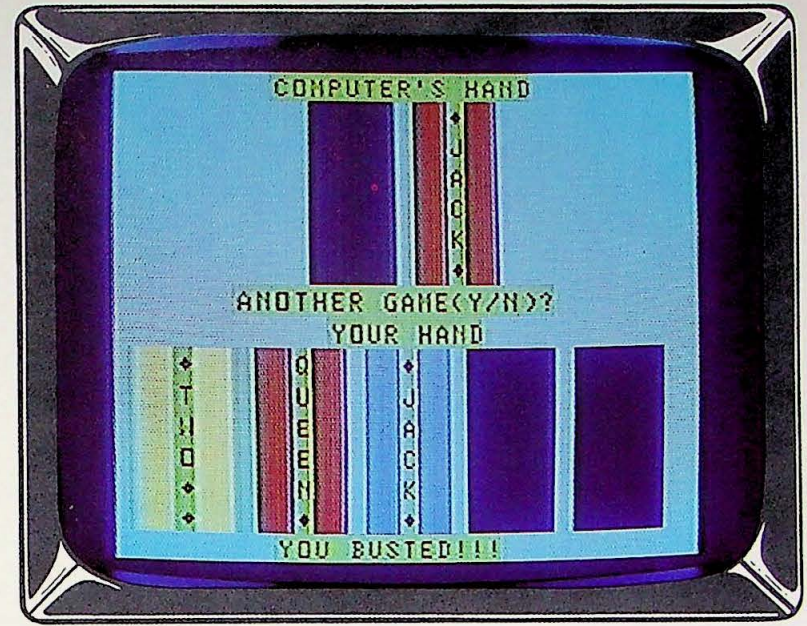


By programming different positions for this man, along with a song, you can easily make him dance. Chapter 19, "Let's Dance," shows how to do it.





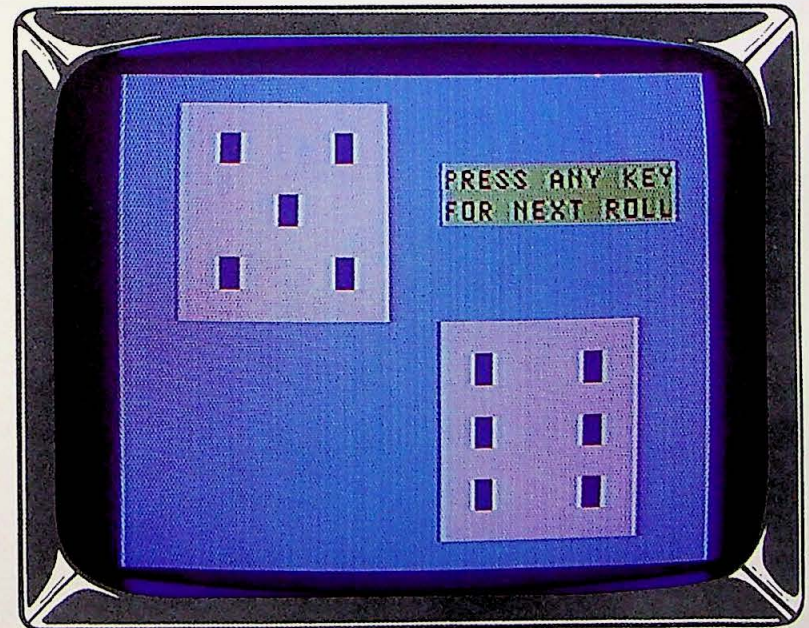
A



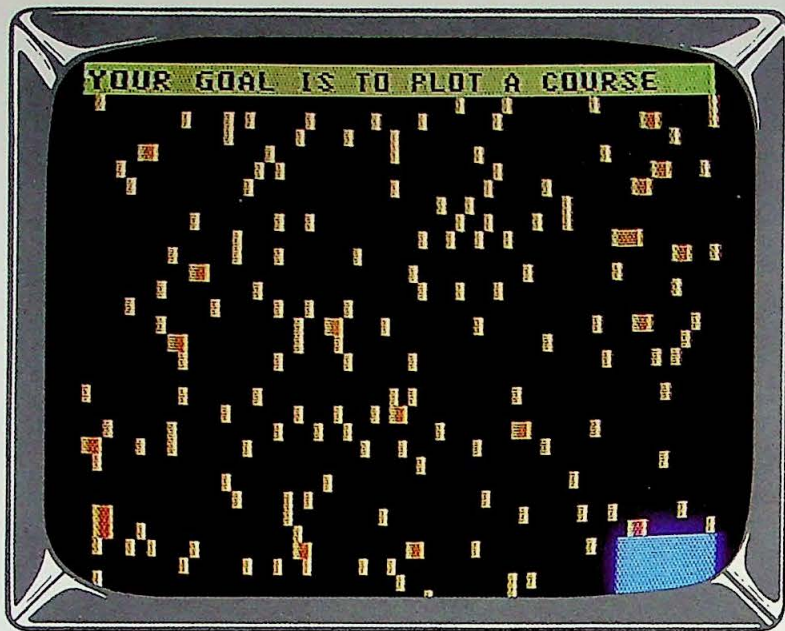
B

(A) and (B) This "Blackjack" game uses different colored cards to represent suits. See Appendix H, "Sample Programs," for a program listing.

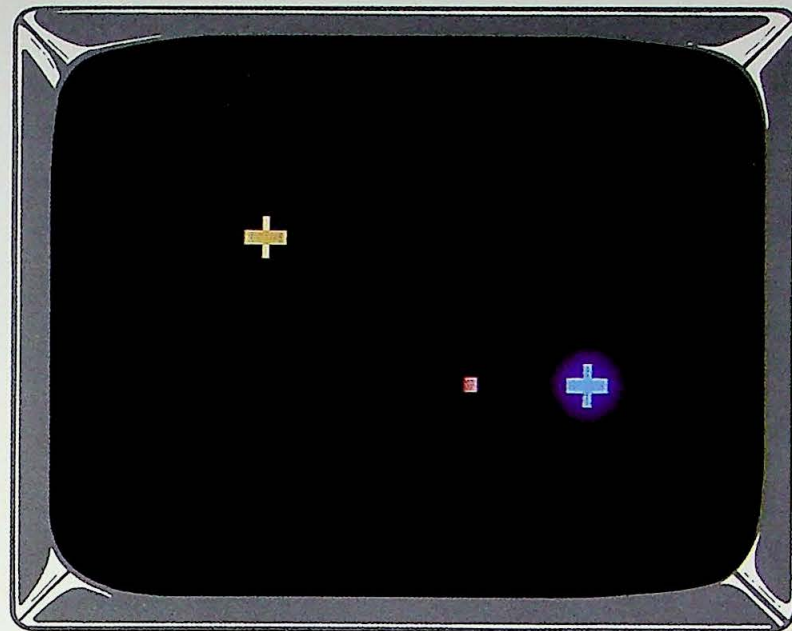
(C) Press any key and the "Electric Dice" will roll. This program is listed in Appendix H also.



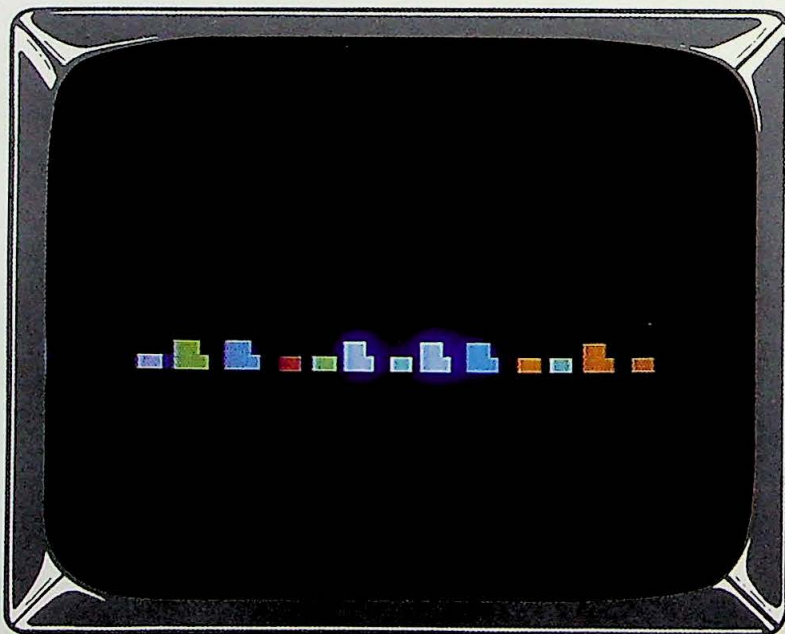
C



D



E



F

(D) Use your joystick to guide your spaceship through this maze of asteroids. Chapter 17, "Games of Motion," shows how to write this program.

(E) The joysticks propel these two spaceships. The blue one has a gun. You can fire it by pressing the button on the joystick propelling it. See "Spaceguns" in Appendix H, "Sample Programs".

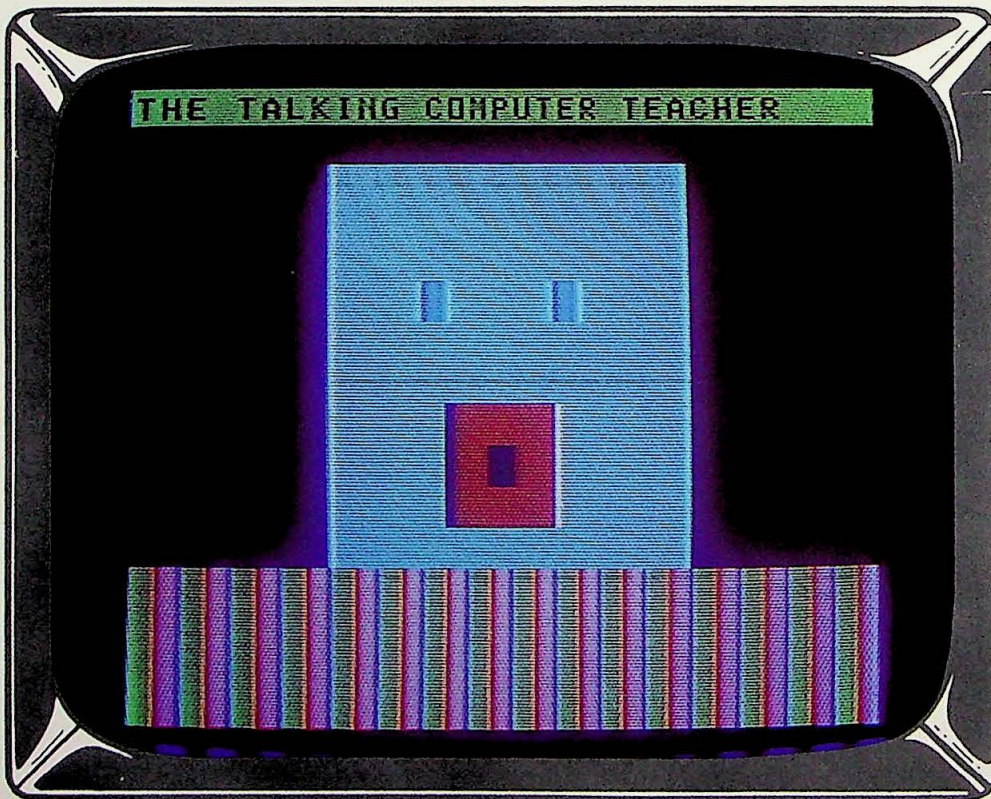
(F) You can create a random traffic jam that moves perpetually with graphic strings. Chapter 18 shows you how.

(G) Chapter 16 shows how to create this "Talking Teacher" by making the T.V. sound your own taped voice.

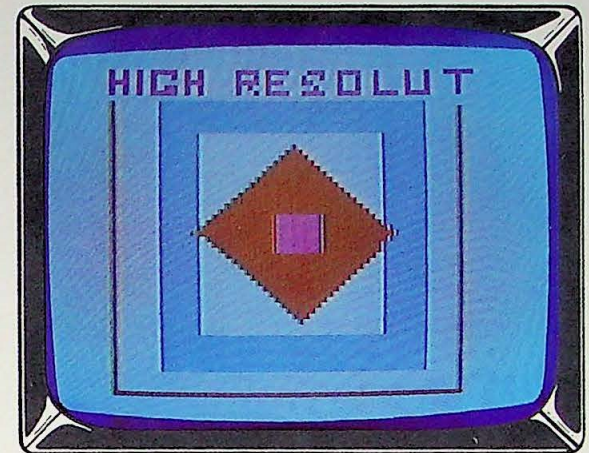
(H) This "Etch-a-Sketch" program uses high resolution graphics. Pressing the "/" key changes the colors (I). See Program Listing #1 in Part A of Section IV for the program listing.

(J) This "Kaleidoscope" program is listed in Appendix H, Sample Programs.

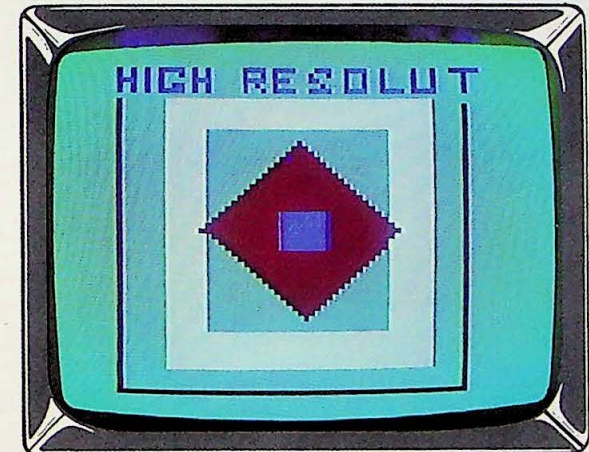
G



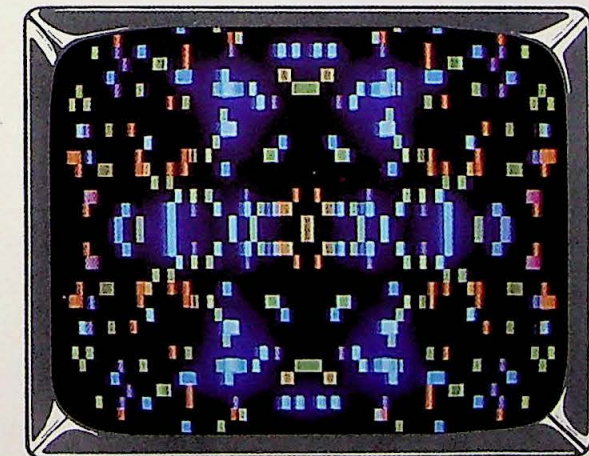
H



I

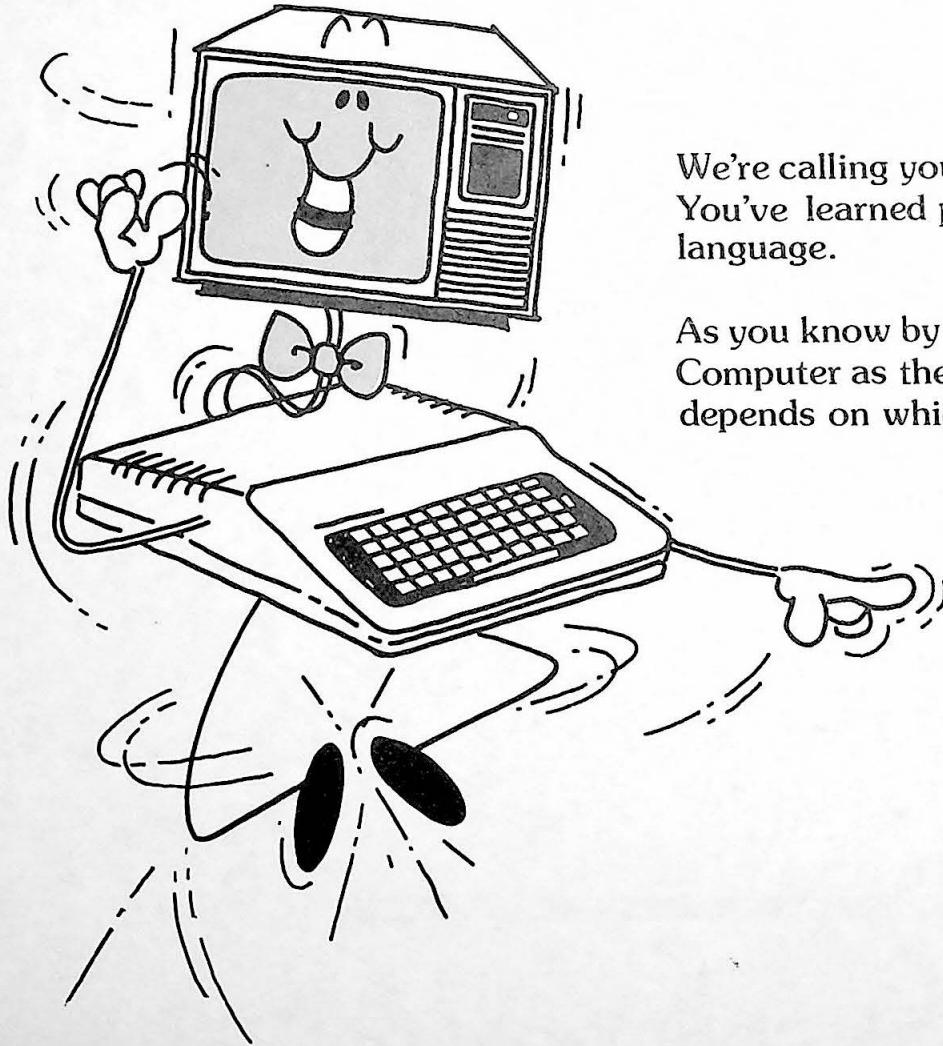


J



CONGRATULATIONS

PROGRAMMER!



We're calling you a programmer, because that's what you are. You've learned practically the entire BASIC programming language.

As you know by now, there are as many ways to use the Computer as there are types of people. What you do now depends on which type you feel like you are right now.



A CREATIVE SORT

You probably want to do a lot more with color graphics. That's why you bought the Color Computer in the first place! The next section, "Graphics with Pizzazz," is devoted to you.

A PRACTICAL BUSINESS TYPE

All of this has been fun, but you're ready to put the Computer to work! Skip section two and go straight to section three, "Getting Down to Business." We come down to earth there and get very practical.

A PERSON ON THE GO

Enough of this reading and learning – you've got a lot of ideas and you're ready to start programming! We think that's great, and we won't get our feelings hurt if you don't read any more of this book. Honest!

A TECHNICAL WHIZ

Armed with a strong curiosity and some staying power, you can step into section four, "Don't Byte Off More Than You Can Chew." There we show you how to change some of the inner workings of your Computer to do high resolution graphics and to call out machine-language programs.

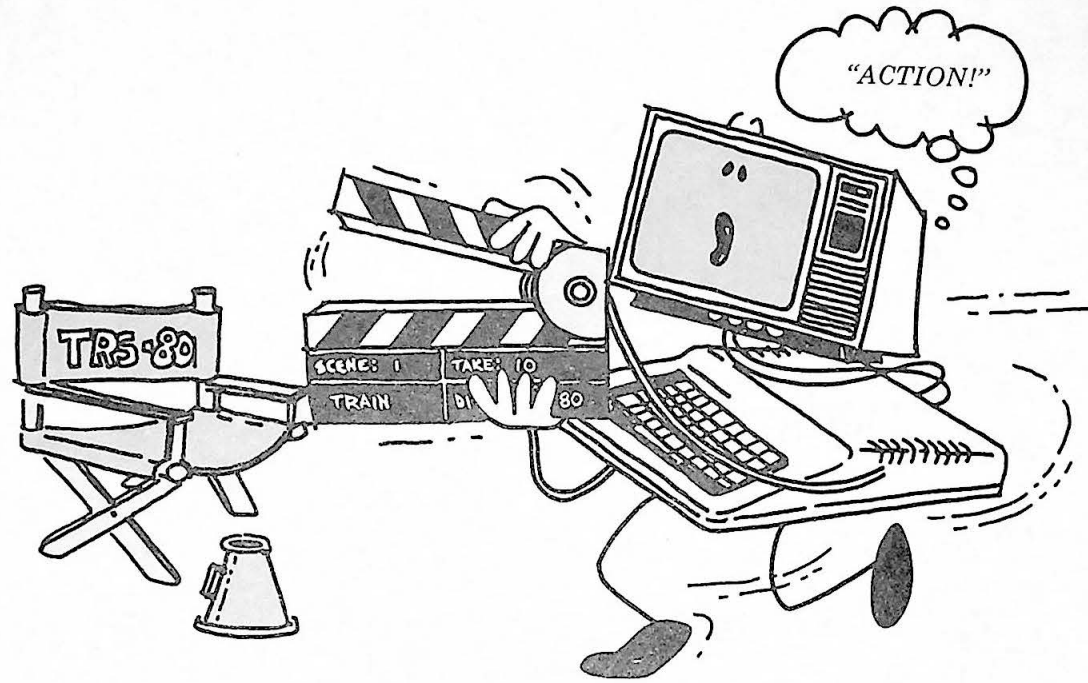
SECTION II

**GRAPHICS
WITH PIZZAZZ**

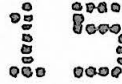
Those of you who want to write colorful and exciting programs will definitely want to read this Section. Here, we'll put pictures on your screen that move, dance, and even talk.

To keep things simple, our program examples are short. Once you understand what we're doing, you can easily create your own much more impressive programs.

CHAPTER 15



MOVING PICTURES



MOVING PICTURES

Ready to put some life into your programs? Animation is the key to making your graphics programs fun.

There are two ways to program graphics. The first is by using SET and RESET to position a “dot” horizontally and vertically. We’ll use this method first. In Chapter 18 we’ll introduce a new method to you.

Type:

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

and the Computer prints:

65

65 is the “ASCII” code for the character A. Type:

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

and the Computer prints A, to tell you that the CHaRacter that code number 65 represents is A.

Look at the list of “ASCII Character Codes” in your Appendix. Each character on



“ASCII” stands for the American Standard Code for Information Interchange. By using these standard codes, your Computer is capable of communicating over the telephone with other computers.


your keyboard has a code. Try testing some of the other characters . . .

So how does this help with graphics? With most of the characters on your keyboard, you can program what your Computer should do when you type them. For example, you could type these lines in your program:


```
10 INPUT A
20 IF A = "W" THEN H = H - 1
```


to tell the Computer what to do when you type the character W.

However if you try to substitute the  key for W, in line 20, the Computer will not let you do it. This is because the Computer has already decided what to do when you type the  key.

