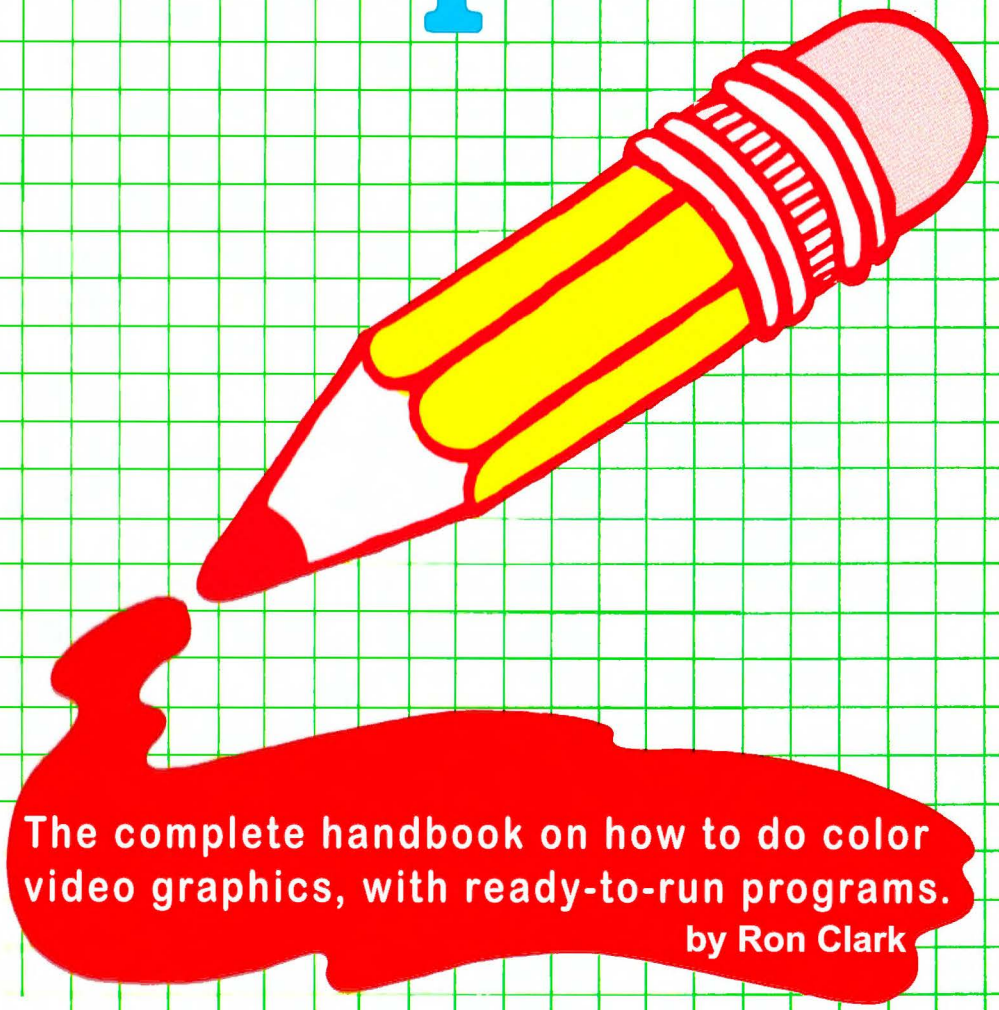
Color Computer Graphics



The complete handbook on how to do color video graphics, with ready-to-run programs.

by Ron Clark

Color Computer Graphics

Color Computer Graphics

by Ron Clark

ARCsoft Publishers
WOODSBORO, MARYLAND

ARCsoft Books by Ron Clark

I	101 Color Computer Programming Tips & Tricks
II	55 Color Computer Programs for Home, School & Office
III	55 MORE Color Computer Programs for Home, School & Office
IV	The Color Computer Songbook
V	My Buttons Are Blue and Other Love Poems from the Digital Heart of an Electronic Computer
VI	Color Computer Graphics

puter can handle is limited only by the imagination. In this book, we have attempted to describe ways to expand the usefulness of the computer by exploiting its video graphics capability for transmitting data from the computer to you.

This book is written for newcomers and beginners, as well as for more advanced users of microcomputers. It contains many ready-to-run programs. You type them in and the machine does the rest. This volume is a companion to *101 Color Computer Programming Tips and Tricks, 55 Color Computer Programs for Home, School & Office, 55 MORE Color Computer Programs for Home, School & Office, The Color Computer Songbook*, and *My Buttons are Blue and Other Love Poems from the Digital Heart of an Electronic Computer*.

—Ron Clark

Preface

Color video graphics is the most exciting artform to come along in this century. It also is the most useful tool in business and education since the printing press.

If a picture is worth a thousand words, a moving picture is worth a thousand still pictures. The video-graphics picture, even when frozen as a single still image, is equivalent to a moving picture because of the ability of the computer to make instantaneous changes, to correct and progress, to add and eliminate.

Businessmen, teachers and students have been heavily impressed with the ability of graphics to transmit endless streams of useful data quickly. Millions have thrilled to the avante garde forms being devised by artists.

All of the programs in this book were written on, and thoroughly tested with, a TRS-80 Color Computer, a versatile microcomputer system with a small lightweight configuration and a flexible version of the BASIC programming language. The number of jobs the Color Com-

FIRST EDITION
FIRST PRINTING

Library of Congress cataloging-in-publication data:
Clark, Ron 1940-
Color Computer Graphics
Includes index
Summary: Advice for beginning computer programmers using the BASIC programming language.
1. Computer graphics.
2. BASIC (computer program language).
3. TRS-80 (Computer)—Programming
4. Programming
5. Computers
I. Title.
T385.C53 001.64'43 81-22788
ISBN 0-86668-012-8 (pbk.) AACR2

L.C. number: 81-22788

ISBN 0-86668-012-8

Table of Contents

Introduction

Introduction

Many programs have been written for the TRS-80 Color Computer and other popular microcomputer systems. But little has been said about the most powerful functions of these computers: *graphics.*

The aim of this book is to provide a complete basic introduction to the elements of computer video graphics, especially as used in the TRS-80 Color Computer. Many complete ready-to-run graphics programs are included for your learn-by-doing fun.

This book is not so much about how to write business, education or game programs but, rather, how to write video graphics programs which can be used to enhance any other useful or just-for-fun software. The book is intended for newcomers and beginners, as well as old-timers in the programming game who never got around to adding graphics to their capabilities.

The computer instructions in this book are BASIC-language words as found in *Color BASIC* and *Extended Color BASIC*. Each program has been thoroughly tested on the TRS-80 Color Computer and is ready-to-run.

Other computers

These programs will run on other microcomputer systems programmed in BASIC but you'll have to make modifications to the program lines. Graphics commands are among the most non-standard parts of BASIC. Each computer manufacturer finds new and different words to make his system at least slightly, if not greatly different from other systems on the market.

One of the varieties of BASIC most like the Extended Color BASIC on the TRS-80 Color Computer, and in this book, is that found on the IBM Personal Computer. The instruction to set up the graphics mode on the IBM P.C. is different but most of the other graphics instructions are the same or very similar. IBM P.C. owners will find the programs, and the instruction in this book, very useful.

For owners of other systems, graphics and sound instructions will vary more widely. However, you can use the graphics instruction in this book and make only necessary modifications to program lines following the list of BASIC commands, instructions, functions, and statements found in your computer's owner's manual. Naturally, programs depending upon color for enhancement won't function on black-and-white-only TV sets, one-color video monitors, and in one-color computers.