To get around this, we can use CHR\$(8) to represent the  character. Type NEW to erase your Computer's memory and type:

```
10 CLS(0)
20 H = 63
25 SET(H,14,3)
30 A$ = INKEY$
40 IF A$ = CHR$(8) THEN 60
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  character. Each time you press it, it backspaces the blue dot.

Line 30 tells the Computer to label whatever key you are pressing or not pressing as A\$. If A\$ equals the character represented by code 8 – the  character – then the Computer will go to line 60.

We'll talk about some more uses for CHR\$ later on in this section.

Need to review INKEY\$? See Chapter 13.

In lines 60 and 70 it “backspaces” H, the horizontal coordinate and SETs a blue dot. Then, in line 75, it RESETs or blacks out the previously SET blue dot.

Write some more lines to the program so that when you press the ☹ character, the Computer will move the dot forward.

DO-IT-YOURSELF PROGRAM

Our answer is in the back of this book.

A TRAIN THAT MOVES

Now that you understand CHR\$, you can use it in a moving picture.

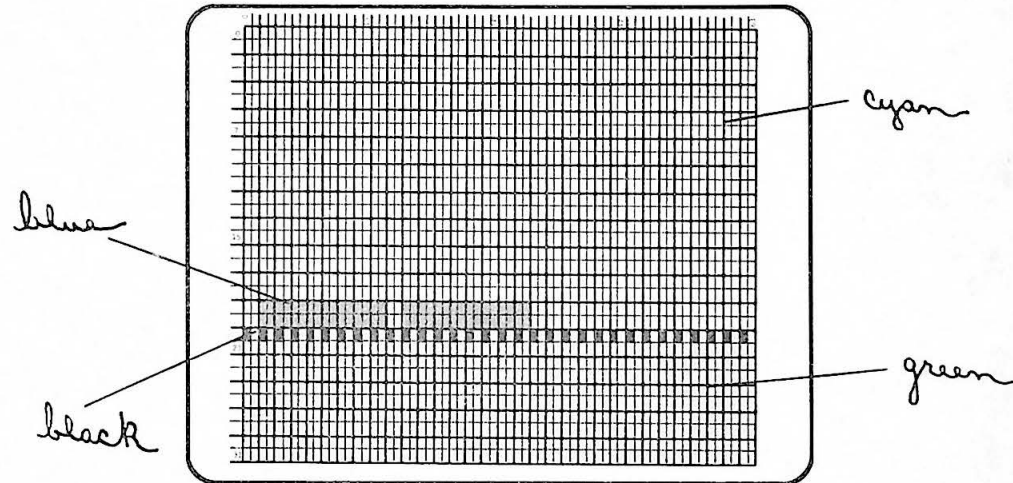
Programming will be much easier if you use this grid to plan your graphics in advance. Use pencil or make some photo copies of it.

Want a review of this? We talk about it in Chapter 9.

Press **BREAK** to get out of program.

Before writing the program, we will draw a grid of what it will look like, using the grid in Appendix D, "Graphics Screen Locations".

Our scene will look something like this:



Notice how the grid is divided into blocks — the groupings outlined with the dark lines. Each block contains 4 dots. All four dots in a block must be:

- all one color, or
- one color and black

Since the track markings are black, we can let them share the same block as the green grass.

Let's create the scene first. After typing the lines to create each part of the scene — the sky, grass, tracks, and train — you might want to RUN the program to see what it looks like.

To make the sky cyan, erase memory and type:

```
10 CLS(6)
```

For green grass type:

```
20 FOR H = 0 TO 63
30 FOR V = 22 TO 31
40 SET(H,V,1)
50 NEXT V,H
```

This SETs every dot green (color # 1) from Horizontal locations 0 to 63 and Vertical locations 22 to 31. Notice that line 50 actually contains two instructions:

```
NEXT V
NEXT H
```

To make the track markings, type:

```
60 FOR H = 0 TO 63 STEP 2
70 RESET(H,22)
80 NEXT H
```

This blacks out (RESETs) every other dot of the green grass in vertical location 22.

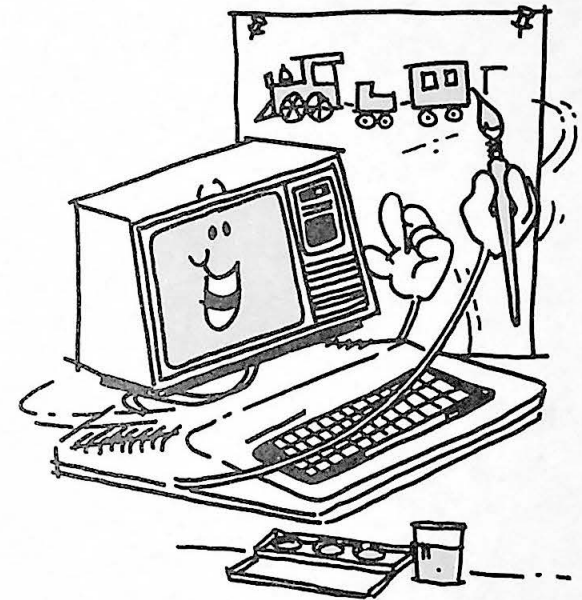
To make the train, type:

```
90 FOR V = 20 TO 21
100 FOR H = 0 TO 15
110 SET(2 + H,V,3): SET(20 + H,V,3)
120 NEXT H,V
```

This sets a train with two cars, each 16 dots long (0 to 15). One begins at horizontal location 2 and the other begins at location 20.

RUN the program to make sure your scene looks like the one we graphed above.

Looks the same? Now we'll make the train move forward. Type:




Since we haven't put a GOTO line in the program to set a perpetual loop, your screen will have a green stripe at the top with the OK message.

```

200 A$ = INKEY$
210 IF A$ = CHR$(9) THEN GOSUB 1000
220 GOTO 200

```


This simply tells the Computer that IF you press the  key – the character represented by code number 9 – THEN the Computer will go to a subroutine in line 1000. Here's the subroutine. Type:

```

1000 REM FORWARDS
1010 IF F > 26 THEN RETURN
1020 FOR V = 20 TO 21
1030 FOR H = 0 TO 1
1040 SET (2 + F + H,V,6): SET(20 + F + H,V,6)
1050 SET (18 + F + H,V,3): SET(36 + F + H,V,3)
1060 F = F + 2
1070 GOTO 1000

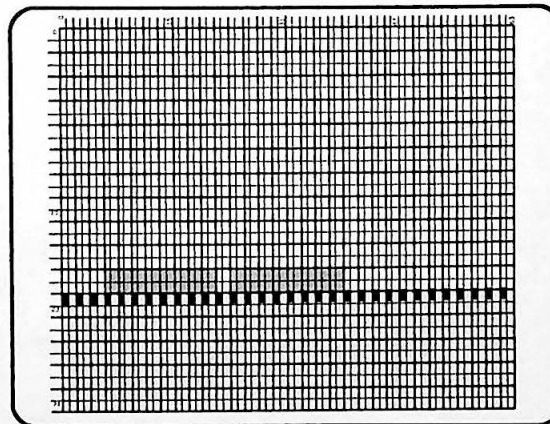
```

F = 0 at this point.

RUN the program. Press the  key and the train will move forwards.

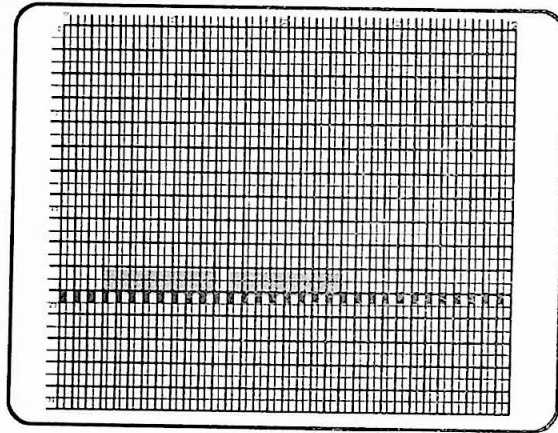
Line 1040 SETs the first block of each car – the blocks beginning at locations 2 and 20 – the color of the sky.

Line 1050 SETs one block ahead of each car – the blocks beginning at locations 18 (2 + 16) and 36 (20 + 16) – the color of the train. After the Computer RUNs line 1050, the screen looks like this:



See how each car has moved over one block. Of course it will only look like this for a split second. Line 1060 adds 2 to F to make it equal 2. Line 1070 sends the Computer back up to do the routine again.

The second time through the routine, the blocks beginning at locations 4 ($2 + F$) and 22 ($20 + F$) are SET the color of the sky and the blocks beginning at locations 20 ($18 + F$) and 38 ($36 + F$) are SET the color of the train. This makes the train move over another block:



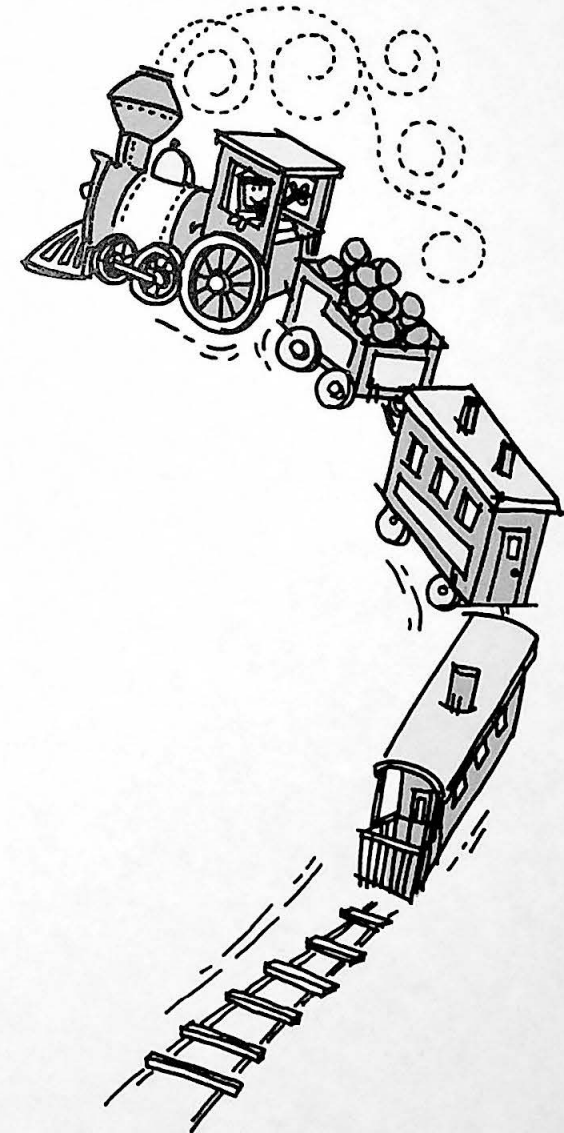
The train continues to “move,” block by block, until it reaches the end of the screen. This happens when F equals 26. After this happens line 1010 will RETURN the Computer back to line 220.



Want to make the train go backwards? Add these lines

```

215  IF A$ = CHR$(8) THEN GOSUB 2000
2000  REM BACKWARDS
2010  IF F < 0 THEN RETURN
2020  FOR V = 20 TO 21
2030  FOR H = 0 TO 1
2040  SET(0 + F + H,V,3): SET(18 + F + H,V,3)
2050  SET(16 + F + H,V,6): SET(34 + F + H,V,6)
2060  NEXT H,V
2070  F = F - 2
2080  GOTO 2000

```



RUN the program. Press  and the train will go backwards. Press  and it will go forwards.

The method we just showed you works great in getting a small and simple image to move. However, if you want to move a larger, more complicated image, you'll prefer the method that we use in Chapter 18.

LEARNED IN CHAPTER 15

BASIC WORD

ASC
CHR\$

NOTES:

Handwritten notes area with horizontal lines.

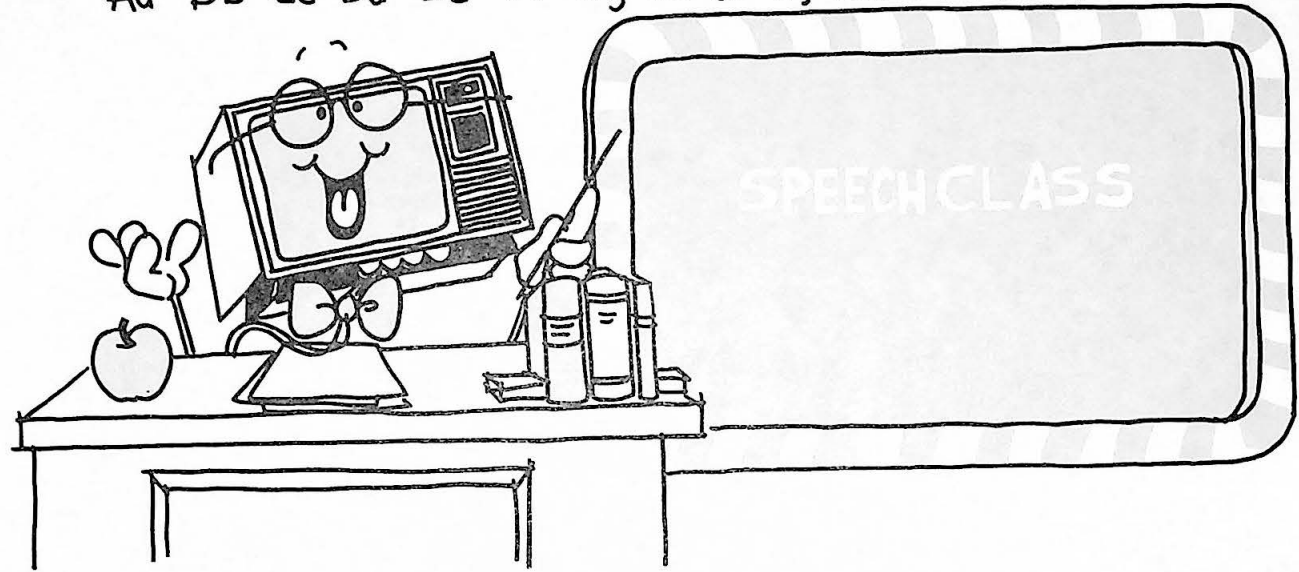
NOTES:

Lined writing area with horizontal lines on both sides of the page.

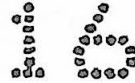


Aa Bb Cc Dd Ee Ff Gg Hh Ii Jj Kk Ll Mm Nn Oo Pp Qq

CHAPTER 16



THE TALKING COMPUTER TEACHER



THE TALKING COMPUTER TEACHER

Who says the Computer can't talk? Its voice, though, will sound strangely like your own . . .

We will get the Computer to talk by using your own taped voice. By doing this, you'll greatly magnify the interest and fun in your programs — particularly games and teaching programs. Even if you don't have a tape recorder, you'll still want to use some of the graphics ideas we have in this Chapter.

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.

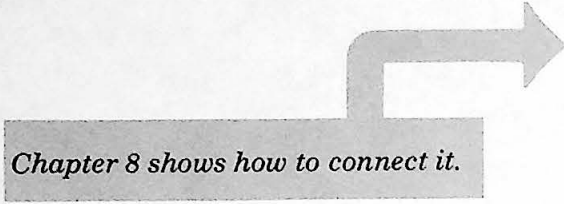
Now type this program:

```
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.

Ready? Before running the program you need to prepare your tape for playing:

- rewind the tape with your recording



Chapter 8 shows how to connect it.

- connect your tape recorder to the Computer
- press the PLAY button on your recorder
- turn up the volume on your T.V.

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 the RESET button. 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 @ 225, "PRESS <X> TO TURN OFF RECORDER"
60 IF A$ <> "X" THEN 40
70 AUDIO OFF
80 MOTOR OFF
```

Prepare your tape for playing and RUN the program.

Line 40 tells the Computer to label whatever key you are pressing or not pressing as A\$. If you are not pressing an "X", line 60 sends program control back to line 40. If you do press an X the recorder's AUDIO connection and MOTOR are turned off.

Now that you understand how it works, you're ready to record the Computer teacher. Ham it up a bit. Here's the script:



SCRIPT

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

(pause for a few seconds)

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

(pause for a few seconds)

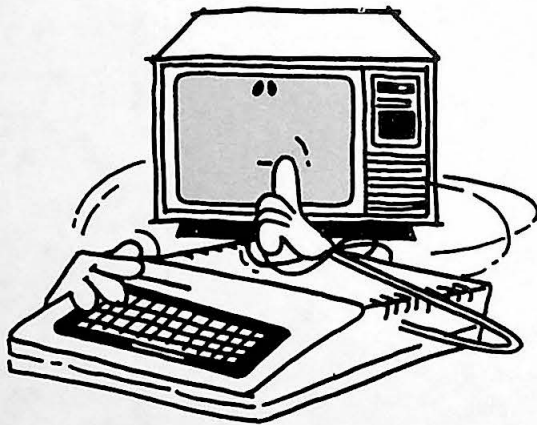
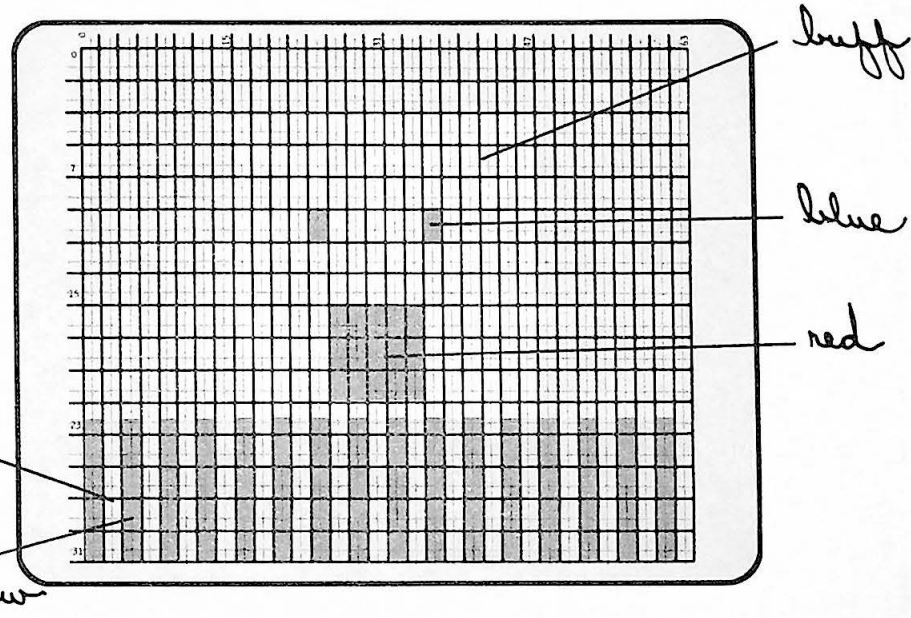
"that's what I'll reward you with when you answer correctly. I won't talk to you again until you give me three correct answers. Press the 'G' key to begin."

(pause for a few seconds)

"Hello again. Now is a good time to start the next lesson. I don't have another lesson, though, so I'm ending the program. Press the 'E' key to turn off the cassette."

Finished? The next thing to do is draw the talking teacher. Here's our grid of what it will look like:

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.



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
```

and RUN. Not as good looking a mouth as mine, but it'll do. Now draw the face. Type:

```

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

```

and the body. Type:

```

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

```

and the eyes. Type:

```

300 FOR V = 10 TO 11
310 SET(24,V,3): SET(25,V,3)
320 SET(36,V,3): SET(37,V,3)
330 NEXT V
340 PRINT @ 0, "THE TALKING COMPUTER TEACHER"

```

RUN it now. Want to make the eyes blink? Type:

```

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

```

and RUN. That's the talking teacher.

Now get it to talk. Type:

```

400 MOTOR ON
410 AUDIO ON
420 A$ = INKEY$
430 IF A$ = "G" THEN MOTOR OFF: END
440 IF A$ = "W" THEN MOTOR OFF: GOSUB 2000
450 IF A$ = "R" THEN MOTOR OFF: GOSUB 3000

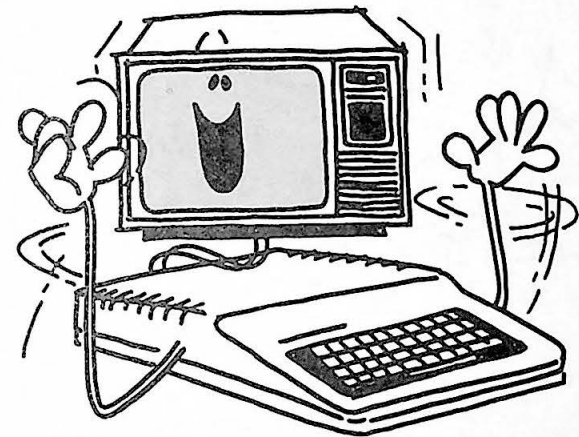
2000 FOR T = 176 TO 89 STEP -10
2010 SOUND T, 1

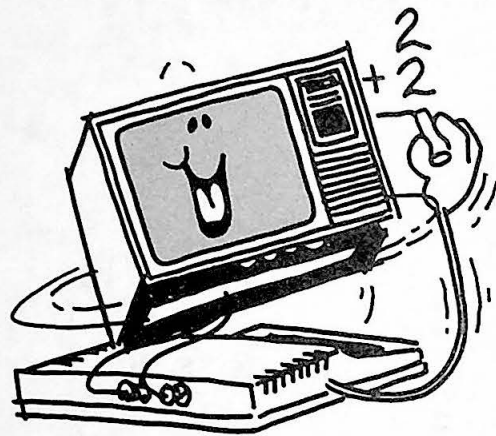
2020 NEXT T
2030 RETURN

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

```

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





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

Before RUNning the program, prepare your tape for playing (rewind it, connect the recorder to the Computer, and press the PLAY button). Now RUN it . . . Do what your voice tells you to do.

Working so far? 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
```

and 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
```

Rewind your tape and press PLAY. RUN the program . . .

There you have it. The Talking Computer Teacher.® Perfect for making arithmetic fun.

LEARNED IN CHAPTER 16

BASIC WORDS

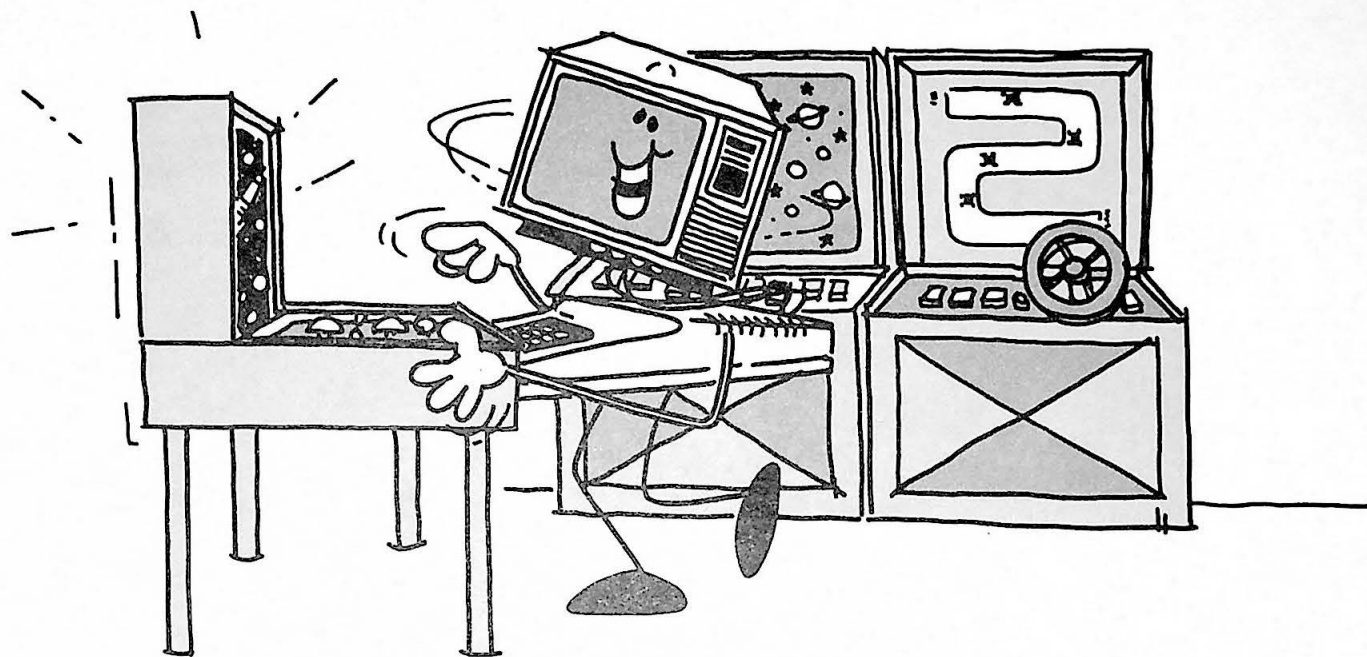
**MOTOR
AUDIO**

NOTES:

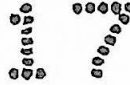
<hr/>	<hr/>
<hr/>	<hr/>
<hr/>	<hr/>
<hr/>	<hr/>
<hr/>	<hr/>
<hr/>	<hr/>
<hr/>	<hr/>
<hr/>	<hr/>



CHAPTER 17



GAMES OF MOTION



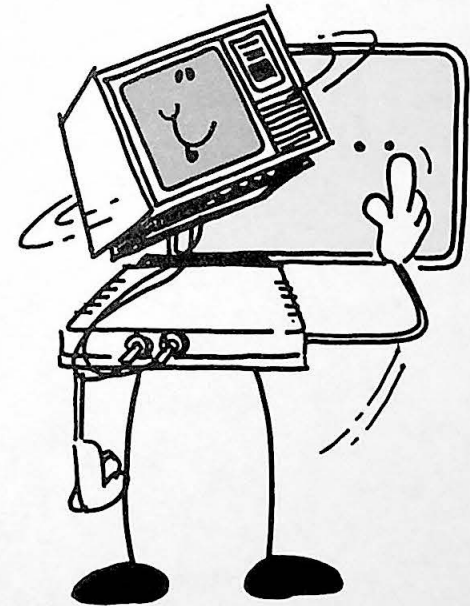
GAMES OF MOTION

Ready to play a little video tennis? How about some target practice or space games? You can teach your Computer to play any of these games as soon as you learn one more BASIC word. That word is POINT, and we're devoting this whole Chapter to it.

Erase memory and type this program:

```
5   CLS(0)
10  FOR X = 1 TO 5
20  SET(RND(64) - 1, RND(30) + 1, 8)
30  NEXT X
40  FOR V = 2 TO 31
50  FOR H = 0 TO 63
60  IF POINT(H,V) <> 0 THEN GOSUB 100
70  NEXT H,V
80  END
100 PRINT @ 0, "LOCATION" H ", " V "IS SET"
110 RETURN
```

In line 60, the Computer scans each POINT (dot) from vertical location 2 through 31 and horizontal location 0 through 63 to see if it is lit up. If it is lit up—that is, the POINT does not equal 0—line 100 prints its horizontal and vertical location.



Delete lines 40, 50, and 70 and change line 10. Type:

```
40
50
70
10 FOR X = 1 TO 300
```

Now change lines 60 and 100, so that if the POINT at location 63, 31 is SET, the Computer will print a message.


PROGRAMMING EXERCISE

This is how we did it:

```
60 IF POINT(63,31) <> 0 THEN GOSUB 100
100 PRINT @ 0, "LOCATION 63, 31 IS SET"
```

Erase memory and type 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
```



RUN it and watch the screen for a while. POINT not only checks to see whether a particular POINT on the screen is lit up, it also checks to see if it is lit up a certain color. The POINT will equal 0 if it is not lit up. If the POINT is lit up it will equal one of the color code numbers listed in Appendix B.

Add two lines to the program so the Computer will also check to see if the POINT at location 31,15 is GREEN or RED.

PROGRAMMING EXERCISE

```
43 IF POINT(31,15) = 1 THEN PRINT @ 160, "LOCATION 31,15  
IS GREEN"  
45 IF POINT(31,15) = 4 THEN PRINT @ 320, "LOCATION 31,15  
IS RED"
```

* PLOTTING THROUGH ASTEROIDS

In this game, we'll be using the right joystick, so make sure it's connected.

We can create asteroids like the way we SET the random POINTs above. Erase memory and type:

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

To SET the planet your ship must reach, type:

```
40 FOR H = 54 TO 63
50 FOR V = 28 TO 31
60 SET(H,V,3)
70 NEXT V,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, we had to add lines 120 and 130.

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 make sure that the first horizontal and vertical dots SET are even numbers and lines 220 through 250 SET the entire block.

RUN the program. Move your joystick around. The cyan colored line will move wherever you position the joystick.

Now make a game out of it. Type:

```
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
```

RUN it again. Each time you hit an orange POINT, the Computer will SOUND a tone.

Notice that line 216 does two things IF the POINT is orange:

- it SOUNDS a tone
- it adds 1 to T, the counter

Add these lines to your 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"
```

and RUN it ... Would you like to have directions printed on the screen? Add these 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 17

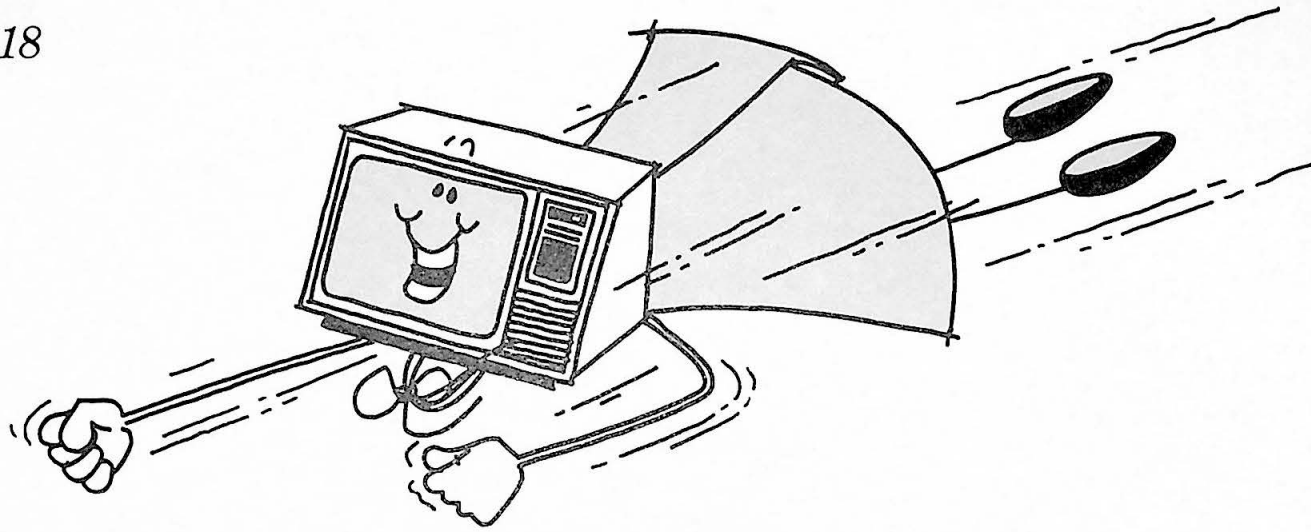
BASIC WORD

POINT

NOTES:

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>
-------------------------------------------------------------	-------------------------------------------------------------

CHAPTER 18



FASTER THAN MOTION

13

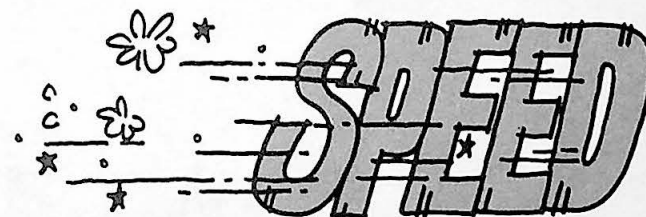
FASTER THAN MOTION

In this Chapter, we'll show you an alternate way to program graphics which we think you'll like. In many cases, it will make programming simpler, and it will definitely speed up your moving picture programs.

Type:

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

The Computer prints a black block which looks like this:



Try some more numbers. Type:

```
PRINT CHR$(129) (ENTER)  
PRINT CHR$(130) (ENTER)  
PRINT CHR$(131) (ENTER)
```

The Computer prints three blocks with different combinations of green and black:



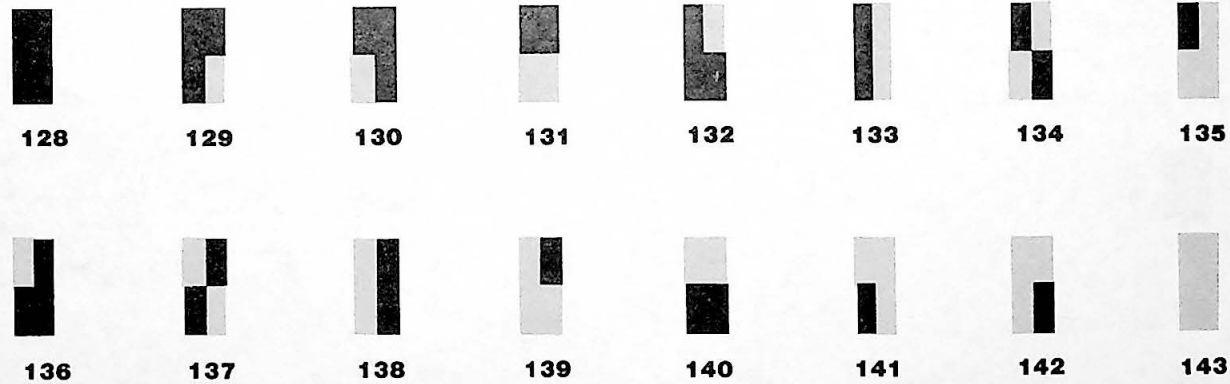
Since the green background makes it difficult to see the outline of the blocks, type this program. It will print the first block against a buff background:

A grid of "PRINT @ Screen Locations" is in Appendix C. (We explained how to use it in Chapter 7). Be sure to type the semi-colon.

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

Remember CHR\$ from Chapter 15? CHR\$ converts a code to the character it represents. For example, CHR\$(65) converts the code 65 to the character "A". The codes above – 128, 129, 130 and 131 – are codes for *graphics characters*.

Look at "Graphics Screen Location" in Appendix D. As we explained earlier the darker lines divide the grid into blocks. Each block contains four dots. These dots can be arranged in sixteen ways to form these *graphics characters*;



To print all these graphics 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 60
70 NEXT C
80 GOTO 10
```

Line 50 prints the graphics characters for codes 128 through 143 at location 240 on your screen.

... Try something a little different. Type:

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

The Computer PRINTs the graphics 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 graphics character codes to create different colors:


16 – yellow
32 – blue
48 – red
64 – buff
80 – cyan
96 – magenta
112 – orange

To see all the different colored characters, add these lines and RUN the program:



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

The semicolon makes the Computer stop printing as soon as it prints your characters. Otherwise, it will continue printing its customary green background for the rest of the line.

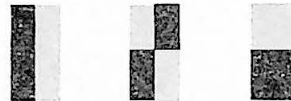


*Notice that these numbers are all multiples of 16. (16 = 16*1; 32 = 16*2; 48 = 16*3 ... 112 = 16*7).*

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

```
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
```

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



PROGRAMMING EXERCISE

Answers:

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

Since these are *characters*, just like A, B, C, and D are, you can treat them the same way as strings. For example, you can combine and store them just as you would combine and store strings. Erase memory and type:

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

and you can position them at, say, the center of the screen in the same way you would position two words – by using PRINT @. Type:


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

and RUN the program. the Computer prints the images of a blue car and an orange truck at the center of your screen.

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



DO-IT-YOURSELF PROGRAM



Note the difference. You PRINT graphics characters using PRINT @ Screen Locations (Appendix C). You SET "dots" using Graphics Screen Locations (Appendix D).

This is how we did it:

```
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
```



TRAFFIC JAM

Erase memory and type:

```
10 A = RND(7) * 16: B = RND(7) * 16
20 A$ = CHR$(129+A) + CHR$(131+A)
30 B$ = CHR$(133+B) + CHR$(143+B) + CHR$(130+B)
```

RUN the program and ask the Computer to print A\$ and B\$. RUN it and print A\$ and B\$ again. Repeat this several times . . .

Each time you run the program, the Computer creates a randomly colored car and truck. Type:

```
40 IF RND(2) = 2 THEN VE$ = A$ ELSE VE$ = B$
```

RUN the program and PRINT VE\$ several times. Sometimes you'll get a car; sometimes a truck. The Computer creates a randomly colored car or truck — at random.

Now you can make a traffic jam. Type:

```
50 IF LEN(TR$ + VE$ + SP$) > 32 THEN 100
60 TR$ = TR$ + VE$
70 GOTO 10
100 PRINT TR$
```

To review LEN, see Chapter 12.

RUN the program several times. Each time, the Computer creates a random traffic jam 32 characters long. To make it move, type:

```
100 INPUT "SPEED(1 - 200)";S
110 FOR P = 0 TO 480
120 PRINT @ P, TR$;
130 FOR X = 1 TO S: NEXT X
140 CLS(0)
150 NEXT P
```

and RUN. The traffic will move from the top left corner to the bottom right corner of your screen.

Line 120 PRINTs the traffic AT location P (0 through 480).

Line 130 puts a pause in the program for the speed you requested.

Line 140 clears the screen so that the traffic can be printed at the next location.

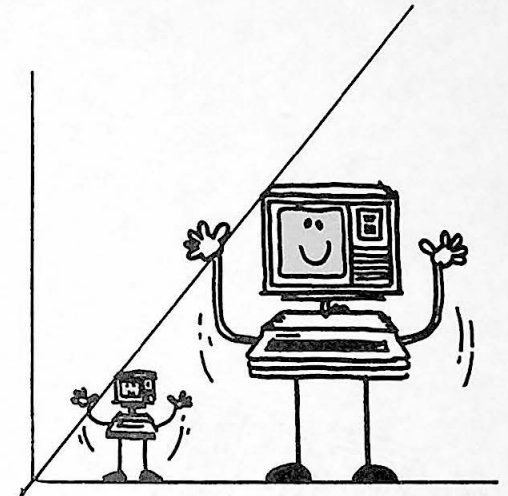
To make the traffic move across your screen perpetually, type:

```
110 P = 320
150 P = P + 1
160 IF P = 351 THEN 110
170 PRINT @ P, LEFT$(TR$, 352 - P);
180 PRINT @ 320, RIGHT$(TR$, P - 320);
190 GOTO 130
```

and RUN.

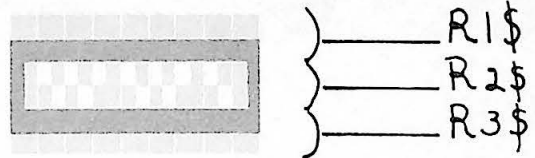
MAKING LARGER PICTURES

Up to now in this Chapter, we have not added height to our graphic images. To print this mouth, we must form three rows of graphic strings:



We show you how to use LEFT\$ and RIGHT\$ in Chapter 12.





To create R1\$ and R3\$, erase memory and type:

```
10 FOR X = 1 TO 9
20 R1$ = R1$ + CHR$(131 + 48)
30 R3$ = R3$ + CHR$(140 + 48)
40 NEXT X
```

RUN the program and PRINT R1\$ and R3\$. R1\$ now equals a string of nine number 131 + 48 graphics characters; R3\$ equals nine number 140 + 48 characters.

To create R2\$, type:

```
50 T1$ = CHR$(137 + 64)
60 T2$ = CHR$(136 + 64)
70 R2$ = CHR$(138 + 48) + T1$ + T1$ + T2$ + T1$ + T1$ + T2$ + T1$ +
CHR$(133 + 48)
```

and to print the entire mouth on your screen, type:

```
80 CLS
90 PRINT @ 5, "LOCATION";
100 INPUT L
110 CLS(0)
120 PRINT @ L, R1$;
130 PRINT @ L + 32, R2$;
140 PRINT @ L + 64, R3$;
150 GOTO 90
```

Line 120 PRINTs the first row – R1\$ – AT L, the location you requested. Let's assume this is location 40.



Line 130 PRINTs the second row – R2\$ AT L + 32, which is location 72. Notice that since there are 32 locations to a row (0-31), location L + 32 is directly under location L.

Line 140 PRINTs R3\$ AT L + 64, which is directly under L + 32 (R2\$'s location).

With this method, we have made each row a string. We could also make the whole mouth a string called MO\$. To do this, we need to combine all the rows – R1\$, R2\$, and R3\$ – plus BK\$, the background between the rows:



Type:

```
72 FOR X = 1 TO 32-9
74 BK$ = CHR$(128) + BK$
76 NEXT X
78 MO$ = R1$ + BK$ + R2$ + BK$ + R3$
```

Since the entire mouth is now one string, you only need one PRINT @ line. Delete lines 130 and 140 and change line 120:

```
120 PRINT @ L, MO$;
```

and RUN it.

By building graphics strings, you'll be able to make your animated programs run fast. In the next Chapter, we'll show you how to make a dancing computer out of these graphic strings.



LEARNED IN CHAPTER 18

BASIC CONCEPT

graphics characters

NOTES:

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>
-------------------------------------------------------------	-------------------------------------------------------------

CHAPTER 19



LET'S DANCE

to reserve plenty of space.

To form the strings made up of black graphics characters, type:

```
10 D$ = CHR$(128) + CHR$(128)
20 G$ = D$ + CHR$(128)
30 B$ = G$ + D$
40 BK$ = B$ + B$ + B$ + D$ + D$
```

RUN the program and ask the Computer to PRINT B\$, D\$, G\$, and BK\$:

PRINT B\$ (ENTER)

██████████

PRINT D\$ (ENTER)

████

PRINT G\$ (ENTER)

██████████

PRINT BK\$ (ENTER)

████████████████████████████████████████

To form the buff colored strings, type:

```
50 C$ = CHR$(143 + 64)
60 F$ = C$ + C$
70 A$ = F$ + C$
```

RUN the program. PRINT A\$, C\$, and F\$:

PRINT A\$ (ENTER)

██████████

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

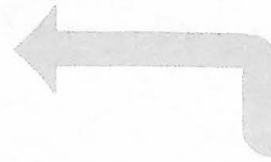
B\$ is actually five characters long. On your screen it will line up with the word PRINT.

D\$ is two characters; G\$ is three; BK\$ is nineteen; A\$ is three.

PRINT C\$ (ENTER)



PRINT F\$ (ENTER)



*C\$ is one character long; F\$ is two;
E\$ is seven.*

To form E\$, the red string, type:

```
80 FOR X = 1 TO 7
90 E$ = E$ + CHR$(143 + 48)
100 NEXT X
```

RUN the program and print E\$:

PRINT E\$ (ENTER)



Form the strings for HD\$, BD\$, and L1\$, so that after RUNNING the program, you can PRINT them like this:

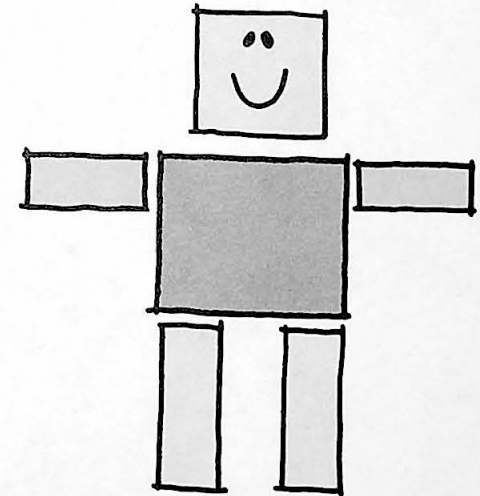
PRINT HD\$ (ENTER)



PRINT BD\$ (ENTER)



PRINT L1\$ (ENTER)

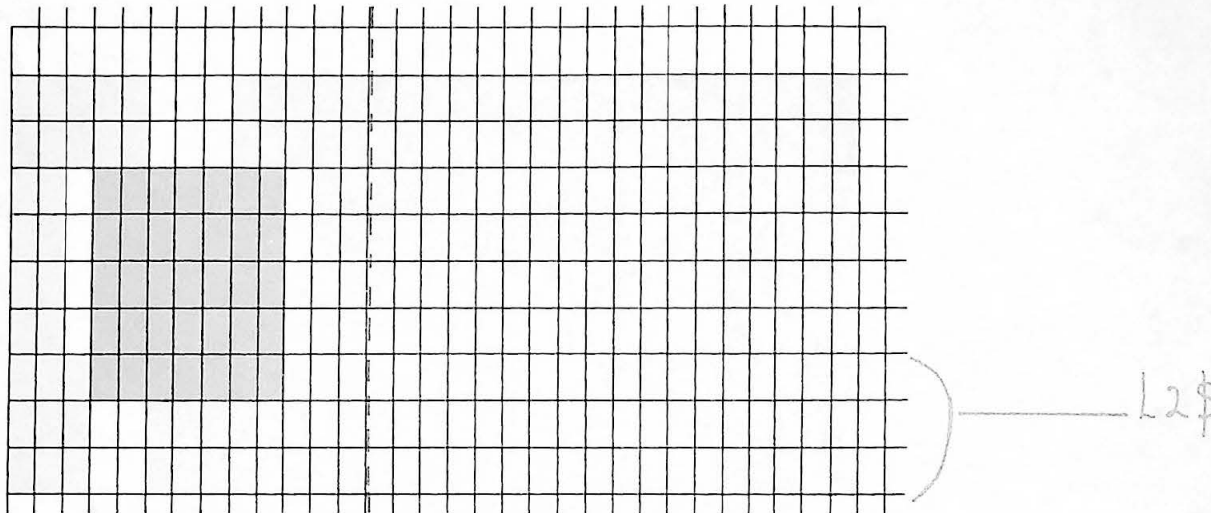


PROGRAMMING EXERCISE

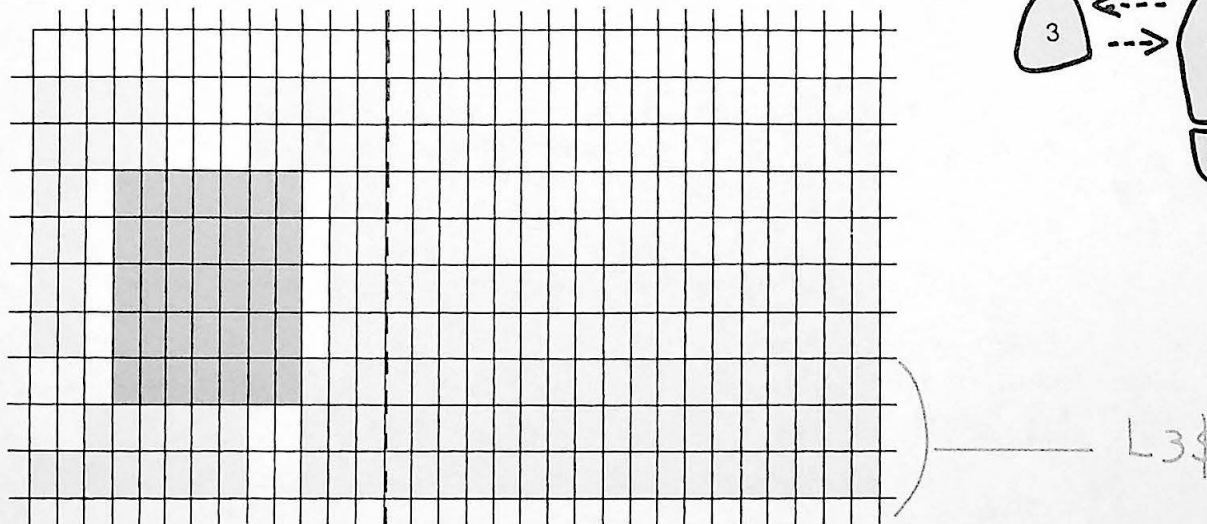
Here is how we did it:

```
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$
```

To make the Computer dance, we'll give it two more leg positions:



Add lines to your program to create the strings L2\$ and L3\$.



PROGRAMMING EXERCISE

We did it this way:

```
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$
```

To see the Computer's three positions, add these lines to your program:

```
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. Try different locations and positions.

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

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

By controlling these locations and positions, you can easily make the Computer move. This is how we did it. Change lines 500 and 510 and add these lines:

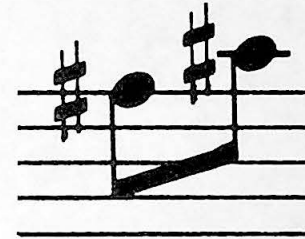
```
500 FOR X = 1 TO 17
510 IF X = 1 OR X = 5 THEN RESTORE

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
```

RUN the program and watch it dance.

Line 515 reads the Location, Position, Tone, and the tone's Duration from lines 5000 through 5060. The first time through the program the Computer will appear at Location 137, Position 2, and, in line 525, will SOUND Tone 89 for a Duration of S times 1. The second time through the program the computer will appear at Location 240, Position 1, and will SOUND Tone 133 for a Duration of S times 2.

As you can see, by adding more positions, you can make this more entertaining. Try it, or look at some of our sample programs in the back for more ideas on how to use graphics characters.



Remember READ and DATA from Chapter 10?

SECTION III

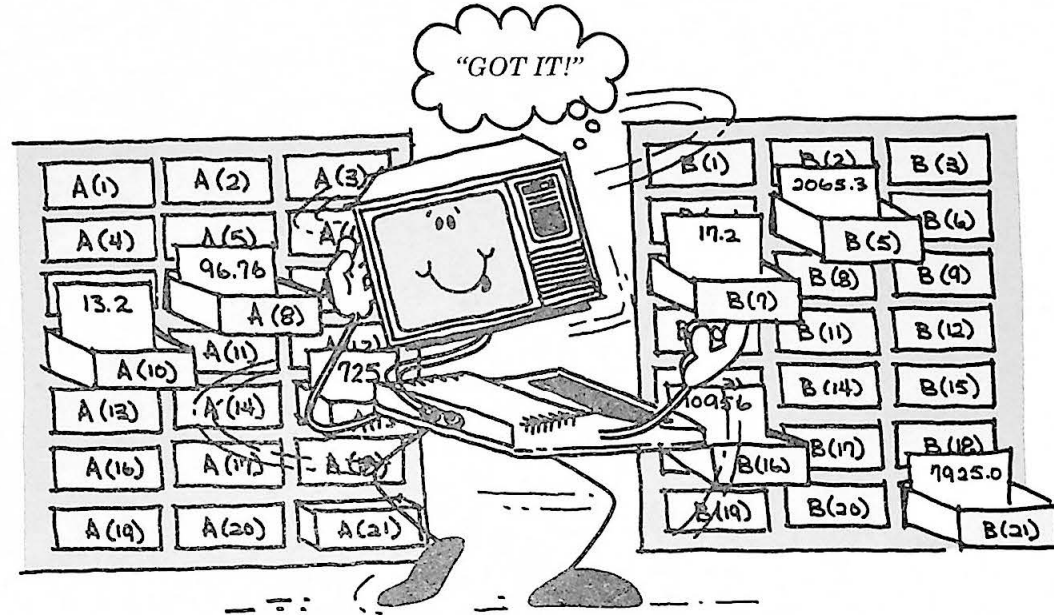
GETTING DOWN
TO BUSINESS

Do you have some lists or files you want the Computer to manage? Here, you'll get the Computer to sort, compare, store, and print your information faster and more accurately than you could ever do it by hand.

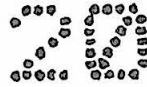
These are some things the Computer has been known to manage:

- | | |
|-------------------------|-------------------------|
| 1. shopping items | 16. inventory |
| 2. checkbook receipts | 17. sales records |
| 3. winter storage items | 18. billing |
| 4. garage sale items | 19. payroll |
| 5. tax records | 20. payable records |
| 6. medical bills | 21. letters |
| 7. addresses | 22. poems |
| 8. phone numbers | 23. songs |
| 9. appointments | 24. essays |
| 10. stock prices | 25. test questions |
| 11. book collections | 26. term papers |
| 12. coin collections | 27. game rules |
| 13. stamp collections | 28. game plays |
| 14. program collections | 29. cards in a deck |
| 15. record collections | 30. card players' hands |

CHAPTER 20



KEEPING TABS ON EVERYTHING!



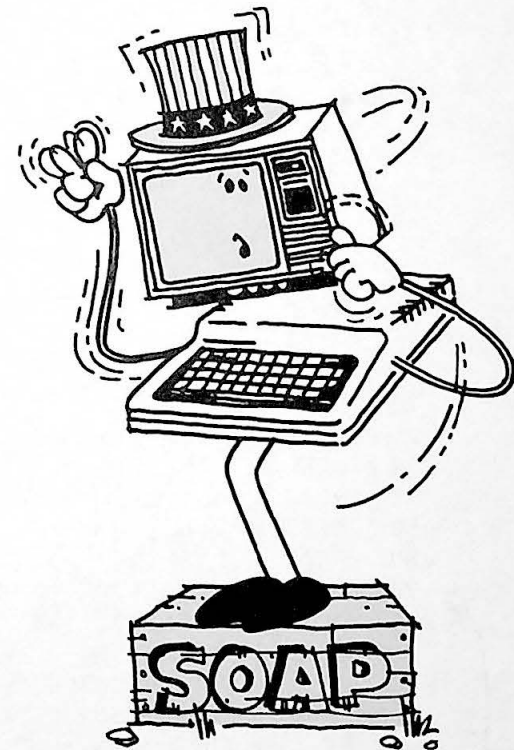
KEEPING TABS ON EVERYTHING!

Have you tried yet to write a program to handle a lot of information? If you have, you'll be happy to know your Computer has a built in "labeling and organizing" system to make this programming much easier!

To begin, how would you get the Computer to remember these election results?

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



For remembering things, we've always used *variables*. To get the Computer to

remember the votes for the first three districts, type:

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

... but there's a much better way. Type this:

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

Each of these variables has a label – A(1), A(2), and A(3). Other than the label, they're the same as the ones above. To see that they work the same, type both of these lines:

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

They both work the same, right? So why are labeled variables better? Take a quick look at these two programs. Both do the same thing:

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
```

Look at Chapter 2 if you want a quick review on variables.