Using this book

We have attempted to start at a very elementary level of previous computer knowledge. We assume you know how to turn on your computer, hook up whatever accessories you own, etc. You probably have written beginning programs as suggested by your computer's owner's manual.

This book has chapters explaining text versus graphics, colors versus black-and-white, dots, lines, boxes, circles, coloring objects, making objects move, and more. You will be able to come to grips with the various BASIC graphics commands, after reading these pages, and use the new knowledge to enhance your other useful and fun computer programs.

The latter part of the book includes several longer color video graphics programs for you to type and run. An appendix at the end of the book provides a handy list of

functions and statements in *Color BASIC* and *Extended Color BASIC*, operators, video control codes, graphic character codes, control keys, special characters, and error messages.

REMarks

As you read through the programs in this book, you will find very few REM or remark statements. The author's training in writing BASIC language computer programs included an emphasis on brevity and saving memory space. A sharp editing pencil was in order—and still is. Remarks and explanations, within the software, were out. Honing, fine-tuning and waste trimming were in. Use of coding form programming worksheets, such as the *Color Computer BASIC Coding Form*, the *Universal BASIC Coding Form*, and other similar tablets published by ARCsoft Publishers, was important. The objective always was, and still is, to make the most efficient use of available memory.

Some of the strings used with the DRAW instruction can be very lengthy. With a stack of such strings in a program, it's easy to forget which string draws what. So sometimes we have used the apostrophe (') with these strings. The apostrophe is a shorthand symbol, or abbreviation, for REM. We use it to indicate what the strings draw.

No two the same

Even though they may be headed toward the same goal, no two programmers will write exactly the same list of BASIC program lines from scratch. As you type the various programs in this book into your computer, you probably will make slight changes to suit your personal needs and interests. For instance, exact wording of PRINT statements can be changed. Two or more programs can be combined into one grand scheme. Applications will vary.

If you want to load more than one of these programs into your computer at the same time, be sure to use different sets of line numbers for different programs. Remember that changing line numbers may necessitate

changing GOTO, GOSUB IF/THEN and other internal references to line numbers.

The author would like to hear of improvement ideas as well as suggestions for future volumes in this series of books. He may be addressed in care of ARCsoft Publishers, P.O. Box 132, Woodsboro, MD 21798 USA.

Getting Started

Getting Started

Your personal computer is a system with four major parts: input, processor, memory and output.

Processor and *memory* are the innards, the brain which does the internal work you ask for.

Input is composed of the various parts of the equipment which allow you talk to the computer, to send in information for the memory to store and for the processor to work on. Input includes the typewriter-style keyboard, a tape, a disk, etc.

Output is the equipment available for the computer to talk back to you, to report the results of work you asked it to do. Output includes the video display screen, a line printer, or other devices.

This book is concerned with a special use of one piece of output equipment, the video display screen. We hope you will learn from these pages how to make the computer display useful pictures on the face of the video tube.

When you turn the power on, the computer knows

how to operate because the manufacturer has written software and inserted it into the computer's innards. That internal program is *system software*.

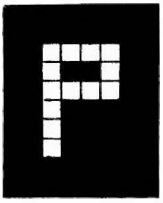
The computer can go beyond its basic internal housekeeping functions to do real-world jobs you ask of it because you write additional programs for it to follow. Your added instructions are *applications software*.

This book, then, will show you how to write applications software especially to create pictures on the video display.

You hear a lot of talk, these days, about various types of *resolution*. Some graphics are said to be *low-resolution*. Some are *high-resolution*. There is a middle ground which could be thought of as *medium resolution*. What's the difference?

Low vs. high resolution

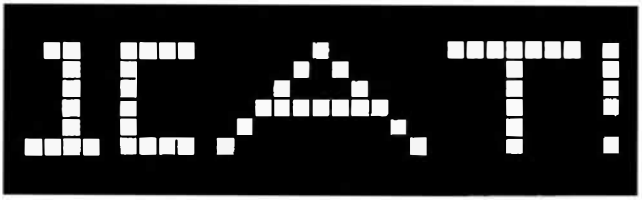
Letters, symbols, numbers, entire words, pictures, charts, graphs, anything displayed on the face of your TV screen or video display monitor is created as a series of lighted dots against a dark background. Imagine your TV screen as a large grid of tiny square rectangles like a piece of graph paper. Suppose you wanted to create the letter P on that grid, as in this approximate drawing:



The overall screen is dark. The light spots, when viewed together, create the image of the letter P. Your education leads you to see the letter P rather than an assortment of 13 white spots against a black background.

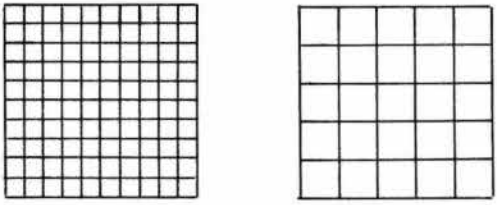
To create the letter P on the face of your TV, the computer lights several small rectangular dots in a pattern you recognize as P. The same for the letters C and A and

T, the number 1 or the symbol we call an exclamation point or any others you can think of:



The size of the face of a TV set is fixed, but it is possible to make the lighted dots larger or smaller. The smaller the dot, the more dots we can squeeze onto the face of the video screen. Like creating graph paper with ever-smaller squares, the more dots we squeeze onto the face of the video tube, the less likely you are to be able to see any one dot.

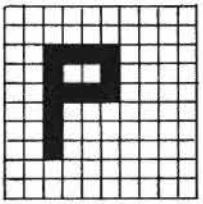
Fewer dots filling a screen mean each dot is bigger, more easily seen. More dots filling a screen mean smaller dots, each less easily seen. For example, look at these two grids. Each is the same size. But one has twice as many small squares in it.



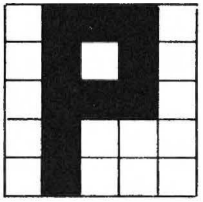
Let's try our letter P in each of two grids. The P on the left,

below, contains more dots. We'll call it "high resolution" since it has a higher number of dots in the same space.

The P on the right contains fewer dots. We'll call it "low resolution" since it contains a lower number of dots in the same space:



High Resolution



Low Resolution

If we had a P with more dots than in our low-resolution P, but with fewer dots than in our high-resolution P, we would have a medium-resolution P.

All information transmitted to you from the computer on the video screen is created the same way, as a pattern of lighted dots.

Text vs. graphics mode

Radio Shack, like most other manufacturers of personal computers, makes its color computer so you have a choice of a *text mode* or a *graphics mode*. These modes refer to the kind of display you can make the computer present on the video tube.

Text mode is used for common letters, numbers, symbols, words, formulas and other kinds of frequently-used English-language communication. In the text mode, the computer calls upon data imbedded in its permanent memory to create the patterns of lighted dots we will recognize as letters of the alphabet or numbers or symbols.

The quantities and descriptions of those patterns of lighted dots are previously established inside the computer and beyond your control. Call for the letter A and

you'll always get the same A. You cannot make that text-mode A short-legged or fatter or slimmer. In text mode, an A is an A is an A...

Graphics mode, on the other hand, is your own personal sketch pad. You can draw shapes and sizes of all sorts of characters and figures to suit your own desires.

In the graphics mode you can't call on the A stored in the computer's memory. If you want an A you have to create an A. In fact, the many letters, numbers and symbols provided by the computer in text mode must be created individually if wanted in the video mode. Later in this book we'll show you how to draw letters on the screen in the graphics mode.