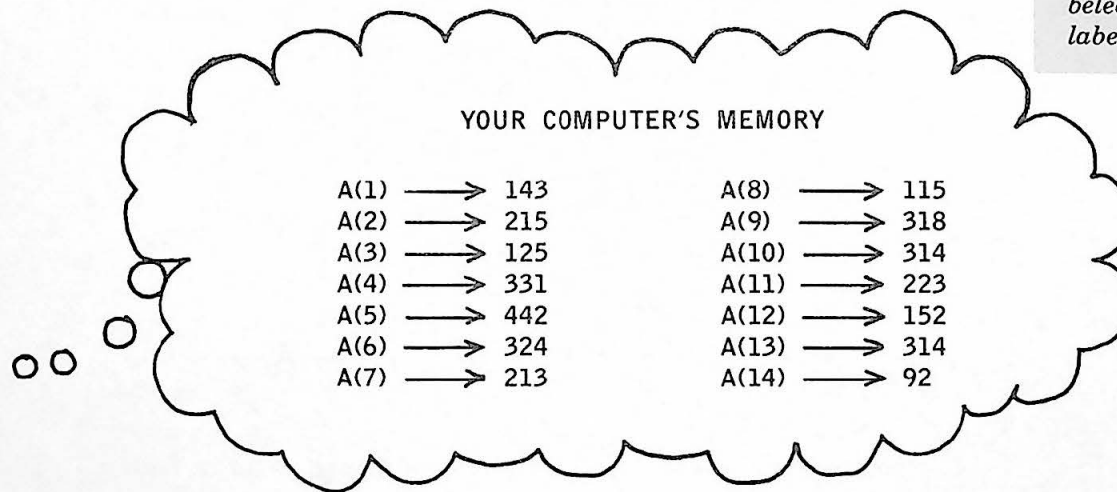
Computer types call an entire list of labeled variables an ARRAY. Each labeled variable is an item in the ARRAY.

The first program uses “regular” variables. The second uses labeled variables. Labeled variables simply make it much easier to program a large list of information.

Type and RUN the second program. Here’s how it works:

Line 40 says make room for a list – an *array* – of information named A with 14 labeled items.

Lines 50 and 70 set up a loop to count from 1 to 14. Line 60 READs this into memory:



Actually, this leaves room for 15 labeled items when you count 0 as a label.

This stores all the votes in an *array* named A. Array A contains 14 labeled items.

Line 80 asks you to INPUT one of the labels, and line 90 PRINTs what you requested.

Now that you have the votes stored in an *array*, it’s easy to do things with them. For instance, if you want your program to be able to change any of the vote totals, you could add these lines:


```

92 INPUT "DO YOU WANT TO ADD TO THIS"; R$
94 IF R$ = "NO" THEN 80
96 INPUT "HOW MANY MORE VOTES"; X
97 A(Z) = A(Z) + X
98 PRINT "TOTAL VOTES FOR DISTRICT" Z "IS NOW" A(Z)

```

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

Or if you want to PRINT a table like the one at the beginning of this chapter, you could add these lines:

```

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

```

You don't need to study these programs if you're anxious to move on. We're just showing some benefits of using these variables with labels.

and change line 100:

```

100 GOTO 72

```

MAKING ROOM FOR CANDIDATE B

Let's assume you also want to keep track of the votes for 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

District	Votes for Candidate A	Votes for Candidate B
10	314	518
11	223	370
12	152	412
13	314	460
14	92	502

We can simply add another array to our program for candidate B. We'll call this array B. This 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)
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

data for array B

saves room

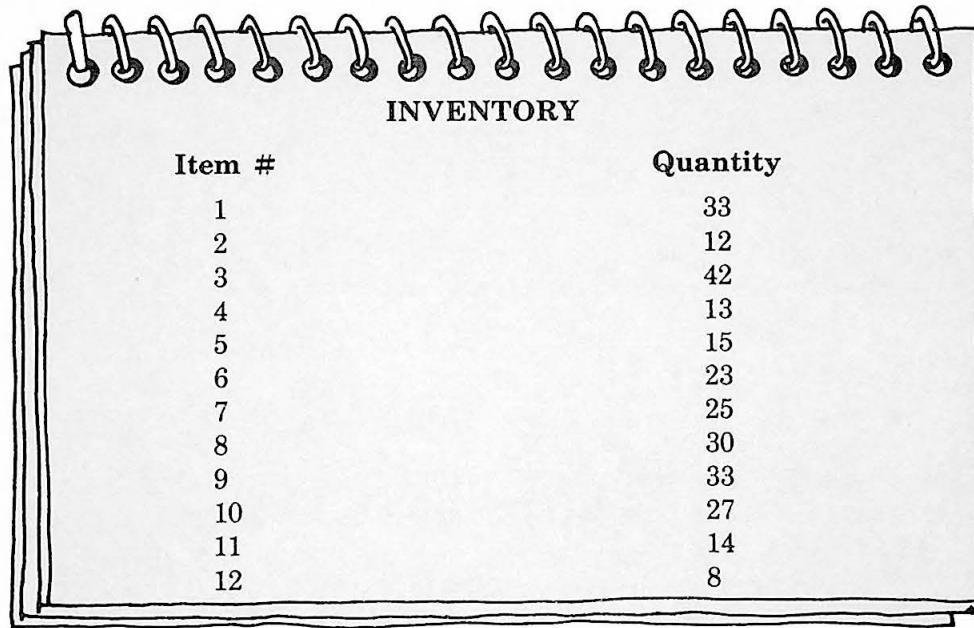
Reads array A data

Reads array B data



KEEPING INVENTORY

Arrays are often used to keep track of business records. Write a program to help a store keep track of its inventory of 12 items:



The image shows a spiral-bound notebook with a page titled "INVENTORY". The page contains a table with two columns: "Item #" and "Quantity". The items are numbered 1 through 12, and their corresponding quantities are listed to the right.

Item #	Quantity
1	33
2	12
3	42
4	13
5	15
6	23
7	25
8	30
9	33
10	27
11	14
12	8

DO-IT-YOURSELF PROGRAM

Here's the program we wrote:

```
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
```

MEMORY TEST

Ready for a breather. This program tests both yours and your Computer's memory. Type NEW to erase your program and type:

```
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
```

Line 5 saves room for an array named A with 7 items.

Lines 20 through 50 assign seven random numbers to the array.

Actually, you don't need this DIM line if none of your array items use a label higher than 10. However, it's still a good idea to put this line in your program to reserve just the right amount of memory.

Remember, you can put instructions on one line, separating them with a colon. These two instructions put a ten second pause in the program.

The Computer uses an array to memorize these numbers. What are you using?

Line 60 puts a ten second pause in the program, and line 70 clears the screen.

Lines 80 through 120 quiz you on each of the seven numbers in array A.

DEAL THE CARDS

Watch carefully while we show you how to use an array to get the Computer to deal the cards.

*If you like, try it on your own first.
We must warn you though – it's
tricky.*

To deal 52 random cards, erase your program and type:

```
40 FOR X = 1 TO 52
50 C = RND(52)
90 PRINT C;
100 NEXT X
```

RUN the program . . . The Computer deals 52 random “cards”. However, if you look closely, you’ll see that some of the cards are the same.

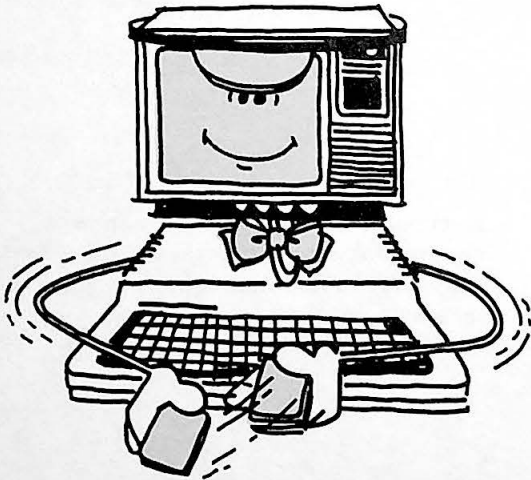
Somehow, we’re going to have to somehow keep track of which cards have been dealt. To do this, we will first build an array named T which contains all 52 cards. Type:

```
5 DIM T(52)
10 FOR X = 1 TO 52
20 T(X) = X
30 NEXT X
```

This simply tells the Computer that $T(1) = 1$, $T(2) = 2$, $T(3) = 3$ and so forth through $T(52)$ which, of course, equals 52.

Now add these lines, which will zero out each card in array T after it is dealt. Type:

```
60 IF T(C) = 0 THEN 50
80 T(C) = 0
```



Now the Computer can't deal the same random card twice. For example, say the first card the Computer deals is 2. Line 80 changes the value of T(2) from 2 to 0. Now say the Computer deals another 2. Line 60 says that since T(2) now equals 0, the Computer must go back to line 50 and deal another card.

RUN the program. Notice how the Computer hesitates before printing the last cards. This is because it's trying many different random cards first before finding one which hasn't been dealt.

If you want to play card games with the Computer, you'll need to get it to remember which cards it has dealt. To do this, we can create another array. We'll name it array D. Type:

```
7 DIM D(52)
70 D(X) = T(C)
90 PRINT D(X);
```

Now array D contains a list of all the cards the Computer dealt in the order that it dealt them.

How would you change this program so it would only print your "hand" — the first 5 cards dealt?




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

DO-IT-YOURSELF PROGRAM

Here's ours:

```
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
```

Line 130 tells the Computer that whatever it PRINTs next should be at location 167.



We will show you a lot more things you can do with arrays in the next chapters.
This is all you need to remember:

RULES ON ARRAYS

1. There are two kinds of variables:
 - A. *SIMPLE VARIABLES*, such as A, B, C, and D.
 - B. *"LABELED" VARIABLES* or *ARRAY ITEMS* such as A(5), A(3), B(2), and B(6).
2. An *ARRAY* is a group of labeled items which each have the same variable name. For example, M(2), M(4), M(5), and M(6) all belong an array named M.

LEARNED IN CHAPTER 20

BASIC WORD

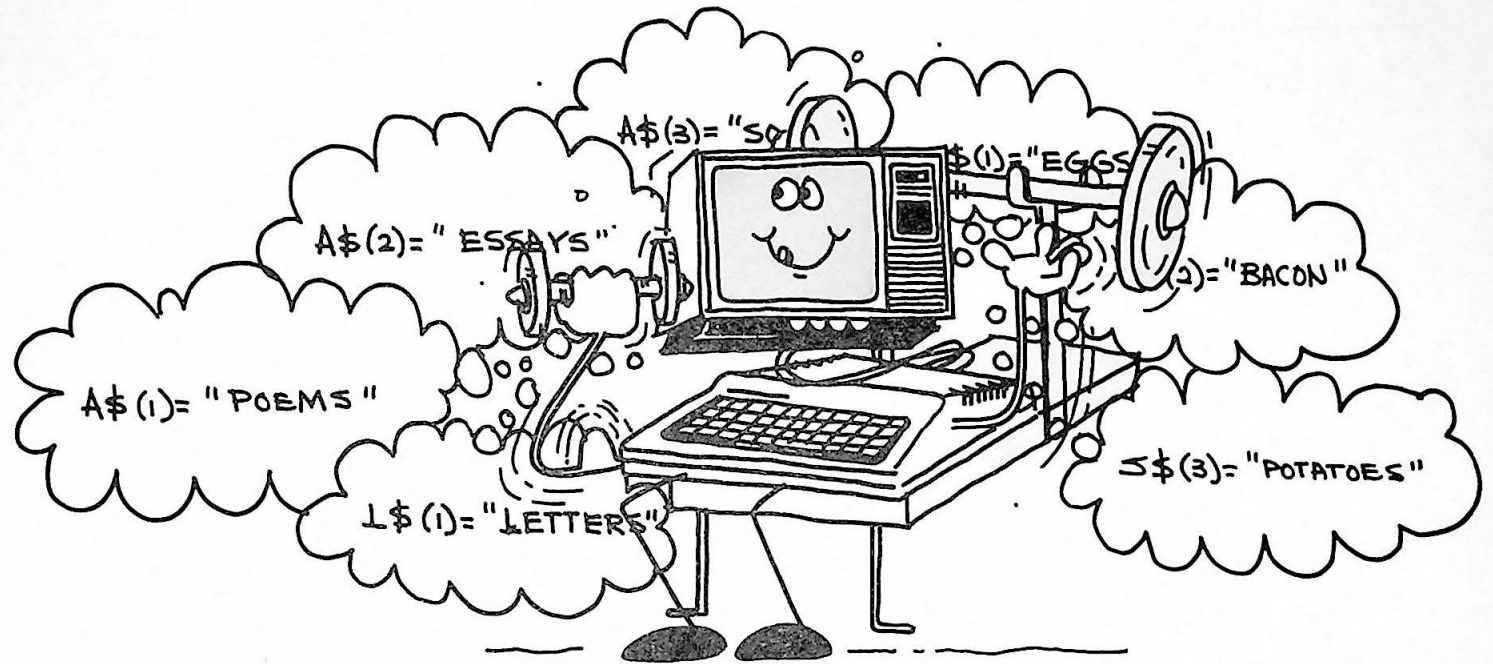
DIM

BASIC CONCEPT

arrays

NOTES:

CHAPTER 21



PUT POWER IN YOUR WRITING



PUT POWER IN YOUR WRITING

In the last chapter, we only used arrays for lists of numbers. But arrays are also for words. Not just a simple list of words, but as you'll see in this Chapter, your Computer can remember, edit, and print an entire essay filled with words.

We'll 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 |

To get the Computer to remember each of these items, we'll assign each one to a labeled *string* variable. For example, for the first three items, you could type:

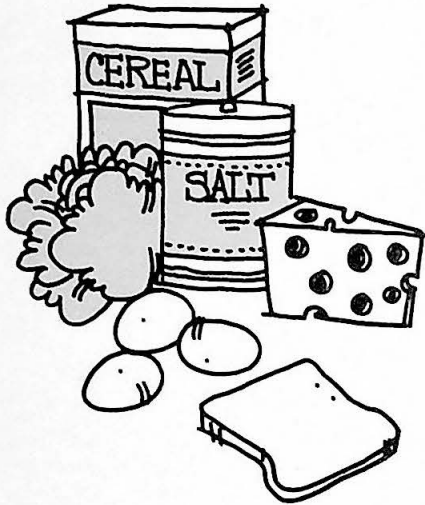
```
S$(1) = "EGGS" (ENTER)  
S$(2) = "BACON" (ENTER)  
S$(3) = "POTATOES" (ENTER)
```

See the dollar sign? That's the only difference between these labeled variables and the ones in the last chapter.

To get the Computer to PRINT these first three items, type:

```
PRINT S$(1), S$(2), S$(3) (ENTER)
```

Here's how to put them in a program:



```
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
```

Reads items into array S\$

Prints array S\$

This program puts all 12 items into an array named S\$ and PRINTs the list. Add some lines to this program so that you can change any of the items on this list.

DO-IT-YOURSELF PROGRAM

Here are the lines we added:

```
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
```

In the back, we show how to add or delete items from this list.

WRITING SONG LYRICS (... A POEM, A LETTER, ETC. ...)

Here's a program using an array to help you write song lyrics. Erase memory by typing NEW and type:

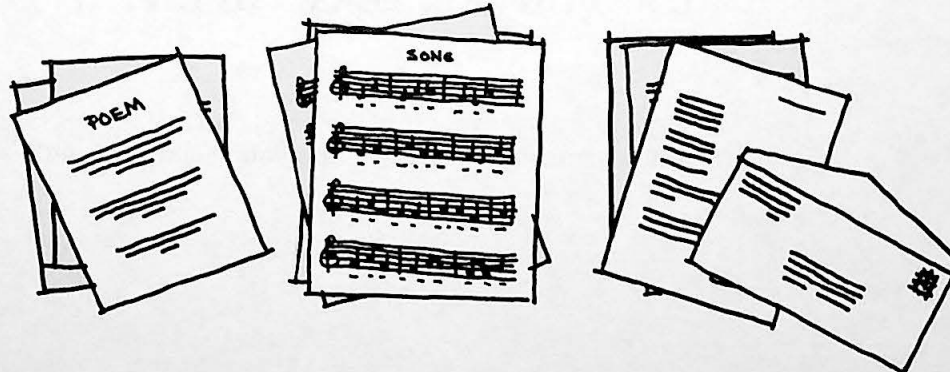
```
5   DIM A$(4)
10  PRINT "TYPE 4 LINES"
20  FOR X = 1 TO 4
30  INPUT A$(X)
40  NEXT X
50  CLS
60  PRINT "THIS IS YOUR SONG:"
70  PRINT
80  FOR X = 1 TO 4
90  PRINT X; " "; A$(X)
100 NEXT X
```

Inputs items into array A\$

Prints array A\$

Want to compose music? Look up "Music Composer" in the Sample Programs Appendix in the back of the book.

RUN it. Add some lines so that you can revise any of the lines in the song.



DO-IT-YOURSELF PROGRAM

Here are the lines we added:

```
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
```

*Haven't heard of word processing?
It's a way of getting the Computer to
memorize what you type, make
changes to it, and print it out on de-
mand.*

WRITING AN ESSAY (... A NOVEL, TERM PAPER ...)

Here is a better program to help you with your writing. Use it along with what you learned in Chapter 12 and you've got yourself a word processing 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

```

A\$ = "/"

Y

Need a refresher on some of this? We talk about CLEAR in Chapter 12 and INKEY\$ in Chapter 13.

Type and RUN the program. Before reading how the program works, try experimenting with it. Type:

```

PRINT A$(1) (ENTER)
PRINT A$(2) (ENTER)
PRINT A$(3) (ENTER)

```

Got an idea how it works? Here is a run down:

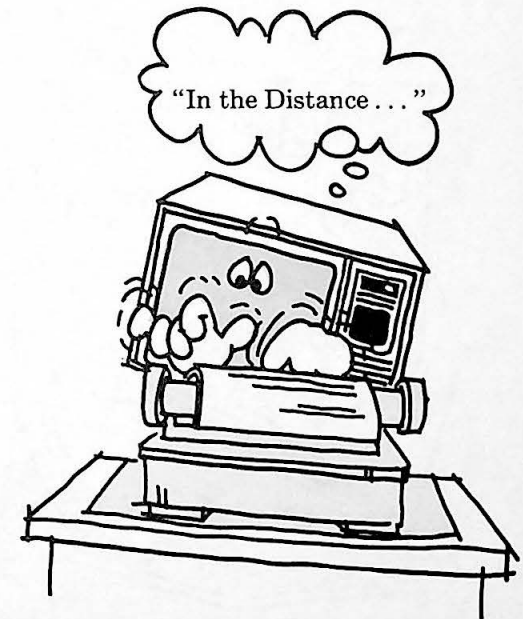
Line 1 CLEARs plenty of string space for the Computer to use.

Line 5 saves room for an array named A\$ which 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 – remember, "" is nothing – line 50 sends the Computer back up to line 40.

Line 60 prints the key you pressed.



Line 70 sends the Computer to the lines which 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”

This might continue until A\$(1) equals “ROSES ARE RED”. At this point, you press “.” A\$(1) now equals your entire sentence – “ROSES ARE RED.” The next letter you press will belong to A\$(2).

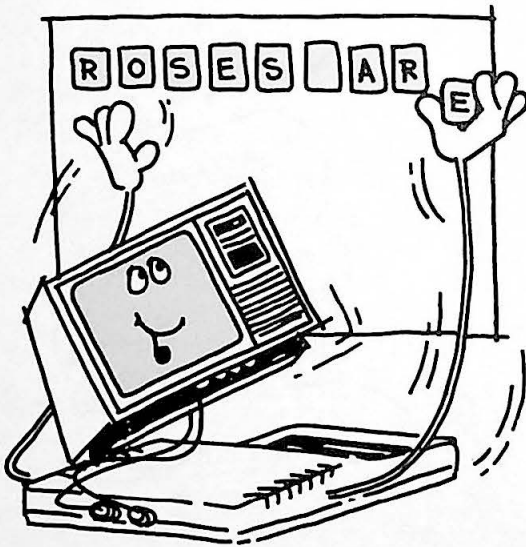
Lines 140 through 160 PRINT your paragraph.

A WORD PROCESSOR CHALLENGER

Here’s a tough one (. . . but it *can* be done! . . .) for those intrigued with word processing. Make this into a full fledged word processing program which:

1. PRINTs whatever sentences you want
2. Lets you revise your sentences

You’ll need to review the challenger program we have at the last of Chapter 12. Our program’s at the back of the book.



DO-IT-YOURSELF CHALLENGER PROGRAM

FOR THOSE WITH A PRINTER

If you have a printer, you're probably ready to put it to use. Connect the printer's cable to the jack marked SERIAL I/O on the back of your keyboard. Power it up and insert paper. The manual that comes with the printer shows you how.

Ready? Type this short program:

```
10 INPUT A$  
20 PRINT # -2, A$
```

Now type:

```
LLIST ENTER
```

... and watch the printer work. This sure will make listing long programs easy! If your program doesn't list on the printer, make sure the printer is ON, ON-LINE, and connected to your keyboard. Try typing LLIST again.

Now RUN the program ... INPUT whatever you want 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 include the comma after the 2, or you will get a syntax error.

Press the (SHIFT) and (0)(zero) keys simultaneously so that the letters you type appear in reversed colors on your screen (green with a black background). You are now in an upper/lower case mode. The reversed colored letters are actually lower case (non capitalized) letters.

Type a letter while holding the (SHIFT) key down. It is a capital letter, so it appears in regular colors.

RUN your program by pressing the (SHIFT) key while you type RUN:

RUN (ENTER)

INPUT a sentence with both upper and lower case letters. Type:

M Y PRINTER PRINTS LOWER CASE LETTERS

A printer with upper and lower case letters would come in handy on a word processor program. Look at the one we discussed earlier in this chapter. How would you change lines 140-160 so that your paragraph is printed on the printer rather than your screen?

.....

Got the answer? You would simply change line 150 to:

150 PRINT # -2, A\$(Y);

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

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

LEARNED IN CHAPTER 21

BASIC WORDS

LLIST
PRINT #-2

BASIC CONCEPT

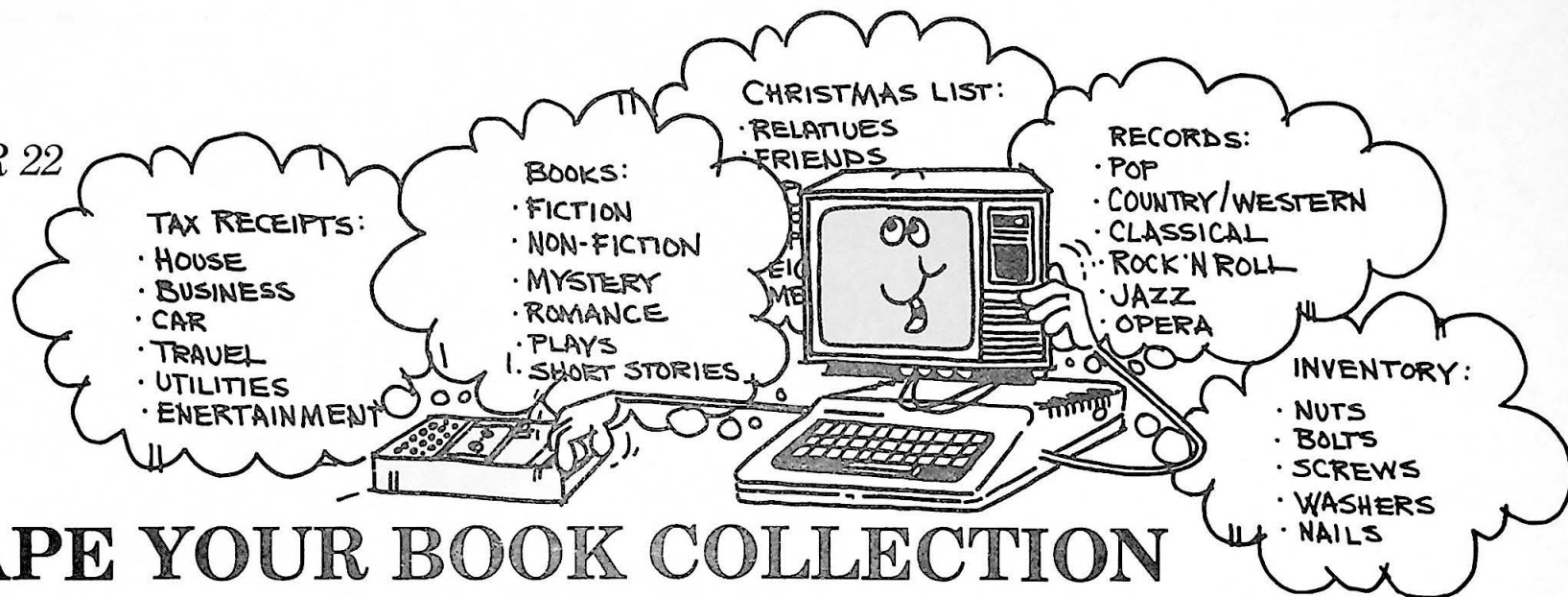
string arrays

NOTES:

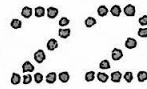
Handwritten notes area with horizontal lines.



CHAPTER 22



**TAPE YOUR BOOK COLLECTION
(or your records, Christmas list,
tax receipts, inventory . . .)**



TAPE YOUR BOOK COLLECTION

(or your records, Christmas list, tax receipts, inventory . . .)

You know you can store programs on tape. You can also use tape to store any lists you want organized. Once you have a list on tape, you can use the time saving power of the Computer to print it, change it, add to it, or analyze it anytime you want.

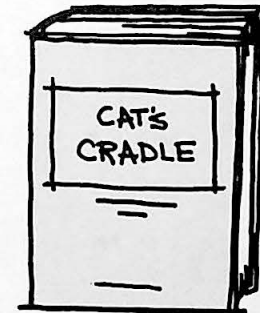
Ready to get organized? We'll start with your books. Here's a very small book list:

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

To put this list on tape, and to read it back into your Computer's memory from tape, you need a program. You have to type the whole program before you can see how it works, so be patient with us.

Begin by typing:

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



*The "0" stands for OUTPUT-
TING.*

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

This tells the Computer to OPEN the lines of communication to device #-1 — the tape recorder. We're going to be sending out — *outputting* — a *file* of information and storing it all under the name BOOKS.

Now type:

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

This lets you INPUT T\$, the title of a book, over and over again. Each time you INPUT T\$, the Computer PRINTs it — *not on the screen, but to device #-1 which is the tape recorder.*

Add these lines:

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

This permits you to type XX when you've finished typing all your book titles. The Computer then CLOSEs communication to device #-1, the tape recorder.

Add three more lines:

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

That's the program. Before RUNning it, you need to:

- Connect your tape recorder. Chapter 8 shows you how.
- Position a tape in the recorder and rewind it to the beginning.
- Press the RECORD and PLAY buttons so that they are both down.

Ready? LIST your program to see if it still looks like ours:

Actually, you can record anywhere on tape you want.

If you're not using a RADIO SHACK tape, make sure you position it past the beginning leader.

```

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

```

Opens lines to recorder

Prints titles on tape

Closes lines to recorder

Got it typed OK? . . . RUN it.

Notice that as soon as you press **ENTER**, the cassette motor turns on. The Computer is OPENING a "file" on tape and naming it BOOKS.

When it asks you for titles, INPUT the four titles we have above and then type XX:

```

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

```

The Computer will clear 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've finished, the tape recorder motor will run again. The Computer is PRINTing all the titles on the tape (line 30) and then CLOSEing communication to the tape recorder (line 50).

Now all the titles are *output* to tape. To load or *input* them back from tape, type:

```

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

```

The "I" stands for INPUT

Line 80 OPENS communication to the tape recorder for a *file* of information named BOOKS. This time, rather than being OPEN for *output*, communication

is OPEN for *input* from the tape recorder.

Add these lines:

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

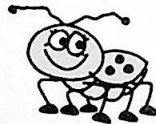
Line 90 *inputs* the first title – T\$. Again, T\$ is not *input* from your typing it on your keyboard. It is *input* from the tape recorder. Line 100 PRINTs T\$ on your screen.

Now add these lines:

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

Line 85 says if you are at the End Of this File of BOOKS then go to 120, which CLOSEs communication with the tape recorder.

Are you wondering what the -1 means? EOF returns a -1 if you have reached the end of the file.



BUG: Be sure to put the EOF(-1) line before the INPUT #-1 line. Otherwise, you'll get an IE error – Input past the end of the file.

List this last part of the program by typing:

```
LIST 60 -
```



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"
85 IF EOF (-1) THEN 120
90 INPUT #-1, T$
100 PRINT T$
110 GOTO 85
120 CLOSE #-1
```

Opens lines to recorder
Inputs titles from tape
Closes lines to recorder

Now RUN it. Type:

RUN 60

Follow the Computer's instructions . . .

When you press **ENTER**, notice that the tape recorder motor is running. The Computer is *inputting* your items from tape. Once they are *input*, the Computer PRINTs the four items on your screen.

Want a quick review?

```
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
60 CLS: PRINT "REWIND THE RECORDER AND PRESS PLAY"
70 INPUT "PRESS <ENTER> WHEN READY"; R$
80 OPEN "I", #-1, "BOOKS"
85 IF EOF (-1) THEN 120
90 INPUT #-1, T$
100 PRINT T$
110 GOTO 85
120 CLOSE #-1
```

Outputs
titles to
tape

Inputs
titles from
tape

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

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.

In line 30 we PRINTed T\$ (the title of the books) on tape. To do this, we had to OPEN communication to the tape recorder for *output*. After finishing the “output” we had to CLOSE communication with the tape recorder.

In line 90 we INPUT T\$ from the tape recorder. To do this, we had to OPEN communication with the tape recorder for *input*. After finishing the *input*, we CLOSEd communication.

Understand it? Think over the answers to these three questions . . .

QUESTIONS

1. What would happen if you leave out line 50 and RUN the program?

.....

ANSWER: *Without line 50, communication with the tape recorder remains OPEN for OUTPUT. Since it's already OPEN, the Computer will not let you OPEN it again for INPUT. Therefore, line 80 will give you an AO error – Attempt to Open a file that's already open.*

2. Would it be O.K. to leave out both lines 50 and 80 and RUN the program?

.....

ANSWER: *This won't work either. Without lines 50 and 80, communication remains OPEN for OUTPUT. When line 90 asks the Computer to INPUT from the tape recorder, you'll get a NO error – File not Open. The file is not OPEN for INPUT.*

3. Would it work if you changed lines 90 and 100 to:

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

.....

ANSWER: *Yes it does work. The Computer doesn't care that you called the titles T\$ when you put them on tape. When you're INPUTting them from tape, the Computer simply looks for a string variable on tape and labels it X\$.*

AN ELECTRONIC CARD CATALOG

Getting ambitious? How about changing the program so you can put all of this on tape:

TITLE	AUTHOR	SUBJECT
WORKING	Studs Terkel	Sociology
CAT'S CRADLE	Kurt Vonnegut	Fiction
SMALL IS BEAUTIFUL	E. F. Schumacher	Economics
STEPHENWOLF	Hermann Hesse	Fiction

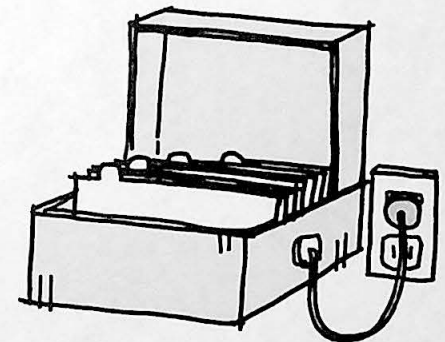
First, we'll work with the first half of the program — the part that *outputs* to tape. Add these lines to the program:

```
26 INPUT "AUTHOR"; A$  
28 INPUT "SUBJECT"; S$  
29 IF A$ = "XX" OR S$ = "XX" THEN 50
```

To PRINT all of this on tape, simply change line 30:

```
30 PRINT # -1, T$, A$, S$
```

Now for the second half of the program. How would you change line 90 and 100 so that the Computer *inputs* from tape and PRINTs the title, author and subject?



PROGRAMMING EXERCISE

Here's the way we did it:

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

Like we said earlier, you don't have to use the same variable names you used when *outputting*. This would also work:

```
90  INPUT #-1, X$, Y$, Z$
100 PRINT "TITLE : " X$
102 PRINT "AUTHOR : " Y$
104 PRINT "SUBJECT : " Z$
```

PICK A SUBJECT

Now you can take advantage of all this organization. For example you might want to have the Computer print a list of books on any given subject.

Add these lines to your program:

```
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, T$, A$, S$
210 IF S$ = C$ THEN PRINT T$, A$
220 GOTO 190
230 CLOSE #-1
```



and RUN it by typing RUN 130. If you choose Fiction, the RUN should go like this:

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

FICTION BOOKS:

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

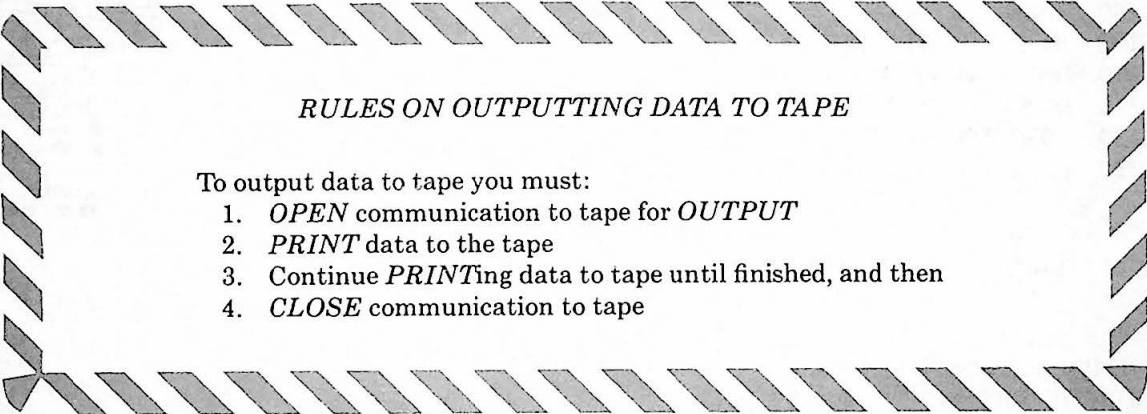
BALANCING YOUR CHECKBOOK

Now its your turn to try it. Say 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 which 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.

Remember how to *output* information to tape:



RULES ON OUTPUTTING DATA TO TAPE

To output data to tape you must:

1. *OPEN* communication to tape for *OUTPUT*
2. *PRINT* data to the tape
3. Continue *PRINTing* data to tape until finished, and then
4. *CLOSE* communication to tape

and how to *INPUT* information from tape:



RULES ON INPUTTING DATA FROM TAPE

To *INPUT* data from tape, you must:

1. *OPEN* communication to tape for *INPUT*
2. Use EOF (-1) to see if you've reached the end of the file on tape.
3. *INPUT* data from tape
4. Continue *INPUTting* data until you've finished or have reached the end of the file, and then
5. *CLOSE* communication to tape

DO-IT-YOURSELF PROGRAM

Here is what we wrote:

```
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
```

Now that you've got an understanding of how to put your information on tape, you might want to look at some of the sample programs in the back of the book. They'll give you some more ideas on how to use these tape "files".

LEARNED IN CHAPTER 22

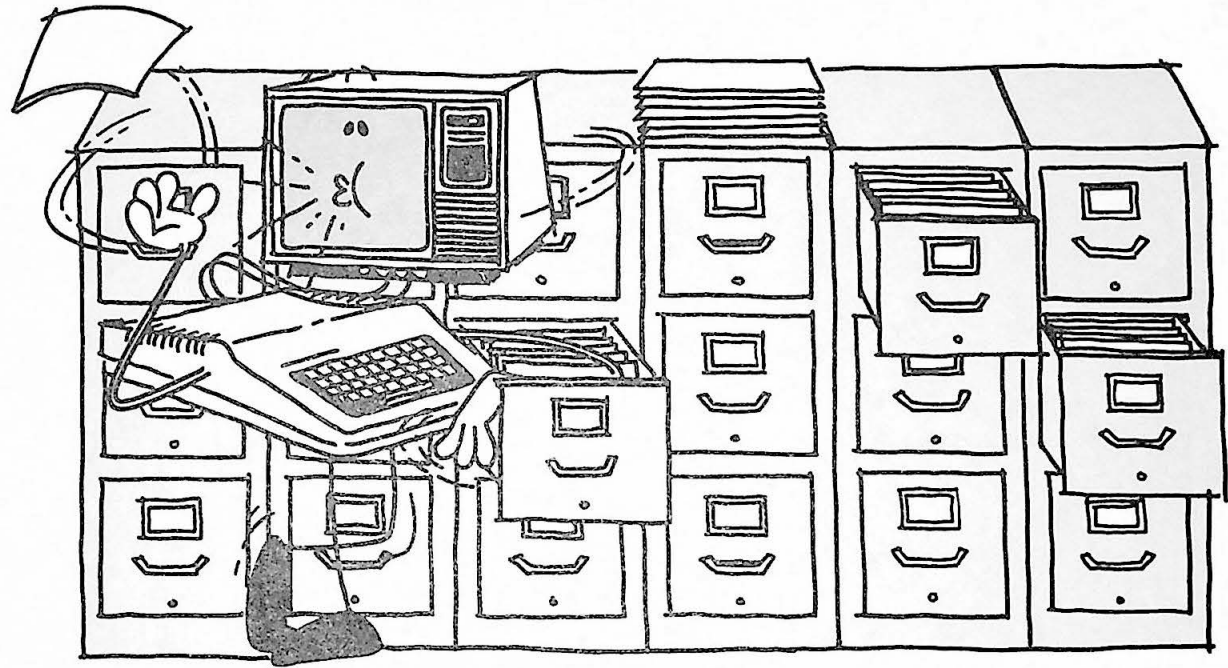
BASIC WORDS

OPEN
CLOSE
PRINT #-1
INPUT #-1
EOF

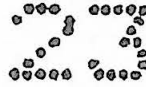
BASIC CONCEPT

data files

CHAPTER 23



FILING – AS EASY AS ABC



FILING – AS EASY AS ABC

Any file clerk can tell you it's much easier to find things if you have them in alphabetical order. To save you the tedium of alphabetizing, why not have the Computer do it? Type this program:

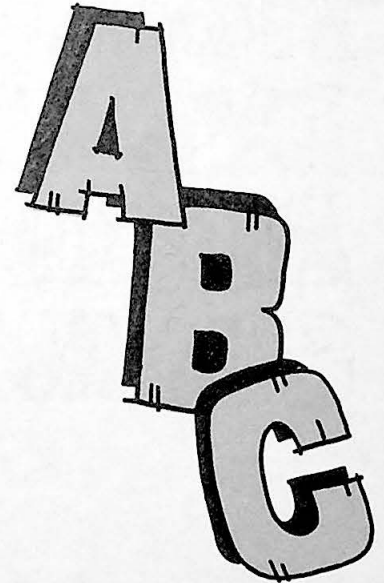
```
10 INPUT "TYPE TWO WORDS"; A$, B$
20 IF A$ < B$ THEN PRINT A$ " COMES BEFORE " B$
30 IF A$ > B$ THEN PRINT A$ " COMES AFTER " B$
40 IF A$ = B$ THEN PRINT "BOTH WORDS ARE THE SAME"
50 GOTO 10
```

RUN the program. Keep INPUTting words until you're convinced the Computer knows how to alphabetize.

With strings, the greater than, less than, and equal signs that we discussed in Chapter 11 take on 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, you can write a program to alphabetize a long list of words. Here is ours:



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

```
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
```



MICHAEL
TRAVIS
DYLAN
ALEXIA
SUSAN

Type and RUN this program.

Before explaining how it works, we'll show what's happening when the program is RUN. Type:

```
30 READ A$(I)
200 DATA MICHAEL, TRAVIS, DYLAN, ALEXIA, SUSAN
```

so that we'll both alphabetize the same words. Delete line 120 and type:

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

These lines are just for appearance – so you can see what is happening in the program. You don't need to study them. Just type them like they are. They don't have anything to do with alphabetizing.

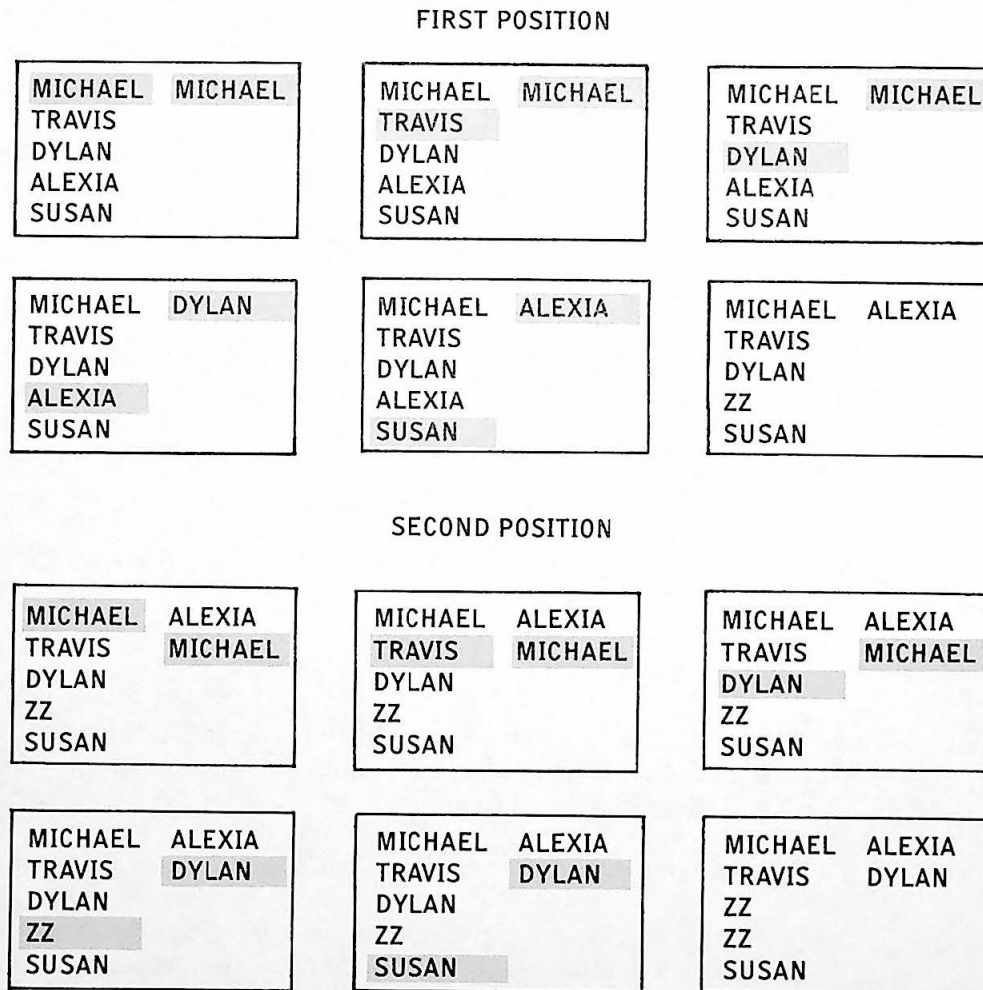
RUN the program.

Too fast? Type this line. It'll slow it down so you can see what's happening:

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

RUN it again and watch carefully. Look at the second column. 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.

Here's a picture of how the Computer determines the first and second positions:



ALEXIA
DYLAN
MICHAEL
SUSAN
TRAVIS

When the program begins, MICHAEL is compared with MICHAEL to see which precedes the other alphabetically. MICHAEL remains at the top. MICHAEL is compared with TRAVIS. MICHAEL still remains at the top.

Next MICHAEL is compared with DYLAN. Since DYLAN precedes MICHAEL, DYLAN now assumes MICHAEL's place at the top.

Now DYLAN is compared with ALEXIA. ALEXIA comes to the top. Finally ALEXIA is compared with SUSAN. ALEXIA remains at the top.

Now that all the names have been compared for the top position, the Computer repeats the cycle to determine the second, third, fourth, and fifth positions. ALEXIA becomes ZZ so that it will not assume other positions.

Now that you see what the program does, let's run through it using the same names we used above.

Lines 50 and 60 set the value of X. The first time through the program, X equals 1.

Then lines 90 through 110 compare A\$(1) – MICHAEL – with every other name in array A\$ until it reaches a word that precedes A\$(1). In our example, the third word – DYLAN – precedes it. Line 100 then makes A\$(X) equal to A\$(3) – DYLAN's place in the array. When DYLAN is compared with the fourth word – ALEXIA – A\$(X) becomes A\$(4).

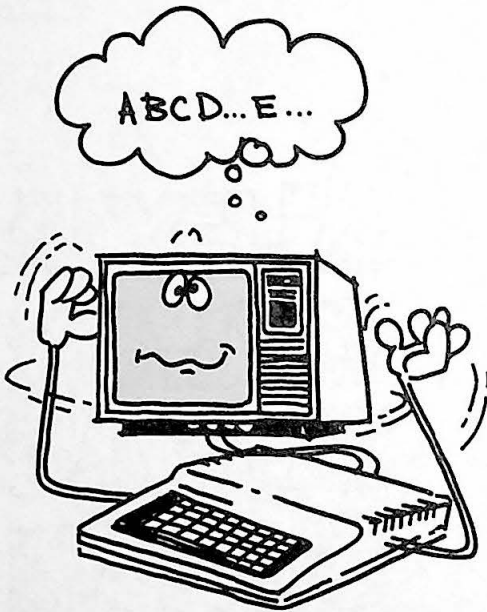
Line 120 PRINTs A\$(4) – ALEXIA – and line 130 makes A\$(4) equal to ZZ.

At this point, lines 50 and 60 make X equal 1 again. A\$(1) – MICHAEL – is again compared with other names in the array.

When MICHAEL's place in the array becomes ZZ, line 80 sends the Computer back up to line 60 which makes X equal to 2. A\$(2) – TRAVIS is then compared with all the names in the array.

When all places in the array contain ZZ, line 70 ends the program.

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



DO-IT-YOURSELF PROGRAM

Our answer is in the back of the book.