When you turn power on, your color computer wakes up in the text mode. Many of the BASIC words you use in programs automatically create text displays. For instance, use of the PRINT instruction makes a text display. Even if you are in the graphics-screen mode, the computer will switch back to text mode to display your message when it encounters a PRINT instruction.

To switch your computer into the graphics mode, you must write the BASIC word SCREEN into your program. When the computer encounters the word SCREEN, it actually takes two commands from you. One is the type of screen you want, whether text or graphics. The second is the set of colors you want. More about colors later.

You tell the computer which screen type you want by using either the number zero or the number 1 after the word SCREEN. Here's the correct format for using the word SCREEN:

SCREEN *type, color set*

If you want a text screen, use the number zero for type. If you want the graphics screen, use the number 1 for type. The color set choice also is either zero or one. Here's a typical way to ask for graphics screen:

SCREEN 1, Ø

The number 1 immediately after the word SCREEN tells the computer to switch to graphics mode.

Remember, the computer wakes up in text mode and automatically switches to text mode when given instructions intended for use in text mode. To get into graphics

mode you must intentionally switch the screen using the SCREEN instruction.

Text-mode graphics

Remember the idea of low, medium and high-resolution graphics? Well, the fewest dots are found on the screen in text mode. And it is possible to light up individual dots while in text mode. Thus, low-resolution graphics are possible in text mode. You can draw letters, numbers, shapes and sizes while in text mode.

There are three other resolutions available when you switch into the graphics mode. Each has more dots than the text screen and can be called medium-resolution and high-resolution.

For our purposes, we will refer to low-resolution graphics as those in the text mode and medium and high-resolution as those done in graphics mode.

How do we do graphics in the text mode? Try this brief program:

```
10 POKE 1200,255
20 END
```

The computer, when you run this program, will light up a spot just above the center of the video display screen. Now type in this program and RUN it:

```
100 RESET (30,10)
110 END
```

This will create a small square dot on the screen right next to the one lighted by POKE 1200,225. The POKE'd dot is orange and the RESET dot is black, both against a green background. Be sure to include the parentheses in the RESET program line.

Now add this small program to the list in your computer's memory.

```
10  CLS
20  B$ = INKEY$
30  IF B$ = "U" THEN Y = Y-1
40  IF B$ = "D" THEN Y = Y + 1
50  IF B$ = "R" THEN X = X + 1
60  IF B$ = "L" THEN X = X-1
70  RESET (X,Y)
80  B$ = ""
90  GOTO 20
```

This program will allow you to draw a maze of letters or numbers or other figures on the video screen in text mode. Here's how it works:

When you type in the program and RUN it, line 10 will erase everything from the screen. The BASIC word CLS as used in the color computer stands for "clear the screen of everything."

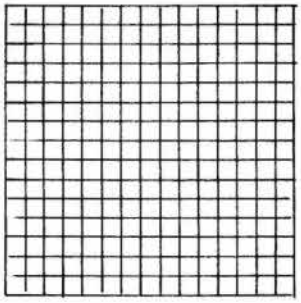
At line 20, the computer uses the powerful INKEY$ function to await your command from the keyboard. You will use only four keys. The D key to draw a line downward, the R key to draw a line to the right, the U key to draw a line upward and the L key to draw a line to the left. No other keys (except the BREAK key) will function during this run.

Be sure to avoid drawing the line off the edge of the screen or you will get an error message. Also, due to slow response internally, you may have to press the letter key you want more than once to get a response from the computer.

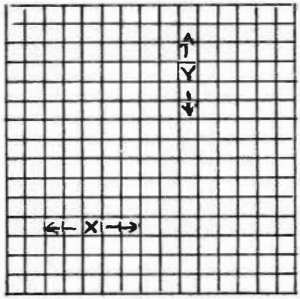
The computer will start with a black dot (against a green background) in the upper left hand corner of the video screen. Try pressing D a few times to move the line down. Then press R to move the line in the right-hand direction. Then press U to take the line up and L to take it left. Now move the end of the line all over the screen, if you like. Fun!

Video graph paper

Remember we said the TV screen can be imagined as having a grid like graph paper? Well, like graph paper you can precisely locate one spot on the face of the screen by counting rows and columns. Here's a grid:



Now, suppose we thought of all the horizontal rows as X and the vertical columns as Y. We might think of lines moving across the TV screen as moving in the X direction and lines moving up and down the screen as moving in the Y direction.



Count the dots across the grid. Start on the left and count toward the right. As you move toward the right hand side of the grid you get more and more dots. The number of dots is increasing. Each new dot adds one to the total. Each new dot is *plus* one.

Now move backward, right to left. Each new dot subtracts one from the total previously counted. Each is *minus* one.

To move left to right, then, add one to the value of X. To move right to left, subtract one from the value of X.

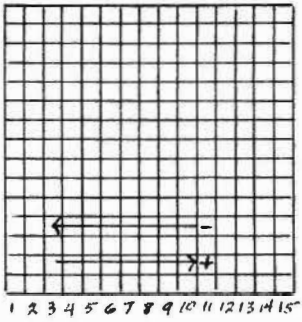


Figure 8

Similarly, to go up or down the screen, the value of Y changes.

Count the dots from bottom to top of the grid. Start at the bottom and count toward the top. As you move toward the top, you get more dots. The number of dots is increasing. Each new dot adds one to the total. Each new dot is *plus* one.

Now, move downward, from top to bottom. Each new dot subtracts one from the total previously counted. Each is *minus* one.

To move bottom to top, then, add one to the value of Y. To move from top to bottom, subtract one from the value of Y.

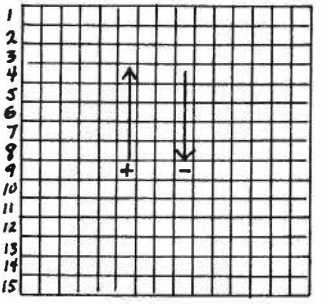
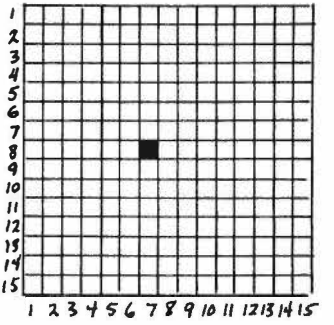


Figure 9

You will note that the position where X,Y is 1,1 is in the upper left hand corner of the grid in figures 8 and 9. What would the lower left hand corner be? Since it is in the fifteenth position for both X and Y it would be 15,15.

Any position on the screen can be located as an X,Y point. For instance 1,1 or 15,15 or 7,8. Where is 7,8?



Remember we said there are several different resolutions possible for color computer graphics? The two lowest-resolution sets are in the text mode and the three highest-resolution sets are in the graphics mode. There are, actually, a total of five different resolutions for you to select from.

Imagining the video screen as a grid, here's the number of X dots and Y dots available in each resolution:

Resolution	Mode	X by Y size
Low	Text	32 x 16
Low	Text	64 x 32
Medium	Graphics	128 x 96
Medium	Graphics	128 x 192
High	Graphics	256 x 192

Thus, the lowest resolution offers 32 points across the screen and 16 down the screen, for you to control. The highest resolution offers 256 points across the screen and 192 points down the screen for your use. With so many more dots available in the high-resolution set, you can see why drawings are of a finer quality. Drawings in the text mode are much less refined.

Exact reproductions of these screen dot patterns are shown as graph paper on pages 172 to 176 of the color computer owner's manual, *Going Ahead With Extended Color BASIC*.

Low-resolution lines

Let's use the two lowest-resolution sets to draw lines on the screen.