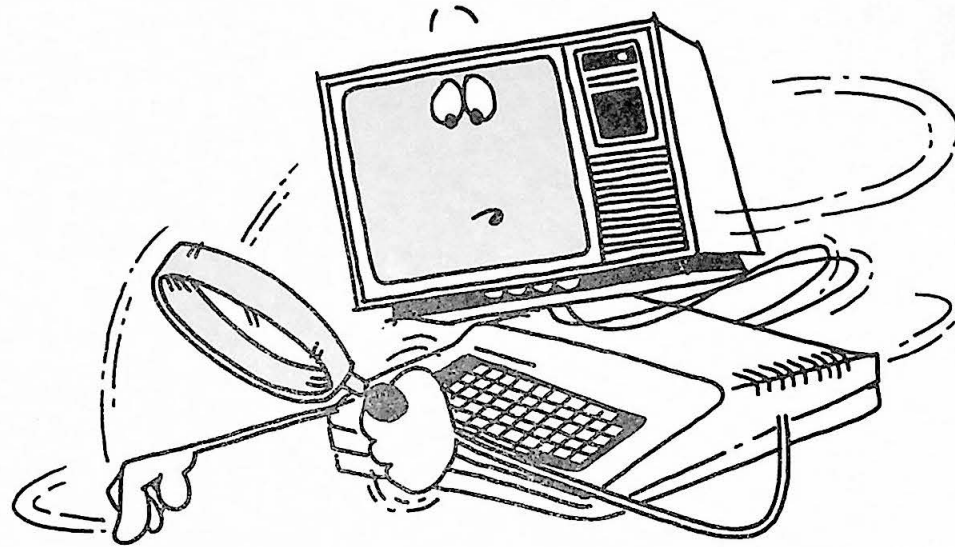
The method of sorting we've demonstrated is one of the easiest to program. There are other more complicated methods which will sort much faster. If you have a great number of items to sort, you may want to investigate one of the other sorting methods.

LEARNED IN CHAPTER 23

BASIC SYMBOLS

>
<
=

CHAPTER 24



**Getting Analytical
(for those with more than 4K RAM)**



GETTING ANALYTICAL

(for those with more than 4K RAM)

If you have more than 4K RAM, you have an easy way to analyze everything. By giving your information several labels, you'll be able to look at it all through several perspectives.

For an example, let's use the voting program from chapter 15. Here's the information:

ELECTION POLL		
District	Votes For Candidate 1	Votes For Candidate 2
1	143	678
2	215	514
3	125	430

We're only using 3 districts to keep it simple.

Were calling them candidates 1 and 2 this time rather than A and B.

By giving each item two labels we can put them all in one array. Here's what it will look like:

In Chapter 15, we created array A for candidate 1 and array B for candidate 2. Now, we're putting them in one array – V.

	ARRAY V	
	Candidate 1	Candidate 2
District 1	V(1,1) 143	V(1,2) 678
District 2	V(2,1) 215	V(2,2) 514
District 3	V(3,1) 125	V(3,2) 430

The first label tells which district the votes are from. The second tells which candidate they are for.

For example, we used V(1,2) to label the 678 votes. By doing that, we said that the 678 votes are in an array named V. They are from district 1 and are for candidate 2.

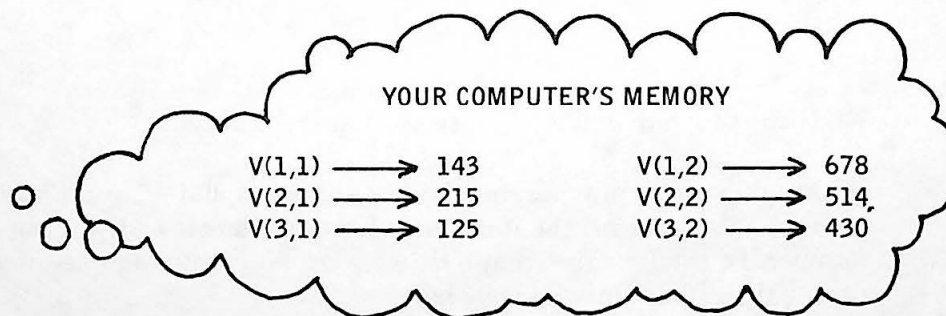
Let's put it all in a program. Type:

```
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 < 1 OR C > 2 THEN 90
110 PRINT V(D,C)
120 GOTO 70
```

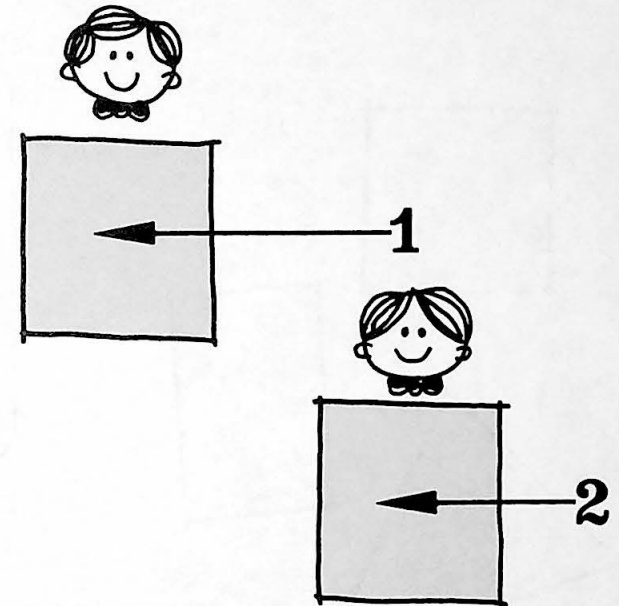
Line 5 reserves space in memory for an array named V. Each item may have two labels. The first label can be no higher than 3; the second, no higher than 2.

RUN the program. Make sure all the votes are labeled like they are above. Try different combinations of district and candidate numbers.

Lines 20 through 60 READ the votes for candidates 1 and 2 from districts 1, 2, and 3, label them, and put them in your Computer's memory:

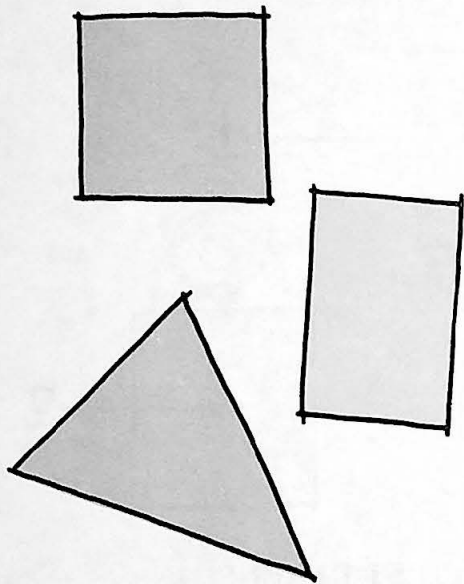


Now that you have two perspectives on the vote groups, you can take advantage of it. Delete lines 70-120 and type:



FIRST DIMENSION

Remember how to delete lines?
70 **ENTER**
Deletes line 70



```
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, don't feel bad. Skip it!

RUN the program and analyze to your heart's content.

What you've just programmed is a two-dimensional array. It's called two-dimensional because all the items in it have two labels – district and candidate number. In the previous chapters, we were programming one-dimensional arrays – their items only had one label.

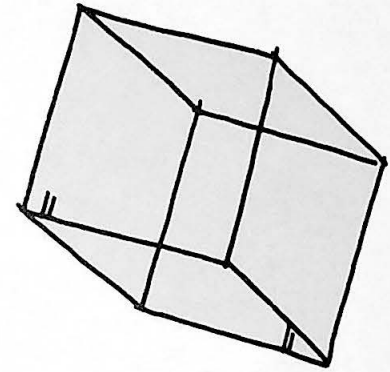
There's no reason to stop with two dimensions. Your Computer will let you program as many dimensions as you want in your array.

THE THIRD DIMENSION

We're now ready to add interest groups — a third dimension to the array. The election poll statistics above were all from interest group 1. We also polled some statistics from groups 2 and 3. Here's the information:

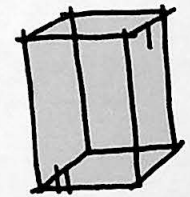
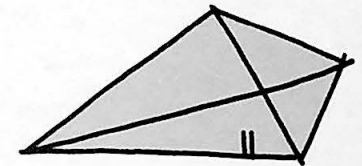
VOTES FROM INTEREST GROUP 1

	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

Here's how we'll label all these votes in array V:

ARRAY V

INTEREST GROUP 1

	Candidate 1	Candidate 2
District 1	V(1,1,1) 143	V(1,1,2) 678
District 2	V(1,2,1) 215	V(1,2,2) 514
District 3	V(1,3,1) 125	V(1,3,2) 430

INTEREST GROUP 2

	Candidate 1	Candidate 2
District 1	V(2,1,1) 525	V(2,1,2) 54

	V(2,2,1)	V(2,2,2)
District 2	318	157

	V(2,3,1)	V(2,3,2)
District 3	254	200

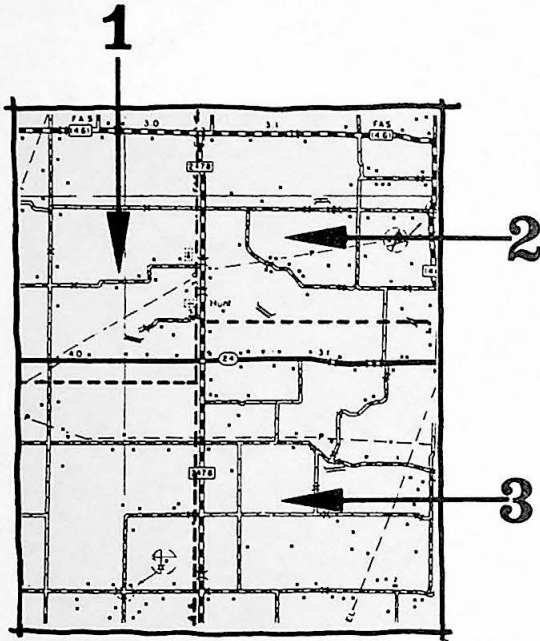
INTEREST GROUP 3

	Candidate 1	Candidate 2
	V(3,1,1)	V(3,1,2)
District 1	400	119

	V(3,2,1)	V(3,2,2)
District 2	124	300

	V(3,3,1)	V(3,3,2)
District 3	75	419

To get all this into your Computer's memory, erase your program and type:



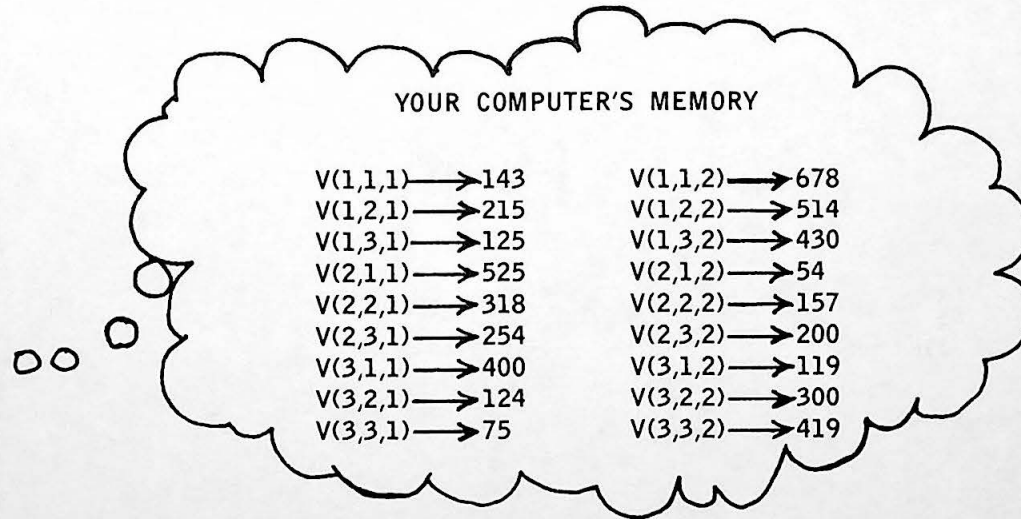
SECOND DIMENSION

```

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 "CANDIDATE NO.(1-2)"; C
160 IF C < 1 OR C > 2 THEN 150
170 PRINT V(G,D,C)
180 GOTO 110

```

RUN the program and test all the labels. Line 40 – 100 put this into your Computer's memory:



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

```

```

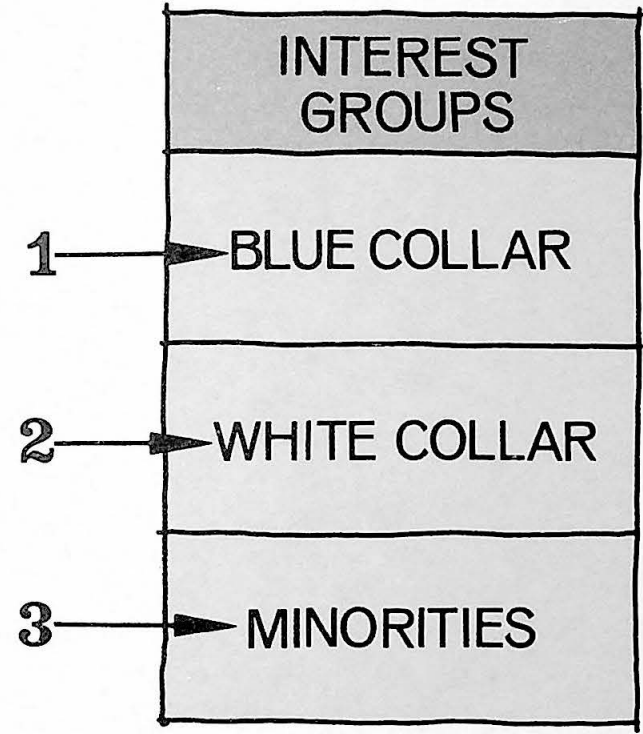
2000 INPUT "DISTRICT(1-3)"; D
2010 IF D<1 OR D>3 THEN 2000
2020 CLS
2030 PRINT @ 102, "VOTES FROM DIST." D
2040 PRINT @ 168, "CAND. 1"
2050 PRINT @ 176, "CAND. 2"
2060 FOR G=1 TO 3
2070 PRINT @ P, "GROUP" G
2080 FOR C=1 TO 2
2100 PRINT @ P+8*C,V(G,D,C);
2110 NEXT C
2120 P=P+32
2130 NEXT G
2140 RETURN

```

```

3000 INPUT "CANDIDATE(1-2)"; C
3010 IF C<1 OR C>2 THEN 3000
3020 CLS
3030 PRINT @ 102, "VOTES FOR CAND." C

```



THIRD DIMENSION

```

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

```

TWO DIMENSIONAL CARDS

Write a program to deal the cards using a two dimensional array. One dimension can be one of the 4 suits and the other dimension can be the card's value (1-13).

DO-IT-YOURSELF PROGRAM

Our answer is in the back of this book.

LEARNED IN CHAPTER 24

BASIC CONCEPT

Multi-dimensional arrays

NOTES:

SECTION IV

**DON'T BYTE OFF
MORE THAN YOU
CAN CHEW**

Pardon our pun, but this section is quite a bit (oops — pardon again) more technical than the rest of this book. It uses some terms and concepts we haven't explored yet, such as machine-language and direct memory addressing.

If you're a technical type, jump right in! But if you're not, be forewarned. You will have to be extra careful in typing in the sample programs. Then double- and triple-check them against our program listings before running them. If your program contains typing errors, it won't work, and you'll probably have to reset the Computer to regain control.

... Still with us? O.K. — now that we've warned you, we can tell you the good part. The results of your labors will be very impressive. Part I will demonstrate how to create high resolution (extremely detailed) graphics on your screen. Part II gives you the information you need to do things you can't do with BASIC — such as greatly intensifying the speed of graphics programs — by calling machine-language subroutines.

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 cannot be accessed 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 which demonstrate how to select a graphics mode and how to use it. The first runs on 4K or 16K RAM systems; the second, on 16K only. We've also included a general-purpose program which you can modify to select any of the graphics modes (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 of the graphics modes. 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 over carefully. Then run it.

After a few seconds, a block will appear 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
	East
	West
	Northwest
	Northeast
	Southwest
	Southeast
	Stops motion

Four-Color Set:

	Color 1
	Color 2
	Color 3
	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 US$ = "↑": DS$ = CHR$(10): WS$ = CHR$(8): ES$ =
CHR$(9)
220 NWS$ = "Q": NES$ = "W": SWS$ = "A": SES$ = "S"
230 C0$ = "0": C1$ = "1": C2$ = "2": C3$ = "3"
240 'CHECK FOR KEYBOARD CHARACTER
250 AS$ = INKEY$
260 IF AS$ = US$ THEN YI = -1: XI = 0: GOTO 400
270 IF AS$ = DS$ THEN YI = 1: XI = 0: GOTO 400
280 IF AS$ = WS$ THEN XI = -1: YI = 0: GOTO 400
290 IF AS$ = ES$ THEN XI = 1: YI = 0: GOTO 400
300 IF AS$ = NWS$ THEN XI = -1: YI = -1: GOTO 400
310 IF AS$ = NES$ THEN XI = 1: YI = -1: GOTO 400
320 IF AS$ = SWS$ THEN XI = -1: YI = 1: GOTO 400
330 IF AS$ = SES$ THEN XI = 1: YI = 1: GOTO 400
340 'CHANGE COLORS IF 0-3 WAS PRESSED
350 IF C0$ < = AS$ AND AS$ < = C3$ THEN CC = ASC(AS$)
- 48: GOTO 400
360 'CHANGE COLOR SET IF "/" WAS PRESSED
370 IF AS$ = "/" THEN CS = (NOT CS AND 8) OR (CS AND
NOT 8): POKE 65314,135 + CS: GOTO 400

```

```

380 IF AS = 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 XMOD4 = 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

```

Note for Extended BASIC Users: *The 64 x 64 mode is not available in Extended BASIC; however, this program will get it for you. But first, 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: 256 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 over 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 VIDEO RAM
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$(1)
110 'PH( ) AND VH( ) CONTAIN HI-RES. BIT PATTERN
120 'PA( ) AND VA( ) CONTAIN TEXT BIT PATTERN
130 'TWO( ) CONTAINS A LIST OF POWERS OF 2
140 DIM PH(6),PA(6),VH(2),VA(2),TWO(7)
150 FOR I = 0 TO 6: READ PH(I): NEXT
160 DATA 0,0,1,0,1,0,0
170 FOR I = 0 TO 6: READ PA(I): NEXT
180 DATA 0,1,0,0,0,0,0
190 FOR I = 0 TO 2: READ VH(I): NEXT
200 DATA 0,1,1
210 FOR I = 0 TO 2: READ VA(I): NEXT
220 DATA 0,0,0
230 READ CH 'HI-RES BIT MASK FOR VID.CTRL. REG.
240 DATA 240
250 READ CA 'TEXT BIT MASK FOR VID.CTRL. REG.
260 DATA 0
270 FOR I = 0 TO 7: READ TWO(I): NEXT
280 DATA 1,2,4,8,16,32,64,128
290 GOSUB 800 'CLEAR OUT VIDRAM
300 'MAIN PROGRAM
310 M$(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 AS$ = INKEY$: IF AS$ = "" 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 590
590 GOSUB 630 : GOTO 320
600 'END OF MAIN PROGRAM
610 'SUBRTNS TO SELECT G6R 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

These subroutines may be used to select any of the graphics modes (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 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

```

A FEW DEFINITIONS

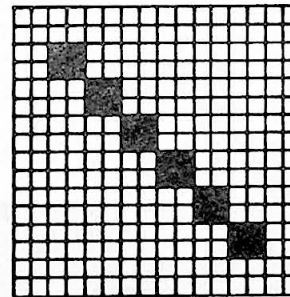
GRAPHICS

This refers to the ability to set or reset blocks or points called "pixels." For each pixel, you may choose from two, four or eight colors, depending on the particular mode selected. By setting various combinations of pixels, you can generate lines, geometric figures, pictures, etc.

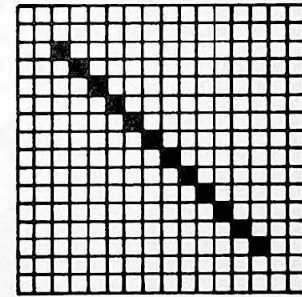
RESOLUTION

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 four times as fine as that of line A.



Line A.
Low Resolution



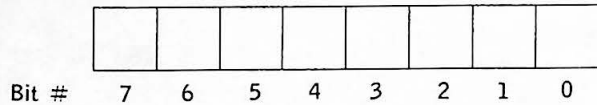
Line B.
High Resolution

RAM, BYTES AND BITS

RAM is divided up 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 programs and data are stored. The Computer also uses RAM for storage of internal values. RAM is erased when the Computer is turned off.

One byte consists of eight on/off switches called "bits." Here is one byte:



Notes:

- (1) In this discussion, we will 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." These terms will be used 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 decimal 0 to 255. (See a math or computer text for a discussion of binary numbering.)

PEEK AND POKE

These BASIC words allow you to examine (PEEK) or change (POKE) the contents of memory. Just for review, here is the syntax for each command (the syntax is the way that 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 like:

```
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 like 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		OR	0	1		NOT 0 = 1
0	0	0		0	0	1		NOT 1 = 0
1	0	1		1	1	1		

Here are some examples of Boolean operations on one-byte binary values:

```

      10101010
AND 11110000
-----
      10100000

      01101110
OR 10001000
-----
      11101110
```

```
NOT (10101010) = 01010101
```

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
```

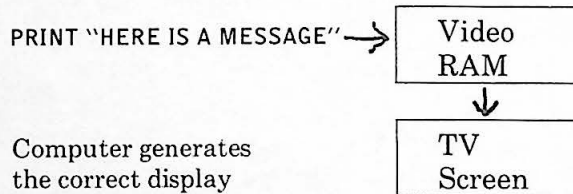
Since bit 7 is set in the value 128, bit 7 will always be set as a result of the operation. The other bits in the result will be the same as those in address #4000.

VIDEO RAM

When you output to the screen, the information is actually stored in a portion of memory. The video display circuitry reads from this "video RAM" in order to generate the screen display.

Text goes
into RAM

You type:



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.

The COLOR COMPUTER can be programmed 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 will probably be 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, and 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 try. There are several questions to ask yourself:

What is the video RAM requirement? Does your Computer have enough RAM to accommodate it? If it does, will there be 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 four colors available for each pixel. Further, you may select between two sets of four colors. G1R on the other hand, offers a 128 x 64 pixel density, with two colors available for each pixel. You may select between two sets of two 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 alphanumeric 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 — addresses above *memend* can be used 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 the 16K systems, 16383.

pagesize depends on which graphics mode you are going to use. For 4K systems, you will probably be limited to G1C or G1R; in either of these modes, *pagesize* = 1024. For 16K systems, you may use any of the modes, even those that use 6144 bytes.

For example, to use G1C in a 4K system, you should 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 out with a clean video screen. To do this, simply 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: Steps 4 and 5 should be performed consecutively, with no pauses in between steps. Otherwise, the screen will show what is often called "garbage."

4. SWITCH IN YOUR VIDEO RAM

Using the page select register, you 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 seven-bit value. (The eight 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* + *I* (see Step 3).

Table 3 doesn't list all possible addresses where you might want to start video RAM. The following procedure will let 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:

$$\text{OFFSET} = \text{VIDRAM} / 512$$

VIDRAM is the start address of your video RAM (usually *memend* + *I*).

For example, in 4K systems with your video RAM starting at 3072, $OFFSET = 3072 / 512 = 6$.

Then express OFFSET as a seven-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, you must give it to the page-select register.

Remember, this register consists of seven 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, we need to 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.

$$65478 + 1 * 2 + DT$$

is a shorthand way to poke the appropriate addresses in the page-select register. DT is the 0/1 value for each of the seven 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:

5-A. Set the VDG register

5-B. Set the control register.

(5-A.) First, look up the three-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 three 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, we need to 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
```

(5-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.*

When you have executed Steps 2 through 5, the Computer will be in the graphics mode you selected. The screen should be blank. The rest of your program can be devoted to using the graphics mode.

In Program #1, see line 100. In Program #2, see lines 740-770.

PUTTING GRAPHICS TO WORK

Once you've selected the graphics mode, you can control what appears on the screen by POKEing data into the graphics page you have selected. How the data is interpreted will depend on the mode you've selected. In some modes, one byte may control a sequence of eight bits; in others, one 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 the graphics modes, the screen is divided up 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 of these statements allow direct reference to (X,Y) coordinates. For example, to set the centerpoint on the screen to blue, we simply use:

```
SET(31,15,3)
```

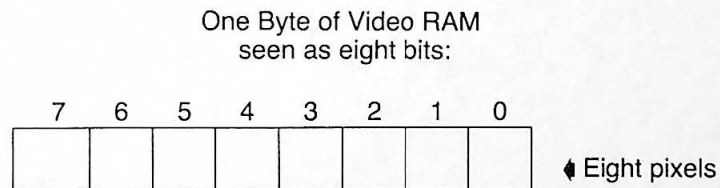
Using the higher-resolutions graphics modes is a little more difficult. We can't deal directly with (X,Y) coordinates; we must translate or "map" the desired (X,Y) coordinates onto the appropriate byte of video RAM. When one byte controls two or more pixels, we must map the (X,Y) coordinates onto the appropriate bit or bits within a byte.

Table 4 shows how each byte of video RAM controls one 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 eight pixels:



Bit 7 controls the leftmost pixel in the sequence; bit 0, the rightmost.

With this in mind, we can construct a series of BASIC operations to map (X,Y) onto one bit in one byte.

Notes: 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 we 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 one-byte value will set the pixel? What one-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 one-byte value will reset the pixel? What one-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 we have 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

NOTES:

- (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"; etc. In all of the "semigraphics" modes, you have eight colors at once. In all of the "with color" modes, you have four colors at once. In all of the "with resolution" modes, you have two colors at once.
- (2) In the four-color modes, you may select between two sets of four colors each. In the two-color modes, you may select between two sets of two 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

COLOR SET	DATA BIT 6	Color			Resolution		Data Byte(s)	Comments	
		Character Color	Background	Border	Columns x Rows	Detail			
0	0	Green	Black	Black	32 x 16	8 dots		 ASCII code	The <i>Alphanumeric Internal</i> mode uses an internal character generator which contains the following five dot by seven dot characters: @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [/] ← SP ! " # \$ % & * + , = / 0 1 2 3 4 5 6 7 8 9 ; : = ? .
1	1	Black	Green	Black		12 dots			
X	X	Orange	Black	Black	64 x 32			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. The color of illuminated parts is determined by three bits. It requires 512 bytes of display memory.	
		Black	Orange	Black					
0	X	Lx C2 C1 C0 Color	Black	64 x 48			The <i>Semigraphics-6</i> mode is similar to the <i>Semigraphics-4</i> mode with the following difference: The eight dot by twelve dot rectangle is divided into six equal parts. Color is determined by the two remaining bits. It requires 512 bytes of display memory.		
1		0 X X Black						0 0 Green	0 1 Green
X	X	0 X X Black	Black	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.		
X	X	Lx C2 C1 C0 Color	Black	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.		

Table 4. Detailed Description of the Graphics Modes (Continued)

COLOR SET	DATA BIT 6	Color			Resolution		Data Byte(s)	Comments																																																																																																																								
		Character Color	Background	Border	Columns x Rows	Detail																																																																																																																										
X	X	Lx C2 C1 C0 0 X X X 1 0 0 0 1 0 0 1 1 0 1 0 1 0 1 1 1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1	Color Black Green Yellow Blue Red Buff Cyan Magenta Orange	Black	64 x 192	<table border="1"> <tr><td>L₁</td><td>L₀</td></tr> <tr><td>L₃</td><td>L₂</td></tr> <tr><td>L₅</td><td>L₄</td></tr> <tr><td>L₇</td><td>L₆</td></tr> <tr><td>L₉</td><td>L₈</td></tr> <tr><td>L₁₁</td><td>L₁₀</td></tr> <tr><td>L₁₃</td><td>L₁₂</td></tr> <tr><td>L₁₅</td><td>L₁₄</td></tr> <tr><td>L₁₇</td><td>L₁₆</td></tr> <tr><td>L₁₉</td><td>L₁₈</td></tr> <tr><td>L₂₁</td><td>L₂₀</td></tr> <tr><td>L₂₃</td><td>L₂₂</td></tr> </table>	L ₁	L ₀	L ₃	L ₂	L ₅	L ₄	L ₇	L ₆	L ₉	L ₈	L ₁₁	L ₁₀	L ₁₃	L ₁₂	L ₁₅	L ₁₄	L ₁₇	L ₁₆	L ₁₉	L ₁₈	L ₂₁	L ₂₀	L ₂₃	L ₂₂	<table border="1"> <tr><td>1</td><td>C₂</td><td>C₁</td><td>C₀</td><td>L₁</td><td>L₀</td><td>X</td><td>X</td></tr> <tr><td>1</td><td>C₂</td><td>C₁</td><td>C₀</td><td>L₃</td><td>L₂</td><td>X</td><td>X</td></tr> <tr><td>1</td><td>C₂</td><td>C₁</td><td>C₀</td><td>L₅</td><td>L₄</td><td>X</td><td>X</td></tr> <tr><td>1</td><td>C₂</td><td>C₁</td><td>C₀</td><td>L₇</td><td>L₆</td><td>X</td><td>X</td></tr> <tr><td>1</td><td>C₂</td><td>C₁</td><td>C₀</td><td>L₉</td><td>L₈</td><td>X</td><td>X</td></tr> <tr><td>1</td><td>C₂</td><td>C₁</td><td>C₀</td><td>L₁₁</td><td>L₁₀</td><td>X</td><td>X</td></tr> <tr><td>1</td><td>C₂</td><td>C₁</td><td>C₀</td><td>X</td><td>X</td><td>L₁₃</td><td>L₁₂</td></tr> <tr><td>1</td><td>C₂</td><td>C₁</td><td>C₀</td><td>X</td><td>X</td><td>L₁₅</td><td>L₁₄</td></tr> <tr><td>1</td><td>C₂</td><td>C₁</td><td>C₀</td><td>X</td><td>X</td><td>L₁₇</td><td>L₁₆</td></tr> <tr><td>1</td><td>C₂</td><td>C₁</td><td>C₀</td><td>X</td><td>X</td><td>L₁₉</td><td>L₁₈</td></tr> <tr><td>1</td><td>C₂</td><td>C₁</td><td>C₀</td><td>X</td><td>X</td><td>L₂₁</td><td>L₂₀</td></tr> <tr><td>1</td><td>C₂</td><td>C₁</td><td>C₀</td><td>X</td><td>X</td><td>L₂₃</td><td>L₂₂</td></tr> </table>	1	C ₂	C ₁	C ₀	L ₁	L ₀	X	X	1	C ₂	C ₁	C ₀	L ₃	L ₂	X	X	1	C ₂	C ₁	C ₀	L ₅	L ₄	X	X	1	C ₂	C ₁	C ₀	L ₇	L ₆	X	X	1	C ₂	C ₁	C ₀	L ₉	L ₈	X	X	1	C ₂	C ₁	C ₀	L ₁₁	L ₁₀	X	X	1	C ₂	C ₁	C ₀	X	X	L ₁₃	L ₁₂	1	C ₂	C ₁	C ₀	X	X	L ₁₅	L ₁₄	1	C ₂	C ₁	C ₀	X	X	L ₁₇	L ₁₆	1	C ₂	C ₁	C ₀	X	X	L ₁₉	L ₁₈	1	C ₂	C ₁	C ₀	X	X	L ₂₁	L ₂₀	1	C ₂	C ₁	C ₀	X	X	L ₂₃	L ₂₂	The <i>Semigraphics-24</i> mode requires twelve column consecutive addresses, and produces a 2x12 block. It requires 6144 bytes of display memory.
L ₁	L ₀																																																																																																																															
L ₃	L ₂																																																																																																																															
L ₅	L ₄																																																																																																																															
L ₇	L ₆																																																																																																																															
L ₉	L ₈																																																																																																																															
L ₁₁	L ₁₀																																																																																																																															
L ₁₃	L ₁₂																																																																																																																															
L ₁₅	L ₁₄																																																																																																																															
L ₁₇	L ₁₆																																																																																																																															
L ₁₉	L ₁₈																																																																																																																															
L ₂₁	L ₂₀																																																																																																																															
L ₂₃	L ₂₂																																																																																																																															
1	C ₂	C ₁	C ₀	L ₁	L ₀	X	X																																																																																																																									
1	C ₂	C ₁	C ₀	L ₃	L ₂	X	X																																																																																																																									
1	C ₂	C ₁	C ₀	L ₅	L ₄	X	X																																																																																																																									
1	C ₂	C ₁	C ₀	L ₇	L ₆	X	X																																																																																																																									
1	C ₂	C ₁	C ₀	L ₉	L ₈	X	X																																																																																																																									
1	C ₂	C ₁	C ₀	L ₁₁	L ₁₀	X	X																																																																																																																									
1	C ₂	C ₁	C ₀	X	X	L ₁₃	L ₁₂																																																																																																																									
1	C ₂	C ₁	C ₀	X	X	L ₁₅	L ₁₄																																																																																																																									
1	C ₂	C ₁	C ₀	X	X	L ₁₇	L ₁₆																																																																																																																									
1	C ₂	C ₁	C ₀	X	X	L ₁₉	L ₁₈																																																																																																																									
1	C ₂	C ₁	C ₀	X	X	L ₂₁	L ₂₀																																																																																																																									
1	C ₂	C ₁	C ₀	X	X	L ₂₃	L ₂₂																																																																																																																									
0	X	C1 C0 0 0 0 1 1 0 1 1	Color Green Yellow Blue Red	Green	64 x 64	<table border="1"> <tr><td>E₃</td><td>E₂</td><td>E₁</td><td>E₀</td></tr> </table>	E ₃	E ₂	E ₁	E ₀	<table border="1"> <tr><td>C₁</td><td>C₀</td><td>C₁</td><td>C₀</td><td>C₁</td><td>C₀</td><td>C₁</td><td>C₀</td></tr> </table>	C ₁	C ₀	C ₁	C ₀	C ₁	C ₀	C ₁	C ₀	The <i>Graphics-1C</i> mode uses 1024 bytes of display RAM in which one pair of bits specifies one picture element.																																																																																																												
E ₃		E ₂	E ₁	E ₀																																																																																																																												
C ₁	C ₀	C ₁	C ₀	C ₁	C ₀	C ₁	C ₀																																																																																																																									
1	0 0 0 1 1 0 1 1	Buff Cyan Magenta Orange	Buff																																																																																																																													
0	X	Lx 0 1	Color Black Green	Green	128 x 64	<table border="1"> <tr><td>L₇</td><td>L₆</td><td>L₅</td><td>L₄</td><td>L₃</td><td>L₂</td><td>L₁</td><td>L₀</td></tr> </table>	L ₇	L ₆	L ₅	L ₄	L ₃	L ₂	L ₁	L ₀	<table border="1"> <tr><td>L₇</td><td>L₆</td><td>L₅</td><td>L₄</td><td>L₃</td><td>L₂</td><td>L₁</td><td>L₀</td></tr> </table>	L ₇	L ₆	L ₅	L ₄	L ₃	L ₂	L ₁	L ₀	The <i>Graphics-1R</i> mode uses 1024 bytes of display RAM in which one bit specifies one picture element.																																																																																																								
L ₇		L ₆	L ₅	L ₄			L ₃	L ₂	L ₁	L ₀																																																																																																																						
L ₇	L ₆	L ₅	L ₄	L ₃	L ₂	L ₁	L ₀																																																																																																																									
1	0 1	Black Buff	Buff																																																																																																																													
0	X	Same colors as Graphics one C		Green	128 x 64	<table border="1"> <tr><td>E₃</td><td>E₂</td><td>E₁</td><td>E₀</td></tr> </table>	E ₃	E ₂	E ₁	E ₀	<table border="1"> <tr><td>C₁</td><td>C₀</td><td>C₁</td><td>C₀</td><td>C₁</td><td>C₀</td><td>C₁</td><td>C₀</td></tr> </table>	C ₁	C ₀	C ₁	C ₀	C ₁	C ₀	C ₁	C ₀	The <i>Graphics-2C</i> mode uses 2048 bytes of display RAM in which one pair of bits specifies one picture element.																																																																																																												
E ₃		E ₂	E ₁	E ₀																																																																																																																												
C ₁	C ₀	C ₁	C ₀	C ₁	C ₀	C ₁	C ₀																																																																																																																									
1			Buff																																																																																																																													
0	X	Same colors as Graphics one R		Green	128 x 96	<table border="1"> <tr><td>L₇</td><td>L₆</td><td>L₅</td><td>L₄</td><td>L₃</td><td>L₂</td><td>L₁</td><td>L₀</td></tr> </table>	L ₇	L ₆	L ₅	L ₄	L ₃	L ₂	L ₁	L ₀	<table border="1"> <tr><td>L₇</td><td>L₆</td><td>L₅</td><td>L₄</td><td>L₃</td><td>L₂</td><td>L₁</td><td>L₀</td></tr> </table>	L ₇	L ₆	L ₅	L ₄	L ₃	L ₂	L ₁	L ₀	The <i>Graphics-2R</i> mode uses 1536 bytes of display RAM in which one bit specifies one picture element.																																																																																																								
L ₇		L ₆	L ₅	L ₄			L ₃	L ₂	L ₁	L ₀																																																																																																																						
L ₇	L ₆	L ₅	L ₄	L ₃	L ₂	L ₁	L ₀																																																																																																																									
1			Buff																																																																																																																													
0	X	Same colors as Graphics one C		Green	128 x 96	<table border="1"> <tr><td>E₃</td><td>E₂</td><td>E₁</td><td>E₀</td></tr> </table>	E ₃	E ₂	E ₁	E ₀	<table border="1"> <tr><td>C₁</td><td>C₀</td><td>C₁</td><td>C₀</td><td>C₁</td><td>C₀</td><td>C₁</td><td>C₀</td></tr> </table>	C ₁	C ₀	C ₁	C ₀	C ₁	C ₀	C ₁	C ₀	The <i>Graphics-3C</i> mode uses 3072 bytes of display RAM in which one pair of bytes specifies one picture element.																																																																																																												
E ₃		E ₂	E ₁	E ₀																																																																																																																												
C ₁	C ₀	C ₁	C ₀	C ₁	C ₀	C ₁	C ₀																																																																																																																									
1			Buff																																																																																																																													

Table 4. Detailed Description of the Graphics Modes (Continued)

COLOR SET	DATA BIT 6	Color			Resolution		Data Byte(s)	Comments																
		Character Color	Background	Border	Columns x Rows	Detail																		
0	X	Same colors as Graphics one R		Green	128 x 192	<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td>L₇</td><td>L₆</td><td>L₅</td><td>L₄</td><td>L₃</td><td>L₂</td><td>L₁</td><td>L₀</td> </tr> </table>	L ₇	L ₆	L ₅	L ₄	L ₃	L ₂	L ₁	L ₀	<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td>L₇</td><td>L₆</td><td>L₅</td><td>L₄</td><td>L₃</td><td>L₂</td><td>L₁</td><td>L₀</td> </tr> </table>	L ₇	L ₆	L ₅	L ₄	L ₃	L ₂	L ₁	L ₀	The <i>Graphics-3R</i> mode uses 3072 bytes of display RAM in which one bit specifies one picture element.
L ₇				L ₆			L ₅	L ₄	L ₃	L ₂	L ₁	L ₀												
L ₇	L ₆	L ₅	L ₄	L ₃	L ₂	L ₁	L ₀																	
1	Buff																							
0	X	Same colors as Graphics one C		Green	128 x 192	<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td>E₃</td><td>E₂</td><td>E₁</td><td>E₀</td> </tr> </table>	E ₃	E ₂	E ₁	E ₀	<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td>C₁</td><td>C₀</td><td>C₁</td><td>C₀</td><td>C₁</td><td>C₀</td><td>C₁</td><td>C₀</td> </tr> </table>	C ₁	C ₀	C ₁	C ₀	C ₁	C ₀	C ₁	C ₀	The <i>Graphics-6C</i> mode uses 6144 bytes of display RAM in which one pair of bits specifies one picture element.				
E ₃				E ₂			E ₁	E ₀																
C ₁	C ₀	C ₁	C ₀	C ₁	C ₀	C ₁	C ₀																	
1	Buff																							
0	X	Same colors as Graphics one R		Green	256 x 192	<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td>L₇</td><td>L₆</td><td>L₅</td><td>L₄</td><td>L₃</td><td>L₂</td><td>L₁</td><td>L₀</td> </tr> </table>	L ₇	L ₆	L ₅	L ₄	L ₃	L ₂	L ₁	L ₀	<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td>L₇</td><td>L₆</td><td>L₅</td><td>L₄</td><td>L₃</td><td>L₂</td><td>L₁</td><td>L₀</td> </tr> </table>	L ₇	L ₆	L ₅	L ₄	L ₃	L ₂	L ₁	L ₀	The <i>Graphics-6R</i> mode uses 6144 bytes of display RAM in which one bit specifies one picture element.
L ₇				L ₆			L ₅	L ₄	L ₃	L ₂	L ₁	L ₀												
L ₇	L ₆	L ₅	L ₄	L ₃	L ₂	L ₁	L ₀																	
1	Buff																							

*Column-consecutive addresses starting at HEX 0400 are 0400, 0420, 0440, 0460, etc.

PART B

USING MACHINE-LANGUAGE SUBROUTINES WITH COLOR BASIC

This part describes how to call a machine-language subroutine from a BASIC program, and lists 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 simply 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’ll 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 be presenting a BASIC program that performs all five 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 will be a simple one. 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 **Ⓛ**, release it, then press any key from **Ⓐ** to **Ⓩ** . 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 this ML subroutine, see the end of this section.

STEP 1. PROTECTING MEMORY

With the CLEAR statement, you can reserve a section of RAM for storage of 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

ML programs may be loaded from tape via CLOADM, or POKED into RAM. In our example, we’ll 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 Color Computer where it starts. You do this by POKEing the two-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 will call 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 two-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 will return the original dummy argument. After a call to GIVABF, the USR-reference in your BASIC program will return 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 **F** **M**

To get any of the codes 1 through 26, press **F**, 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	Source	Code	Comments
Object Code			
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 NEXT KEY
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	

Notes: "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

The 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 will be described in the following format:

NAME – *Entry address*
Operation Performed
Entry Condition
Exit Condition

Note: The subroutine NAME is only for reference. It is not recognized by the Color Computer. The entry address is given in hexadecimal form; you must use an indirect jump to this address. Entry and Exit Conditions are given for machine-language programs.

BLKIN = (A006)
Reads a Block from Cassette

Entry Conditions

Cassette must be on and in bit sync (see CSRDON). 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 55s 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

CHROUT outputs a character to the device specified by the contents of 6F (DEVNUM).

DEVNUM = -2 (printer)

DEVNUM = 0 (screen)

Entry Conditions

On entry, the character to be output is in A.

Exit Conditions

All registers except CC are preserved.

CSRDON = [A004]

Starts Cassette

CSRDON starts the cassette and gets into bit sync for reading.

Entry Conditions

None

Exit Conditions

FIRQ and IRO are masked. U and V 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.

PART C

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	USRJMP – Jump to BASIC's USR routine
277-281	115-119	Can be used by machine language programs
282	11A	Keyboard Alpha lock – 0 = 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	4000-7FFF	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

APPENDIXES

MUSICAL TONES

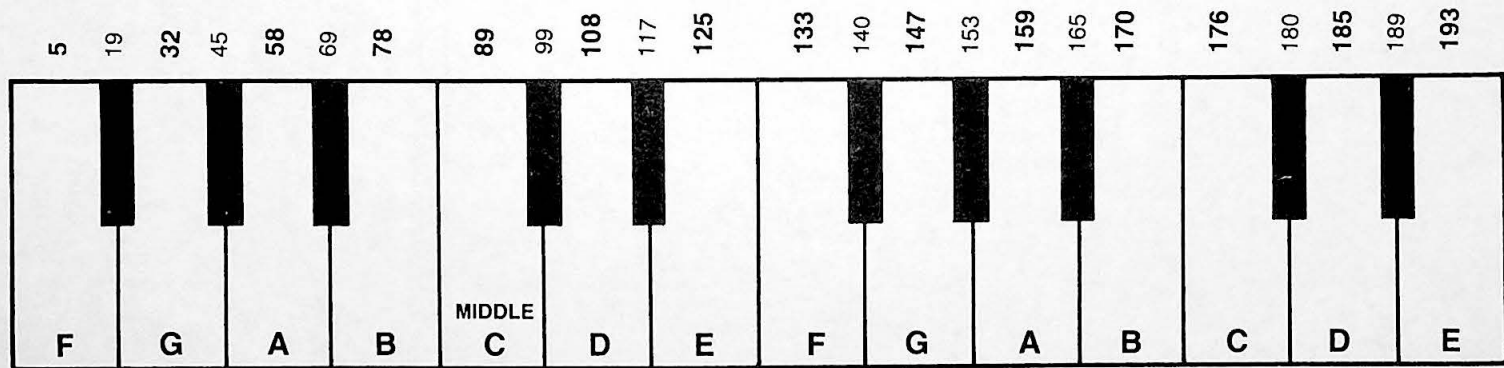
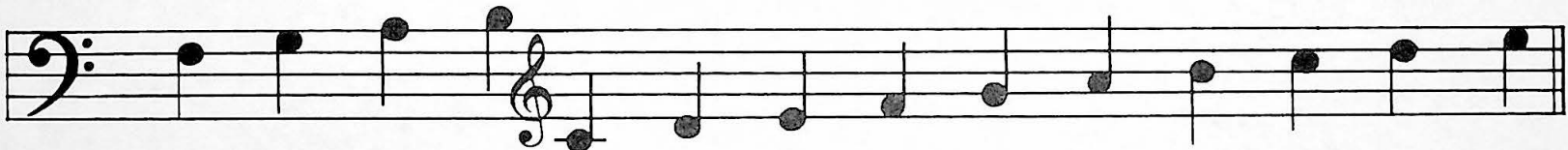
Your Computer can come fairly close to matching (although it can't *exactly* match) the musical tones shown below. You may either use the piano keyboard or the musical staff to produce electronic music.

If you're using the piano keyboard, the Computer tone for each key is directly over the key. For example, the Computer tone number for Middle C is 89.

If you're using the musical staff, the tone number for each note is below the note. For example the tone number for:



is 108.



If the note is a flat, select the tone number immediately preceding the note.
 For example:



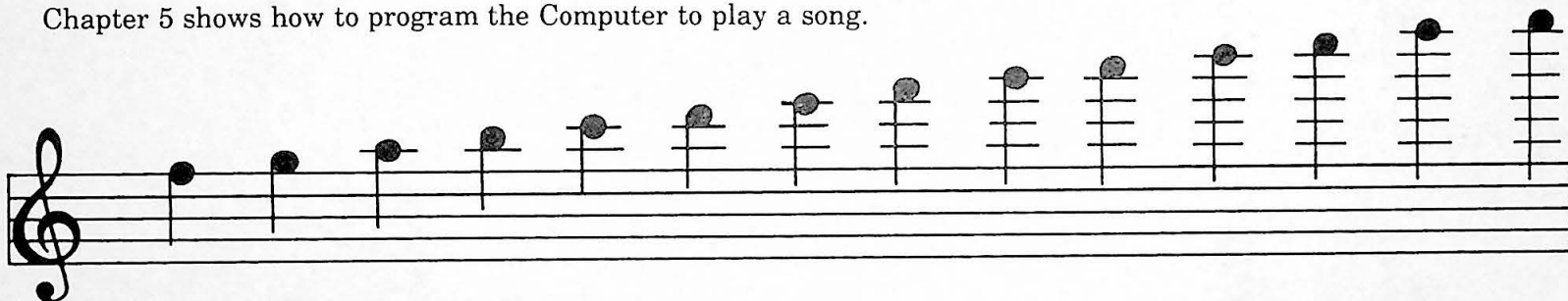
is 99.

If the note is a sharp, select the tone number immediately following the note.
 For example:



is 117.

Chapter 5 shows how to program the Computer to play a song.



197	200	204	207	210	213	216	218	221	223	225	227	229	231	232	234	236	237	238	239	241	242	243	244
	■		■		■			■		■			■		■		■			■		■	
F	G	A	B	C	D	E	F	G	A	B	C	D	E										

BASIC COLORS AND GRAPHICS CHARACTERS

These are the codes for the colors you can create on your screen. Chapters 1 and 9 show how to create them.

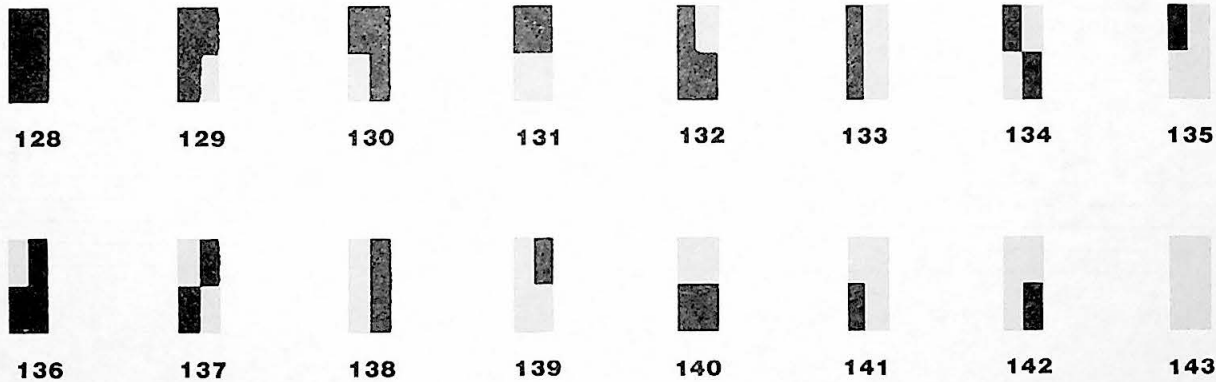
BASIC COLORS

- | | | |
|------------------------------|----------|-------------|
| 0 – black (absence of color) | 3 – blue | 6 – cyan |
| 1 – green | 4 – red | 7 – magenta |
| 2 – yellow | 5 – buff | 8 – orange |

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.

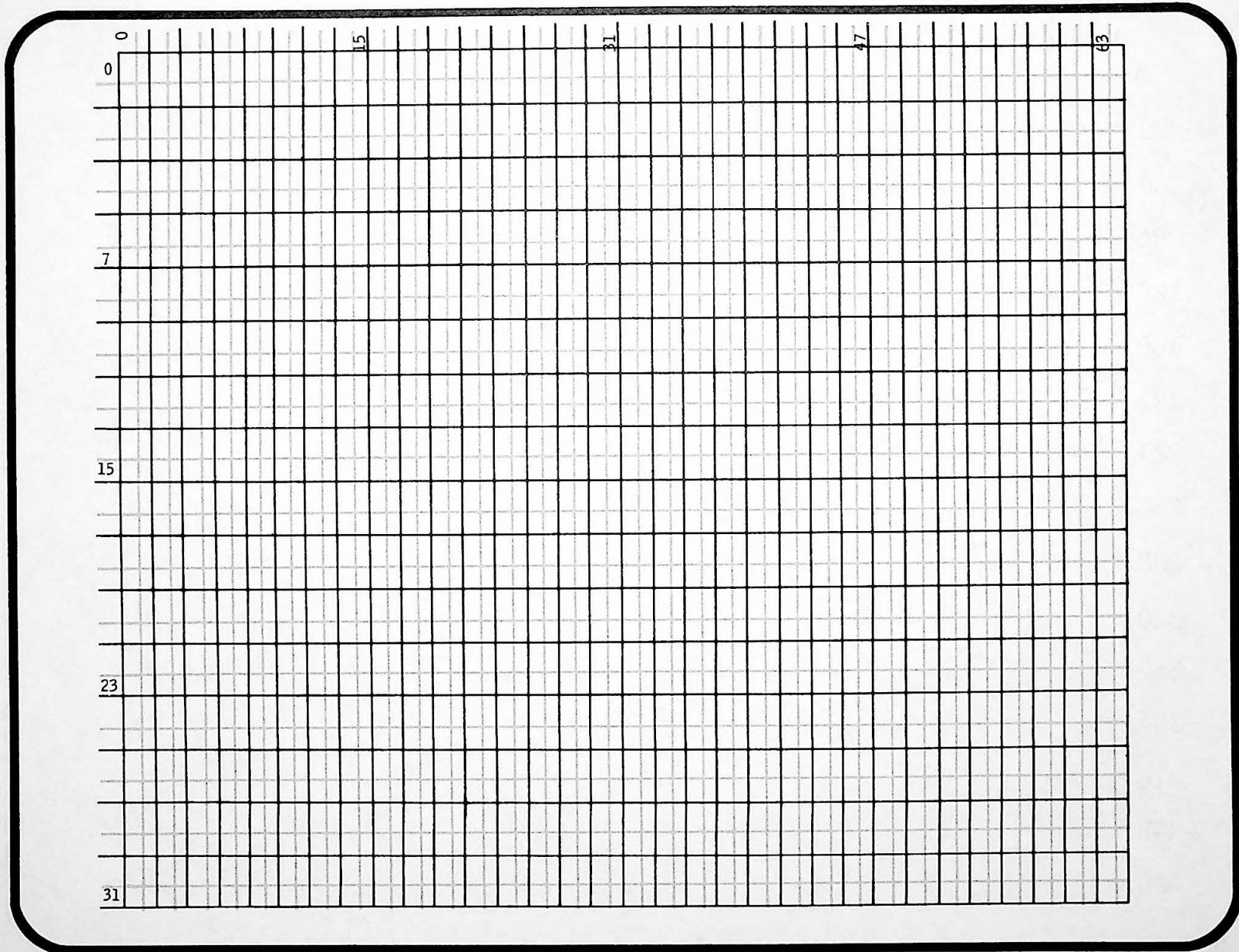
- | | | |
|---------------|-------------|----------------|
| + 16 – yellow | + 64 – buff | + 96 – magenta |
| + 32 – blue | + 80 – cyan | + 112 – orange |
| + 48 – red | | |

Chapter 18 explains how to use graphics characters.

PRINT @ SCREEN LOCATIONS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0																																
32																																
64																																
96																																
128																																
160																																
192																																
224																																
256																																
288																																
320																																
352																																
384																																
416																																
448																																
480																																

GRAPHICS SCREEN LOCATIONS



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).

Chapter 15 shows how to use these codes with CHR\$ to produce a character.

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
SPACE BAR	32	20
!	33	21
"	34	22
#	35	23
\$	36	24
%	37	25
&	38	26
'	39	27
(40	28
)	41	29
*	42	2A
+	43	2B
,	44	2C
-	45	2D
.	46	2E
/	47	2F
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

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
;	59	3B
<	60	3C
=	61	3D
>	62	3E
?	63	3F
@	64	40
A	65	41
B	66	42
C	67	43
D	68	44
E	69	45
F	70	46
G	71	47
H	72	48
I	73	49
J	74	4A
K	75	4B
L	76	4C
M	77	4D
N	78	4E
O	79	4F
P	80	50
Q	81	51
R	82	52
S	83	53
T	84	54
U	85	55

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
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 code 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).

LOWER-CASE CODES

These are the ASCII codes for lower-case letters. You can produce these characters by pressing the **SHIFT** and **0** keys simultaneously to get into an upper/lower case mode. The lower case 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
m	109	6D

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
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

ANSWERS TO EXERCISES

CHAPTER 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
```