Suppose we want a horizontal line drawn all the way from left to right across the screen at about the midpoint between top and bottom. We have two ways we can draw that line in the text mode. We can use the PRINT @ instruction or the RESET instruction. The first program below uses the PRINT @ instruction:

```
10 FOR L = 224 TO 255
20 PRINT @ L,CHR$(255)
30 NEXT L
```

When using PRINT @ there are 32 points across the screen and 16 down. That's 16 rows and 32 columns. The upper left-hand corner is numbered 0,0 by the computer. The upper right-hand corner, then is 31,0. For conve-

nience, the computer numbers each dot left to right, top to bottom, across and down the video screen for the PRINT @ instruction. Thus the dot in X,Y position 31,0 is numbered 31. The first dot in the second row, which is at X,Y position 0,2 is numbered 32. The middle of the screen is numbered 240. The lower left-hand corner is 480 and the lower right-hand corner is 511.

If you use the PRINT @ instruction to create low-resolution, you must convert the X,Y position to one of the sequential numbers.

That's why we use the numbers 224 to 255 in the FOR/NEXT loop in the program above. The actual line is drawn by the PRINT @ instruction in line 20. The FOR/NEXT loop in lines 10 and 30 cause the PRINT @ to move across the screen from location 224 to location 255, lighting dots as it progresses. The result of a run is a bright orange line across the screen from left to right.

Note that the bright orange line is very thick. You might even call it a bar. Thinner lines come with higher-resolution graphics, as you'll see later.

Now type in this additional program:

```
100 FOR X=0 TO 63
110 RESET (X,16)
120 NEXT X
```

Here we use the RESET instruction to create a line across the screen immediately below the orange line drawn by the last program.

RESET requires its instructions in the X,Y position notation. We use the FOR/NEXT loop to cause points to be RESET from position 0,16 to 63,16. The result is a fat black line across the screen.

Video memory poke

You may recall the program with the line POKE 1200,255. That's yet another way to draw a line across the screen while in the text mode.

The computer knows at all times what it is displaying on the video screen. The information about what is being done with each dot is stored in memory. You can look into memory with the PEEK instruction to see what is happening at a particular location. The information at one

memory location will tell you what is happening at the corresponding dot on the screen.

Similarly, you can change the contents of a memory location using the POKE instruction. Try this program again:

```
10  POKE 1200,255
```

Running that program line causes the screen character identified by number 255 to be poked into memory location number 1200. The character numbered 255 is an orange dot.

How can you draw a line with the POKE instruction? Use the FOR/NEXT loop for repeated dots in a line.

```
200 FOR A=1312 TO 1343
210 POKE A,255
220 NEXT A
```

Type in that program and run it. You'll find an orange bar across the screen.

Now let's put the three line programs together and see what results:

```
 10 FOR L=224 TO 255
 20 PRINT @ L, CHR$(255)
 30 NEXT L
100 FOR X=0 TO 63
110 RESET (X,16)
120 NEXT X
200 FOR A=1312 TO 1343
210 POKE A,255
220 NEXT A
300 GOTO 300
```

We'll add a never-ending loop at line 300 to freeze our picture on the screen. To end the run, press the BREAK key.

See how it works? You have three different ways to cause patterns of lighted dots to appear on the screen. You can use the PRINT instruction with the @ symbol to specify location. You can use RESET. And you can use POKE.

Draw a box

Let's use these three kinds of low-resolution drawing abilities to create some boxes on the video display. First, type in this program which demonstrates the use of PRINT @ to create an orange three-dimensional box:

```
10  CLS
20  DATA 11,12,13,14,15,16,17,18
30  DATA 19,20,42,51,52,73,82,84
40  DATA 104,113,116,135,144,148
50  DATA 166,167,168,169,170,171
60  DATA 172,173,174,175,180,198
70  DATA 207,212,230,239,244,262
80  DATA 271,275,294,303,306,326
90  DATA 335,337,358,367,368,390
100 DATA 391,392,393,394,395,396
110 DATA 397,398,399
200 FOR L=1 TO 61
210 READ P
220 PRINT @ P,CHR$(255)
230 NEXT L
```

Figure 13 shows the resulting box. Each individual point, lighted during program run, is stored in the data lines numbered 20 through 110. The FOR/NEXT loop in lines 200-230 causes the DATA to be read 61 times. The actual PRINT @ instruction is located in line 220.

The fat orange dot, represented by the character number 255, is printed 61 times during the loop operation.

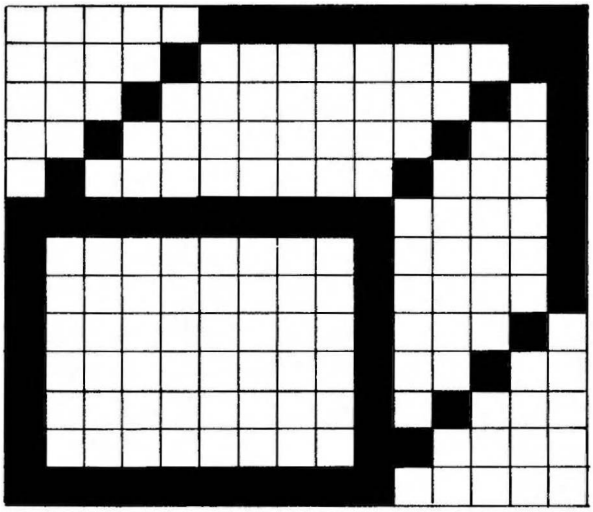


Figure 13

Each such printing is at a different location. The end result is a group of orange dots arranged in a pattern to look like a three-dimensional box.

Now let's try to make the same box using the POKE instruction. We will POKE character number 255 into a set of video-memory locations with a box on the screen as a result.

For convenience we have made most of the program exactly the same as that you just typed and used for the PRINT @ example. We only make changes to lines 220 to 240. Lines 10 through 210 remain unchanged. Change these lines:

```
220 N = P + 1060
230 POKE N,255
240 NEXT L
```

When you run this program you will get the same box as before. See figure 13. The difference is in the way you create the box on the screen.

With PRINT @ you instructed the computer to display a character at a specific location on the screen. With POKE you instruct the computer to put the character in memory at locations where it stores video-screen information. The computer looks in a video-screen memory location, sees the printable character you placed there and prints it on the screen. Either way, you get the orange box.

Now use RESET to create the box. Remember we said there are two low resolutions available in the text mode? PRINT @ and POKE, as we used them here, create the lowest-possible resolution of the box. RESET will be the higher of the two low resolutions available in the text mode. RESET has twice as many dots available. So, using the same points, our box will be only half as large. It is, in effect, drawn with finer detail.

PRINT @ uses 32 points across the screen and 16 points down the screen. RESET uses 64 points across and 32 down.

Using RESET you specify individual screen locations by X,Y. Here's a program to draw our three-dimensional box (in black) on the video screen:

```
10 FOR X = 8 TO 17
20 RESET (X,2)
```

```
 30 NEXT X
 40 RESET (7,3)
 50 RESET (16,3)
 60 RESET (17,3)
 70 RESET (6,4)
 80 RESET (15,4)
 90 RESET (17,4)
100 RESET (5,5)
110 RESET (14,5)
120 RESET (17,5)
130 RESET (4,6)
140 RESET (13,6)
150 RESET (17,6)
160 FOR X = 3 TO 12
170 RESET (X,7)
180 NEXT X
190 RESET (17,7)
200 RESET (3,8)
210 RESET (12,8)
220 RESET (17,8)
230 RESET (3,9)
240 RESET (12,9)
250 RESET (15,9)
260 RESET (3,10)
270 RESET (12,10)
280 RESET (16,10)
290 RESET (3,11)
300 RESET (12,11)
310 RESET 15,11)
320 RESET (3,12)
330 RESET (12,12)
340 RESET (14,12)
350 RESET (3,13)
360 RESET (12,13)
370 RESET (13,13)
380 FOR X = 3 TO 12
390 RESET (X,14)
400 NEXT X
500 GO TO 500
```

Again, we've added a never-ending loop at line 500 to freeze the picture. Press the BREAK key to end the run.

As you can see from the program listing, use of

RESET in this case drew a smaller slightly-higher-resolution box but the program used more memory. The number of program lines is greater than the number required to draw the same box using the PRINT @ and POKE instructions.

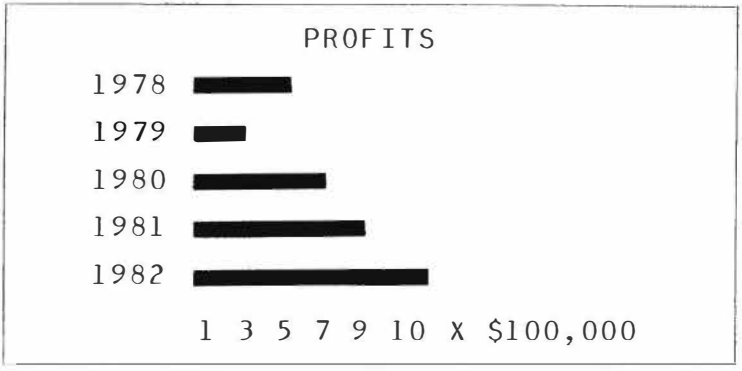
The main advantage to making video drawings in the text mode is the ease of adding words, numbers and other labels to your art. For example, suppose you want to create a bar graph on the screen. Type in this program:

```
 10 CLEAR:CLS
 20 F=100000
 30 LINE INPUT "1978 PROFITS: $";VV$
 40 V=VAL(VV$):V=V/F
 50 LINE INPUT "1979 PROFITS: $";WW$
 60 W=VAL(WW$):W=W/F
 70 LINE INPUT "1980 PROFITS: $";XX$
 80 X=VAL(XX$):X=X/F
 90 LINE INPUT "1981 PROFITS: $";YY$
100 Y=VAL(YY$):Y=Y/F
110 LINE INPUT "1982 PROFITS: $";ZZ$
120 Z=VAL(ZZ$):Z=Z/F
130 CLS
140 PRINT @ 74,"PROFITS"
150 PRINT @ 128,"1978"
160 PRINT @ 192,"1979"
170 PRINT @ 256,"1980"
180 PRINT @ 320,"1981"
190 PRINT @ 384,"1982"
200 PRINT @ 133,STRING$(V,176)
210 PRINT @ 197,STRING$(W,176)
220 PRINT @ 261,STRING$(X,176)
230 PRINT @ 325,STRING$(Y,176)
240 PRINT @ 389,STRING$(Z,176)
250 PRINT @ 421,1
260 PRINT @ 423,3
270 PRINT @ 425,5
280 PRINT @ 427,7
290 PRINT @ 429,9
300 PRINT @ 432,"10 X $100,000"
```

Sample Run

```
RUN ENTER

1978 PROFITS: $
675345  ENTER
1979 PROFITS: $
465987  ENTER
1980 PROFITS: $
789456  ENTER
1981 PROFITS: $
998567  ENTER
1982 PROFITS: $
1250455 ENTER
```



See how easy it is to add the years in a vertical column on the left-hand side of the screen and the dollar amounts across the bottom. And, of course, the label at the top of the screen.

Bar Graphs

Here's a convenient way to convert information or data to an easy-to-read bar graph.

```
 10 CLS
 20 A=RND(26):B=RND(26):C=RND(26)
 30 D=RND(26):E=RND(26):F=RND(26)
100 PRINT @ 32,"1981 ";STRING$(A,191)
110 PRINT @ 96,"1982 ";STRING$(B,191)
```

```
120 PRINT @ 160,"1983 ";STRING$(C,191)
130 PRINT @ 224,"1984 ";STRING$(D,191)
140 PRINT @ 288,"1985 ";STRING$(E,191)
150 PRINT @ 352,"1986 ";STRING$(F,191)
200 FOR H=388 TO 414 STEP 3
210 PRINT @ H,H-388
220 NEXT H
230 PRINT @ 424,"MILLIONS OF DOLLARS"
300 A$=INKEY$
310 IF A$=""THEN 300
320 GOTO 10
```

This program uses PRINT @ to create the bars of a graph. Main construction of the graph is program lines 100 to 220. The length of the bars in the graph are controlled by the variables A, B, C, D, E and F in lines 100 to 150.

If you want to feed data from some other computation into this graph, you must arrive with values stored in A through F. The values should be in the range of zero to 26.

For demonstration purposes, we get those values here by using the random number generator in lines 20 and 30. Line 20 creates values we need for A, B and C. Line 30 generates numbers for D, E and F.

In this case, we have made the bars of the graph red. That's why we use the character number 191 at the ends of lines 100 to 150 in the STRING$ function. You can make your bars other colors by substituting these character numbers:

Number	Color
128	Black
143	Green
159	Yellow
175	Blue
191	Red
207	Buff
223	Cyan
239	Magenta
255	Orange

The Y-axis of the graph is labeled with year dates from 1981 to 1986 (lines 100 to 150) in our sample program. The X-axis is labeled by lines 200 to 230. The loop in lines 200 to 220 prints the row of numbers beneath the bars.

Lines 300 to 320 are used to allow you to generate new random numbers and new graphs at the press of any keyboard key.

Graphics Mode

It may be easy to add labels to artwork in the text mode but the big clunky drawings leave something to be desired. You need to know how to get more streamlined artwork. You need the graphics mode.

The switch to convert the display from text mode to graphics mode is the instruction SCREEN. Remember it is used in the form:

SCREEN *type, color set*

Let's compose a small program which will demonstrate this switching of the display. First we need to generate some "garbage" on the text screen:

```
10  FOR L=1 TO 9
20  PRINT "SCREEN GARBAGE"
30  NEXT L
```

If you run that three-line program you'll get nine lines of the words SCREEN GARBAGE. The computer will remain in the text mode and follow the text-mode instruction, PRINT, to make the display. Now add these two lines:

```
40  SCREEN 1,0
50  GOTO 50
```

When you run the program, action starts at line 10 but everything happens so quickly you'll probably not see the results of the PRINT instruction in line 20. What you probably will see will be a full blank screen of green color. This is the graphics screen. You may use a variety of graphics commands to instruct the computer to make drawings on that graphics screen.

For the moment, the graphics screen is blank as the computer only was instructed to turn it on (line 40) and freeze it at that point (line 50).

Try modifying line 40 so that the type of screen called for is the text screen. It should then look like this:

```
40  SCREEN 0,0
```

Running the program with SCREEN 0,0 results in the display remaining in the text mode so you see the lines which say GARBAGE SCREEN. Now change it back to SCREEN 1,0. It should look like this:

```
40 SCREEN 1,0
```

Color in the computer is so fascinating, let's stop and play with that for a moment. Change the color set number in line 40 to a one. It should look like this:

```
40 SCREEN 1,1
```

Now, when you run the program, the computer will encounter the instruction at line 40 to switch to graphics mode. And, in that same line, it will find an instruction to change color sets. The screen will change from a bright green to a light background.