CHAPTER 5

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
```

Program which Shows 9 colors for 1 second each:

```
10 FOR C = 0 TO 8
20 CLS(C)
30 FOR X = 1 TO 460
40 NEXT X
50 NEXT C
```

CHAPTER 7

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"
```

Russian Roulette program

```
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"

```

CHAPTER 10

Test Your Arithmetic Program

```

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

```

CHAPTER 11

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 RASING 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

```

CHAPTER 12

Editing a Sentence

```

10 PRINT "TYPE A SENTENCE : "
15 INPUT S$
20 PRINT "TYPE A PHRASE TO DELETE"
23 INPUT D$
25 L = LEN(D$)
30 PRINT "TYPE A REPLACEMNT PHRASE"
35 INPUT R$
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$) - E)
120 PRINT "NEW SENTENCE IS :"
130 PRINT NS$

```

CHAPTER 13

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"

```

CHAPTER 15

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

```

CHAPTER 21

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 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

```

CHAPTER 23

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 "1", #-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

```

CHAPTER 25

Deal two-dimensional card deck:

```
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
```

SUBROUTINES

These subroutines will let you run programs which 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: Accuracy of the subroutines is less than the accuracy of the COLOR BASIC's math operators and functions. This is due to two factors: 1. The subroutines contain many chain calculations, which tend to magnify the small error of individual operations. 2. These subroutines are only approximations of the functions they replace. In general, the subroutines are accurate to five or six 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: SQR(X), \sqrt{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=***
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: 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: 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 3061
0
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*.464286E-2
30620 W=Y+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: 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
```

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 1000
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: U = INT(YT/2)*2
460 SET(H,U,C): SET(H+1,U,C)
462 SET(H,U+1,C): SET(H+1,U+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 D$(6)
8 DIM DF(21), P(6), D$(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$(13), B(200): Y=1
10 FOR X = 1 TO 25: READ A(X): NEXT X
20 DATA 89, 99, 108, 117, 125
30 DATA 133, 140, 147, 153, 159
40 DATA 165, 170, 176, 180, 185
50 DATA 189, 193, 197, 200, 204
60 DATA 207, 210, 213, 216, 218
70 FOR X = 1 TO 13: READ S$(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 TU
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 "0", #-1, "LIST"
5070 FOR X = 1 TO Y-1
5080 PRINT #-1, S$(X)
5090 NEXT X
5100 CLOSE #-1: RETURN
5900 REM
6000 REM LOAD ITEMS FROM TAPE
6010 CLS: PRINT @ 136, "LOAD ITEMS FROM TAPE"
6020 PRINT @ 235, "REWIND TAPE"
6030 PRINT @ 300, "PRESS PLAY"
6040 PRINT @ 388, "PRESS <ENTER> WHEN READY"
6050 INPUT R#
6060 OPEN "1", #-1, "LIST"
6070 Y = 1
6080 IF EOF(-1) THEN 6120
6090 INPUT #-1, S$(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 "BUDJETED": 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$(X);
360 PRINT @ 97, "BUDJETED";
370 PRINT @ 257, "ACTUAL";
380 PRINT @ 440, CHR$(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 60
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(5)
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 = 108 : 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

```

ERROR MESSAGES

- /0** Division by zero. The Computer was asked to divide a number by 0, which is impossible.
- AO** Attempt to open a data file which 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 which dimensions array A for 12 or more elements, you will get this error.
- CN** Can't continue. If you use the command CONT and you are at the END of the program, you will get this error.
- DD** Attempt to redimension an array. An array can only be dimensioned once. For example, you cannot have DIM A(12) and DIM A(50) in the same program.
- DN** Device number error. Only three devices may be used with OPEN, CLOSE, PRINT, or INPUT: 0, - 1, or - 2. If you use another number you will get this error.
- DS** Direct statement. There is a direct statement in the data file. This could be caused if you load a program with no line numbers.
- FC** Illegal Function Call. This happens when you use a parameter (number) with a BASIC word that is out of range. For example SOUND (260,260) or CLS(10) will cause this error. Also RIGHT\$(S\$,20), when there are only 10 characters in S\$, would cause it. 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. This error occurs when you PRINT data to a file, or INPUT data from the file, using the wrong type of variable for the corresponding data. For example, INPUT # - 1, A, when the data in the file is a string, causes this error.
- FM** Bad file mode. This error occurs when you attempt to INPUT data from a file OPEN for OUTPUT (O), or PRINT data into a file OPEN for INPUT (I). 0
- ID** Illegal direct statement. You can only use INPUT as a line in the program, not as 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 only be 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 mis-spelled command, incorrect punctuation, open parenthesis, or an illegal character. Type the program line or command over.
 - 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 an nonexistent line number.

BASIC SUMMARY

WORD	PURPOSE	EXAMPLES	PAGES DISCUSSED
ABS	Computes the absolute value of a number.	PRINT ABS(-5)	141
ASC	Converts the first character in a string to its ASCII code. For example, ASC ("CAT") converts "C" to its ASCII code, 67. A listing of ASCII codes is in <i>Appendix E</i> .	PRINT ASC("B") X = ASC(T\$)	149
AUDIO	Connects or disconnects the sound coming from your tape recorder to your T.V. speaker.	AUDIO ON AUDIO OFF	159-64
CHR\$	Converts an ASCII code to the character it represents. A listing of the codes is in <i>Appendix E</i> . The graphics codes are listed in <i>Appendix B</i> .	PRINT CHR\$(143) PRINT CHR\$(67) Y\$ = CHR\$(32)	149, 154-5, 175-194
CLEAR	Reserves space in the Computer's memory for working with strings. (Without CLEAR, the Computer reserves 200 characters). If you are loading a machine language program, you can use a second number to specify the highest address BASIC can use. CLEAR always sets all numeric variables to zero and string variables to null (nothing).	CLEAR CLEAR 500 CLEAR 100, 14000	97-100
CLOAD	Loads the first program from cassette tape. You may specify the name of the program.	CLOAD CLOAD "PROGRAM"	72-75
CLOADM	Loads a machine-language program from cassette tape. You can specify an offset address for the Computer to add to the program's loading address.	CLOADM "PROG" CLOADM CLOADM "PROG", 1000	267

CLOSE	Closes a file by closing communication to the device you specify. See OPEN for a listing of devices.	CLOSE # - 1 CLOSE # - 2	220-30
CLS	Clears the screen to green, or to the color code you specify. See <i>Appendix B</i> for a list of the color codes.	CLS CLS(2)	12, 13, 47, 221
CONT	Continues executing a program after pressing BREAK or using STOP.	CONT	136
CSAVE	Saves a program on cassette tape. You may use a program name with up to 8 letters.	CSAVE CSAVE "PROGRAM"	71-75
DATA	Stores data in your program. Use READ to assign this data to variables.	DATA 5, 3, PEARS DATA PAPER, PEN	94-100, 127
DIM	Reserves space in memory for the arrays you specify.	DIM R(65), W(40) DIM W\$(8,25)	198-9, 201, 203, 241-8
END	Ends your program.	END	55-58
EOF	Checks to see if you've reached the end of the data in a file. If you have, EOF(-1) will be true; if you haven't, EOF(0) will be true.	IF EOF(-1) THEN CLOSE IF EOF(0) THEN INPUT # - 1	222
EXEC	Transfers control to a machine-language program at the address you specify. If you don't specify an address, the Computer will use the address specified at the last CLOADM command.	EXEC EXEC 32453	—

FOR . . . TO STEP/ NEXT	Creates a loop in your program which the Computer must repeat from the first number to the last number you specify. You may use STEP to specify how much to increment the number each time through the loop. If you omit STEP, 1 is the increment.	FOR X = 2 TO 5/NEXT X FOR A = 1 TO 10 STEP 5/ NEXT A FOR M = 30 TO 15 STEP - 5/ NEXT M	35-36, 64, 117-8
GOSUB	Sends the Computer to the subroutine beginning at the line number you specify.	GOSUB 500 GOSUB 5000	103-10
GOTO	Sends the Computer to the line number you specify.	GOTO 300	28-31, 58
IF/THEN . . . ELSE	Tests the relationship. If it is true, the Computer executes the instruction following THEN. If it is not true the Computer executes the instruction following ELSE or, if ELSE is omitted, the next line in the program.	IF A = 5 THEN 300 IF B\$ = "YES" THEN PRINT "XYZ" IF A = 3 THEN PRINT "CORRECT" ELSE PRINT "WRONG"	32, 55-8, 63, 138
INKEY\$	Strobes the keyboard and returns the key or non-key being pressed.	A\$ = INKEY\$	125-31
INPUT	Causes the Computer to stop and await input from the device you specify. If you do not specify a device number, the Computer will await input from the keyboard. See OPEN for device numbers.	INPUT X\$ INPUT "NAME"; N\$ INPUT R - 1, A, B\$	26, 27, 63, 104-5, 113, 220-30
INT	Converts a number to an integer.	X = INT(5.2)	96-100
JOYSTK	Returns the horizontal or vertical coordinate of the left or right joystick: 0 — horizontal, right joystick 1 — vertical, right joystick 2 — horizontal, left joystick 3 — vertical, left joystick	M = JOYSTK(0) H = JOYSTK(2)	84-9, 169-71
LEFT\$	Returns the left portion of a string. You specify the string and the length of the left portion.	P\$ = LEFT\$(M\$,7)	115-22

LEN	Returns the length of a string.	X = LEN(M\$)	113-22
LIST	Lists the entire program, or the lines in the program you specify.	LIST LIST 50-85 LIST 30 LIST -30 LIST 30-	26, 64
LLIST	Lists the entire program, or the lines you specify, on the printer.	LLIST LLIST 20-50	215-6
MEM	Tells you how much space the Computer has remaining in memory.	MEM	136
MID\$	Returns a substring within a string. You specify the string, the position which begins the substring, and the length of the substring. For example, MID\$("HOMES";2, 3) returns OME.	PRINT MID\$("WORDS;"2,3)	116-22
MOTOR	Turns the cassette ON or OFF.	MOTOR ON MOTOR OFF	159-64
NEW	Erases everything in memory.	NEW	25
ON ... GOSUB	Sends the Computer to one of the subroutines you specify.	ON Y GOSUB 50, 100	137
ON ... GOTO	Sends the Computer to one of line numbers you specify.	ON X GOTO 190, 200	138
OPEN	Opens communication to a device for Inputting (I) or Outputting (O) data. The devices are: #0 — screen of keyboard # - 1 — cassette recorder # - 2 — printer (It is not necessary to Open communication when you are INPUTting from the keyboard or PRINTing on the screen). You may specify an 8-character name of the data file	OPEN "I", # - 1, "FILE" OPEN "O", # - 1, "DATA"	219-20, 228

PEEK	Returns the contents in the memory location you specify.	A = PEEK(32076)	88, 257
POINT	Tells whether a dot at the horizontal and vertical location you specify is lit up. It will return a -1 if the dot is in the character mode, 0 if it is off, or a color code if it is on. See <i>Appendix B</i> for the color codes.	IF POINT(15,12) = 3 THEN PRINT "BLUE"	167-72
POKE	Puts a value in the memory location you specify. The value may be a number between 0 and 255.	POKE 15872,255	257
PRINT	Prints the message you specify on the device you specify. If you do not specify a device, your message will be printed on the video screen. See OPEN for device numbers.	PRINT "HI" PRINT A\$ PRINT #-1, A\$ PRINT #-2, "HI"	9-11, 104, 209-30
PRINT@	Prints your message at the screen position you specify. See <i>Appendix C</i> for the screen positions.	PRINT "HI", 256 PRINT A\$, 331	64, 164-94
READ	Reads the next item in the DATA line and assigns it to the variable you specify.	READ A\$ READ C, B	94-100, 127
REM	Allows you to insert a comment in your program. The Computer ignores everything on a REM line.	REM THIS IS IGNORED	105
RESET	Erases the dot SET on the screen location you specify. See <i>Appendix D</i> for the screen locations.	RESET(14,15)	83-4, 149-64
RESTORE	Sets the Computer's pointer back to the first item on the DATA lines.	RESTORE	97-100
RETURN	Returns the Computer from the subroutine to the BASIC word following GOSUB.	RETURN	103

RIGHT\$	Returns the right portion of the string you specify beginning at the position you specify.	ZP\$ = RIGHT\$(AD\$,5) PRINT RIGHT\$("ONE",2)	115-22
RND	Returns a random integer between 1 and the number you specify.	A = RND(10)	61-8, 91-3
RUN	Executes a program.	RUN	25
SET	Sets a dot at the screen location you specify, using the color you specify. See <i>Appendix D</i> for the screen locations and <i>Appendix B</i> for the color codes.	SET(14,13,3)	77, 78-88, 149-55
SGN	Tells the sign of a number. Returns a 1 if the number is positive, 0 if it is zero, or -1 if it is negative.	PRINT SGN(-4) X = SGN(A*B)	141
SIN	Returns the sine in radians	Y = SIN(5)	—
SKIPF	Skips to the end of the next program on cassette tape, or to the end of the program you specify.	SKIPF SKIPF "PROGRAM"	74
SOUND	Sounds the tone you specify for the duration you specify. Both the tone and the duration may be a number between 1 and 255.	SOUND 128, 3	13, 30, 39-41, 50-2, 126-8, 163-4
STOP	Makes the Computer stop executing the program.	STOP	135
STR\$	Converts a number to a string.	X\$ = STR\$(5) X = STR\$(Y)	141
TAB	TABs to the position you specify.	PRINT TAB(2)"HI" PRINT #-2, TAB(5) "HI"	—
USR	Calls a machine-language subroutine whose address is stored at 275-276.	X = USR(Y)	267
VAL	Converts a string to a number.	A = VAL(B\$)	129-30

KEYBOARD CHARACTERS

CHARACTER	PURPOSE	PAGES DISCUSSED
␣	Backspaces the cursor (the blinking light)	8
ENTER	Tells Computer you've reached the end of your program line or command line.	7, 25
BREAK	Stops execution of your program.	28-33
SHIFT @	Pauses execution of your program. Press any key to continue.	61
SHIFT ①	Switches Computer to and from upper/lower case mode.	14, 216

BASIC SYMBOLS

SYMBOL	EXPLANATION	PAGES DISCUSSED
" "	Indicates that the data in quotes is a constant.	8, 9, 125
:	Separates program "statements" on the same line.	104
()	Tells the Computer to perform the operation in the inside parenthesis first.	107, 108
;	Causes constants and variables to be PRINTED right next to each other.	29, 85, 177

BASIC OPERATORS

OPERATOR	PURPOSE	PAGES DISCUSSED
+	Combines strings	113-4
+	Addition	8-11
-	Subtraction	8-11
*	Multiplication	8-11
/	Division	8, 11
=	Equals	84, 87, 125, 233-8
>	Greater Than	84, 87, 125, 233-8
> = or = >	Greater than or equal to	84, 87, 125, 233-8
< = or = <	Less than or equal to	84, 87, 125, 233-8
<	Less than	84, 87, 125, 233-8
<> or > <	Not equal to	84, 87, 125, 233-8
AND	Logical AND	139-40, 257
OR	Logical OR	139-40, 257
NOT	Logical NOT	257

Index

Subject	Page
ABS	141
Alphabetizing	*233-8
AND	139-40, 257
Animation	149-94
Array	*199-207, 239-48
rules	206
string	209-16
ASC	Appendix, 149
AUDIO	159-64
BASIC	Appendix, 3
Bits	257
Boolean algebra	257
BREAK	28-33
Bytes	257
Cassette	See "Recorder"
Changing words	26, 28-9
Character codes	Appendix, 149
CHR\$	See "Character codes," Appendix, 149, 154-5, 175-194
CLEAR	*97-100, 212
CLOAD	*72-75
CLOADM	268
CLOSE	*220-30
CLS	12, 13, *47, 221
Colon (:)	104
Color	7, 12, 14, 77-89, 177
Comma (,)	29
CONT	136
Counting (time)	35-41, 43-9
CSAVE	*71-75
DATA	*94-100, 127
Delete	28, 118
DIM	*198-9, 201, 203, 241-8
Dividing	10
Dollar sign (\$)	*18-20
ELSE	138
Empty string	*125-131
END	55-58
ENTER	7
EOF	222
Erase	28, 118
Error messages	11, 12, 19, 21, 30, 95, 104, 114, Appendix
Exponential notation	142
Files	220-30
Filing	See "Alphabetizing"
FOR/NEXT	35-36, 64, 117-8

Subject	Page
Games	167-194
GOSUB	*103-10
GOTO	*28-31, 58
Graphics	77-89, 149-93
High resolution	252
Graphics characters	Appendix, 175-93
Greater than (>)	
Numeric	84, 87
String	233
Grid	Appendix, 65, 78, 152
High resolution graphics	252
IF/THEN	32, *55-8, 63
INKEY\$	*125-31
INPUT	26, 27, 63, 104-5, 113
INPUT # - 1	220-30
INT	*96-100
Joysticks	*84-9, 169-71
JOYSTK	See "Joysticks"
Labels	199-206
LEFT\$	115-22, 181
LEN	113-22
Less than (<)	
Numeric	84
String	233
LIST	26, 64
LLIST	215-6
Load from tape	See "CLOAD"
Loop	*28, 45-52
Lower Case Mode	181
Machine-Language	267
Sub-routines	
Math	103-10, *106
MEM	136
Memory	17, 136, 203
Microphone	159-64
MID\$	116-22
Motion	See "Animation"
Motor	159-66
Multiplying	10
Nested Loop	45-52
NEW	25
NOT	257
Not equal to (<>, ><)	125
Numeric data	*9, 20
OK	7
ON ... GOSUB	137
ON ... GOTO	138

Index

Subject	Page
OPEN	219-20, 228
Operations	106
OR	140, 257
Output	219-21, 228
Parentheses ()	107, *108
Pause	40
PEEK	*88, 257
Plus sign (+)	113-22
POKE	257
POINT	167-72
PRINT	9-11, 104, 209-30
PRINT # - 1	220-30
PRINT # - 2	216
PRINT @	Appendix, *64, 164-94
Printer	215-16
Print punctuation	29, 177
Program	3, 25
Prompt	7
Quotation marks (" ")	8, *9, 125
READ	*94-100, 127
Recorder	71-5, 159-64, 219-30
Relational Operators	See "Alphabetizing" 84, 87, 125
REM (Remark)	105
RESET	83-4, 149-64
RESTORE	*97-100

Subject	Page
RETURN	103
Reverse colors	7, 14
RIGHT\$	115-22, 181
RND	*61-8, 91-3
RUN	25
Save on tape	See "CSAVE"
Semi-colon (;)	29, 85, 177
SET	77, *78-88, 149-55
SGN	141
SHIFT @	61
SKIPF	74
SOUND	13, 30, 39-41, 50-2, 126-8, 163-4
STEP	38-40
STOP	135
Strings	*9, 19, 113-22
STR\$	141
Subroutines	*103-10
Machine-language	267
Tape	See "Recorder"
Timer	44-52, 128
USR	268
VAL	129-30
Variables	18, 22, 197-206
"Labeled"	199-206
Word processing	213-15

RADIO SHACK, A DIVISION OF TANDY CORPORATION

U.S.A.
FORT WORTH, TEXAS 76102

CANADA
BARRIE, ONTARIO, L4M4W5

TANDY CORPORATION

AUSTRALIA
*91 KURRAJONG ROAD
MOUNT DRUITT, N.S.W. 2770*

BELGIUM
*PARC INDUSTRIEL NANINNE
5140 NANINNE*

UNITED KINGDOM
*BILSTON ROAD, WEDNESBURY
WEST MIDLANDS WS10 7JN*