Now put the instruction back to SCREEN 1,0 so line 40 looks like this:

```
40 SCREEN 1,0
```

Also delete line 50 so our next additions to the program will run after line 40.

A higher-resolution line

Delete line 50 and add the following lines, 100 and 110, to the program:

```
100 LINE(0,110)-(255,110),PSET
110 GOTO 110
```

Try running the program with the new lines. What do you get? You should find a line from the left-hand edge of the graphics screen to the right-hand edge of the graphics screen, about half-way down the screen.

One thing should be very noticeable at this point. There is a border around the graphics screen. You can't make the computer draw out in that border. Using color-set zero, you will see the same color screen inside the graphics area as outside in the border. If you look closely at the face of your video display you should be able to see a faint line around the graphics area. Most TV sets will show a very slight difference of display between the border and the graphics area.

The important thing to remember is that your artwork will be confined to the graphics area with none out in the border. When we refer to the number of points across or

up and down the graphics screen, we mean inside that graphics area, not out in the border.

A flashy box

The result of running the program, including lines 100 and 110, will be a line across the graphics area. Now you know of four ways to draw a line on the display.

Line 110, by the way, is our old friend, the frame freezer. It is a never-ending loop to hold the last picture in place. To end the run, press the BREAK key.

But where has our GARBAGE SCREEN message gone? You will recall we printed nine lines of it on the text screen but went on to the graphics screen so quickly it looked as if it might have been lost. But, no. Press the BREAK key.

The result is an ending of the program run and return by the computer to the text screen. It displays whatever it last had on the text screen because that was the last thing stored in text-mode display memory.

In other words, we ordered the computer to display nine lines of SCREEN GARBAGE. It did that and then went on to other business. During the time it was doing that other business, it remembered the display for the text mode. When we ended the run, the machine switched back out of the graphics mode to text mode and resumed displaying the contents of its text screen memory.

Let's continue to build on this program we have typed into the computer. Here's how to make the computer display the line we have drawn in the graphics mode for only a moment and then erase the line:

```
200 FOR L=1 TO 200:NEXT L
300 PCLS
310 GOTO 310
```

Line 200 is a loop which appears to do nothing because it makes no output from the computer. It is, in fact, a time delay to keep the computer busy for a short period of time so you can have time to see the line previously drawn on the screen. To increase the length of time in the delay loop, increase the number 200 in line 200. To decrease the time in the loop, reduce the number 200 in line 200.

Line 300 clears the screen. Line 310 is our old friend,

the freeze-action instruction. We freeze action by causing the computer to go into a never ending loop.

What's that P?

Remember how you use the instruction CLS to clear the screen in text mode? Well, PCLS is the instruction to clear the graphics video screen.

The P comes form the word *page*. In the Color Computer, we visualize a screen as a page. Each page uses up 1.5 kilobytes of memory or, to be more exact, 1536 bytes. That is, each page of graphics-screen info is stored in 1536 memory locations. This video memory is large enough to hold up to a total of eight pages. You can store up to eight different pages of video drawings.

Radio Shack, in composing the various names for the special BASIC graphics instructions in the color computer decided to attach the letter P to the beginnings of many of those words. We'll soon know about PCLEAR, PMODE, PCOPY, PPOINT, PSET and PRESET.

When they needed a screen-clear instruction especially for the graphics screen, they chose PCLS as we have used in line 300 above.

Since it's so easy to make a line disappear, wouldn't it be nice to make it flash on and off? Delete line 310 and try this program:

```
400 FOR L=1 TO 200:NEXT L
410 GOTO 100
```

Remember that line 300 erased the line? Line 400 is a time-delay loop to hold the erased display briefly. Then line 410 pushes action back up to line 100 where the line is redrawn.

The alternating displaying and erasing of the line causes it to appear to blink on and off. It will do this endlessly until you press the BREAK key.

For a fatter line, add a comma and the letters BF at the end of line 100. Line 100 should look like this:

```
100 LINE(0,100)-(225,110),PSET,BF
```

The letter B at the end of the LINE instruction causes the computer to draw a rectangle. The corners of the rectangle will be at X,Y locations 0,100 and 255,100 and 0,110 and 255,110. Since the LINE command takes up only one

program line, this is a fast and easy way to draw a box!

Even more exciting is the ability to fill that box with a color, different from the background, instantly. Use the letter F at the end of the LINE instruction, after the B as we have done in line 100, to fill in the box with color. F stands for fill.

So, our revised line 100 changes the thin line into a box and fills it with a color.

PMODE is not pie with ice cream

The PMODE instruction is used to select one of the eight graphics pages and to put the computer into either medium or high resolution. The format of the instruction is:

PMODE *mode, start-page*

The modes are numbered from zero to five. The pages are numbered one to eight. If you don't specify a mode, the computer will select mode 2. Once you have selected a mode, the computer stays in that mode until you select another.

If you omit the PMODE statement, the computer automatically switches into PMODE 2,1. If you don't tell it the start page, it starts on the last page you were on.

Remember the computer sets aside 1536 bytes of memory for each page of graphics. It automatically sets up *four* such pages unless you tell it some other number of pages. With each page needing 1.5K of memory set aside, those four pages take up 6 of the 16 kilobytes in your computer's memory.

You can change the number of pages set aside for graphics by using the PCLEAR instruction. The format for the instruction is:

PCLEAR *number*

The number is from one to eight. The computer wakes up, when you turn its power on, at PCLEAR 4. It will stay at PCLEAR 4 until you give it some other number. If four pages are okay, you don't need to use the PCLEAR instruction at all.

PMODE, PCLS and other instructions have things to do with the color choices you will make for your graphics but we'll come back to that. First, let's take a closer look at those graphics pages. Remember that you specify the

particular graphics page you see by using the PMODE statement.

For the time being, we'll stick to the medium-resolution graphics in PMODE 1, *start-page*. More on resolutions later.

Here's a program to draw a line on the graphics screen:

```
10 PMODE 1,1
20 PCLS
30 SCREEN 1,0
40 LINE(10,50)-(100,50),PSET
50 GOTO 50
```

You'll see PMODE 1,1 in the beginning of the program at line 10. That instruction is telling the computer you are going to use some medium-resolution graphics on graphics page number 1.

Line 20 clears the screen and line 30 instructs the computer to shift the screen into graphics. PMODE sets up the computer to use page 1, PCLS clears the graphics screen, and SCREEN switches the computer to displaying graphics.

Line 40 draws the line on the screen and line 50 is our familiar frame-freezing loop. Without line 50, the action would come to the screen and be gone so quickly you might not be able to see it! Line 50 holds the picture until you press the BREAK key.

So, we have drawn a line from X,Y position 10,50 to position 100,50 on page 1 and displayed it on the screen. Now let's draw a line at a different location and on a different page.

We'll select page 2 by using PMODE 1,2 and we'll draw the line from X,Y position 10,100 to position 100,100. Change line 50 and add lines 100 to 150 to our program:

```
 50 FOR L=1 TO 100:NEXT L
100 PMODE 1,2
110 PCLS
120 SCREEN 1,0
130 LINE(10,100)-(100,100),PSET
140 FOR L=1 TO 100:NEXT L
150 GOTO 10
```

Remember that line 40 drew a line on the screen by placing that line on page 1 and displaying it. Line 50 is a brief

time delay so you can observe that line for a moment.

Line 100 switches the computer's innards to working on graphics page 2. Line 110 clears the graphics screen, thus getting rid of the first line we drew. Line 120 keeps the machine in the graphics-display mode.

Line 130 draws a new line at X,Y position 10,100 to 100,100. Line 140 is another brief time delay so you can have a chance to observe, momentarily, the line drawn by program-line 130. Then line 150 shoots action back up to line 10 where the whole process starts over.

At line 10 the computer finds PMODE 1,1 and goes back to writing on graphics page 1. The result of a continuous run is a line alternating between two locations on the screen. It gives the appearance of motion or animation, as the line seems to jump back and forth.

PCOPYcat

You can copy drawings from one page to another using the PCOPY instruction. The correct format is:

PCOPY *source TO destination*

The source can be any page on which you already have drawn something. The destination can be any page as long as you have used PCLEAR to set aside enough memory.

There are eight graphics pages. The computer starts its work day in PCLEAR 4. If you haven't changed that, you can PCOPY from page 1, 2, 3, or 4 to page 1, 2, 3, or 4. If you have used PCLEAR 5, PCLEAR 6, PCLEAR 7, or PCLEAR 8, you can copy to one of those page numbers.

For example, suppose you have drawn something on grahics page 1 and you want to copy that something onto page 3. You would use PCOPY 1 TO 3 in your program.

Here's how to add a PCOPY demonstration to the most-recent program we have been building. Delete line 150 and add lines 200 to 270:

```
200 PMODE 1,3
210 PCLS
220 SCREEN 1,0
230 PCOPY 1 TO 3
240 FOR L=1 TO 100:NEXT L
250 PCLS
```

```
260 FOR L=1 TO 100:NEXT L
270 GOTO 10
```

The use of PMODE 1,3 in line 200 causes the computer to move to graphics page 3. The PCOPY instruction in line 230 moves the line drawn on page 1 (at program line 40) onto page 3.

The line on page 3 is displayed and that display is held for a moment by the time-delay loop in line 240. Then line 250 clears the graphics display. That blank screen is held for a moment by the time-delay loop in line 260. Then action is pushed back up to line 10 where the program starts over.

As a result of running lines 10 to 270 over and over continuously you will see the line on page 1 displayed, the different line on page 2 displayed and then the line on page 3 displayed, all repeatedly until you press the BREAK key.

The fat line moves

Just for fun, figure out a short program to set up graphics page 1, clear the graphics screen, switch the computer into graphics mode and draw a fat line from left to right across the screen. Here's one way:

```
10 PMODE 1,1
20 PCLS
30 SCREEN 1,0
40 FOR R=0 TO 255
50 LINE(R,25)-(R,50),PSET,BF
60 NEXT R
70 GOTO 70
```

Remember that the BF on the end of line 50 causes the LINE instruction to draw a box and fill it in with color. The loop in lines 40 to 60 causes this to happen 256 times.

There are 256 points across the screen and 192 from top to bottom. The line is 25 points fat and 256 wide.

Like that line movement? How about making it move from right to left across the screen? Try changing line 40 so it reads like this:

```
40 FOR R=255 TO 0 STEP -1
```

You can make this program run slightly faster and take up fewer program lines, saving on memory space, by telescoping lines 10, 20 and 30 into one line. Try this:

```
10  PMODE 1,1:PCLS:SCREEN 1,0
```

Computer graphics should look like *graphics*, right? How about converting that fat horizontal line into a big letter T. Here's a way to modify the program by deleting line 70 and adding lines 100 to 130:

```
100 FOR D=50 TO 190
110 LINE(110,D)-(145,D),PSET,BF
120 NEXT D
130 GO TO 130
```

The loop in lines 100 to 120 draws the vertical leg of the letter T. Line 130 freezes the picture for you to see it. To end the run, press BREAK.

Medium vs. high resolution

PMODE is a useful instruction. It not only is used to designate the start page but also to select the resolution of the graphics. The higher the resolution, the finer the detail, the smaller the individual point lit on the screen.

We said a while ago that the Color Computer can perform in five different graphics resolutions. The two lowest resolutions, with the largest lit points, are in the text mode. The three higher resolutions are in the graphics mode. The number of points on the screen are:

resolution	mode	x by Y size
low	text	32 x 16
low	text	64 x 32
medium	graphics	128 x 96
medium	graphics	128 x 192
high	graphics	256 x 192

The text mode is SCREEN type zero. Any time you call for text output to the video screen, the computer automatically does a SCREEN 0,0 switch. To get medium or high-resolution graphics, you must use PMODE number zero through five and SCREEN type one.

PMODE 0 and PMODE 1 get the 128x96 medium resolution. PMODE 2 and PMODE 3 select medium resolution of 128x192. PMODE 4 selects the 256x192 highest resolution.

PMODE #	resolution	X by Y size
0	medium	128 x 96
1	medium	128 x 96
2	medium	128 x 192
3	medium	128 x 192
4	high	256 x 192

PMODE 4 lights the smallest individual dot on the screen. The dot lit by PMODE 3 or PMODE 2 is twice as large as that in PMODE 4.

The dot lit in PMODE 0 or PMODE 1 is twice as large as the dot lit by PMODE 2 or PMODE 3, and four times as large as that lit by PMODE 4.

High resolution needs four times as many dots lit to fill the screen as PMODE 0 or PMODE 1.

The computer creates the larger dots by combining two or four smaller dots. So, to draw on the screen you must specify exact X,Y locations on a 256x192 grid. That is, *all* locations are someplace between zero and 255 horizontally and 0 and 191 vertically. No matter which PMODE number you are using, you still identify locations on the screen using the 256x192 grid numbers.

For instance, 128x96 always is the center of the graphics screen. And 255,191 always is the lower right-hand corner of the screen. The X,Y location 0,191 always is the lower left corner and, of course, 0,0 is the upper left corner. The upper right is 255,0 whether you are in PMODE 0 or PMODE 4.

See the difference

Let's cut the jawboning and look at some pictures. Here's a comparison of the various resolutions. This program uses a FOR/NEXT loop in lines 10 and 70 to make the computer alternate between PMODE 0, PMODE 2 and PMODE 4. A line is drawn in each resolution from X,Y location 10,100 to location 100,100.

As you run this program, note that when the computer is in PMODE 0 the line is noticeably thicker. Each dot in the PMODE 0 line has four times as many dots in it as the line in PMODE 4. The PMODE 2 and PMODE 4 lines are so much finer that the difference in size is harder to see. However, the PMODE 2 line has twice as many dots as the PMODE 4 line.

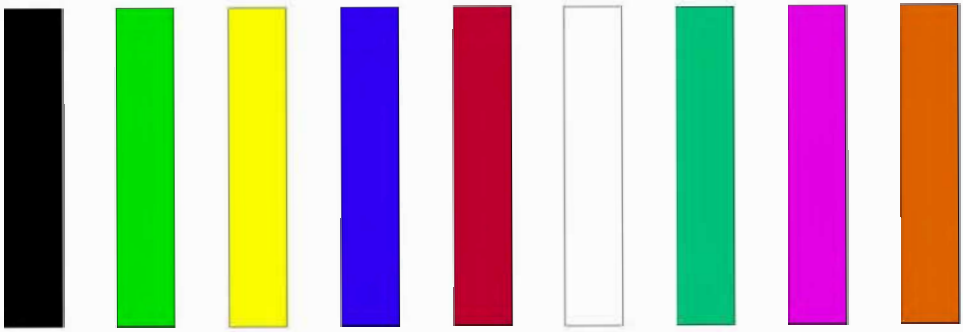
```
10  FOR L=0 TO 4 STEP 2
20  PMODE L
30  PCLS:SCREEN 1,0
40  FOR R=0 TO 5
50  LINE(10,100)-(100,100),PSET
60  NEXT R
70  NEXT L
80  GOTO 10
```

Here's an easier way to see the difference. Change program line 50 so the computer draws a box, fills it with color each time:

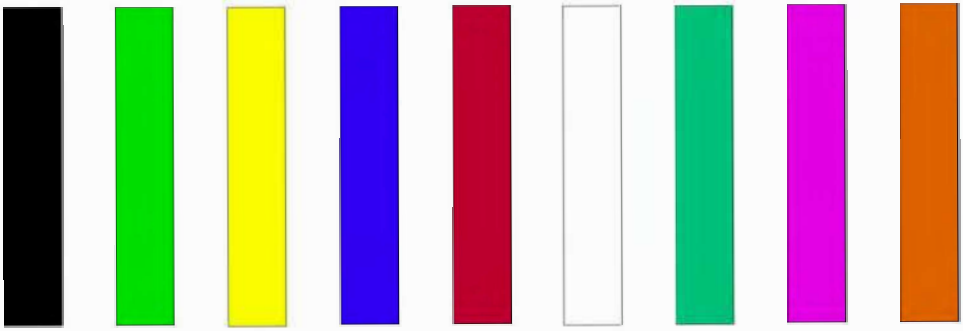
50 LINE(10,50)-(100,100),PSET,BF

When you run the revised program, you will be able to see the difference in the number of dots which have to be lit inside the box. In PMODE 0 the box is drawn and filled much faster than the PMODE 2 box and the PMODE 2 box is drawn and filled more rapidly than the PMODE 4 box. The PMODE 4 box is created much more slowly than the PMODE 0 box.

The reason: the computer has four times as many individual dot-lighting assignments to handle in PMODE 4 than in PMODE 0. It simply takes longer to do that work than in PMODE 0 where the computer is at liberty to light four dots at a time, thus completing its task more quickly.



Adding Color



Adding Color

Can you believe it? We've managed to get this far into the subject of color-computer graphics and only now we're ready for *color*. Color, at last! I know you've been waiting breathlessly so here we go.

Remember how we used RESET, in the text mode, to light a single point on the video display? SET and RESET create some really big dots on the screen. The equivalent illuminator and eraser in the graphics mode are PSET and PRESET. Since we enjoy medium and higher resolution in the graphics mode, PSET and PRESET make smaller dots as you'll see.

Let's go back and more thoroughly explore SET and RESET. Then we'll move into PSET and PRESET. Here's the correct format for using SET:

SET *X,Y,color*

You must tell the computer, each time you use set, the X,Y location on the 64x32 grid and the *color* of the dot to be displayed. Color numbers are zero to eight.

Number	Color
0	black
1	green
2	yellow
3	blue
4	red
5	buff
6	cyan
7	magenta
8	orange

The chart above shows each of the nine colors you can create on the color computer. The colors will vary depending upon the color adjustment of your own television set. However, you will be able to tell them apart.

Color zero, which we call black, actually is an absence of color. When you use SET, color zero will leave a dot's color unchanged.

The BASIC word RESET, as used in the color computer, erases a previously-set dot. Here's the proper format for RESET:

RESET *X,Y*

Note that in RESET you only use the X,Y coordinates. You don't specify a color since the computer erases a dot by returning it to the background color which makes it disappear.

Now type in this brief demonstration program to see how SET and RESET work:

```
10 CLS
20 SET(32,16,8)
```

Run this program and you'll find a fat black dot with an orange corner appears near the middle of the TV screen. The upper left corner of the black dot is orange.

Color has been removed from the X,Y location you specified as 32,16. Then color number eight, an orange color, has been turned on in a smaller spot in the upper left-hand corner of that dot-with-no-color.

To make the entire character black add lines 30 and 40:

```
30  FOR L=1 TO 100:NEXT L
40  RESET(32,16)
```

Now the fat dot ends up all black. As you run the program lines 10 through 40, line 10 clears the text screen of all display. Line 20 turns on the black-and-orange dot. Line 30 is a time-delay loop to allow you to see the black-and-orange dot momentarily. Then line 40 turns off the orange part of the fat black dot.

The black dot remains but without the smaller orange point in its corner.

To make the orange dot seem to blink on and off, add program lines 50 and 60:

```
50  FOR L=1 TO 100:NEXT L
60  GOTO 20
```

Now, once the larger black dot is established by line 20, it stays there. But the orange smaller dot is turned on and off repeatedly. Line 20 turns on the orange dot and line 40 removes the orange dot. Line 50 is a time-delay so you can see the no-orange-dot configuration. Line 60 pushes action back up to line 20 where the orange is turned on again. You press the BREAK key to stop this continuous action.

How about making the entire black-and-orange dot blink on and off? Use the CLS instruction. Change line 40 like this:

```
40  CLS
```

Now the computer will alternately display and erase the entire black-and-orange spot. Line 20 turns it on and line 40 clears the screen, effectively erasing it. Press BREAK to end the run.

Oh, so small

Let's use the same X,Y values and test the ability of the computer to set and reset points on the graphics screen. Remember, for the graphics screen the words to use are PSET and PRESET.

First, let's clear out our previous program by typing

NEW and pressing ENTER. Now type in the lines necessary to establish the graphics mode:

```
10 PMODE 1,1
20 PCLS
30 SCREEN 1,0
```

Line 10 tells the computer we will be doing some medium-resolution graphics on graphics page number one. Line 20 clears the graphics screen. Line 30 switches the computer into graphics-screen display. Add these lines:

```
40 PSET(32,16,8)
50 GOTO 50
```

Run the program. Lines 10 to 30 establish the graphics screen mode and display. Line 40 causes a point to light up in orange. Line 50 freezes the action so you can see it.

Where has the dot gone? To the upper left-hand portion of the screen. We kept the same X,Y values as in the text mode, but the text mode grid was only 64x32 while the graphics mode grid is 256x192. Where there were only 2048 locations to choose from when using SET and RESET, now we have 49,152 tiny dots we can light up!

We found the center of the SET/RESET grid to be at X,Y location 32,16. The center of the graphics-screen grid is at 128,96.

The point we have illuminated at position 32,16 is very, very tiny. It is somewhat hard to see. Let's make it easier to find by making it blink on and off. Change line 50 and add lines 60 through 80:

```
50 FOR L=1 TO 100:NEXT L
60 PRESET(32,16)
70 FOR L=1 TO 100:NEXT L
80 GOTO 40
```

Run the program. Line 50 is a time delay so you have a chance to see the point blink on. Then line 60 erases the point by turning back to the background color. Line 70 is another time delay, this time so you can see that the spot has been erased. Line 80 pushes action back up to line 40 where the dot is turned on again.

The entire program will run over and over again until you press the BREAK key.

Now that we have a blinking dot, let's make it even smaller by changing from PMODE 1 to PMODE 3. Change line 10 to this:

```
10 PMODE 3,1
```

Bet you had thought it couldn't get any smaller! Well, there it is. Smaller. Now it's really hard to see. Good thing it's blinking.

Make it even smaller by changing again, this time to PMODE 4. Change line 10 again:

```
10 PMODE 4,1
```

Run the program. Ignore the color changes. Where's the blinking dot? It's still there but so small you really have to squint to find it! Imagine, to fill the screen in mode four, the computer must work on almost 50,000 tiny dots!

Let's get back to the larger dot and reduce eye strain. And move it to the center of the screen. Change lines 10, 40 and 60:

```
10 PMODE 1,1
40 PSET(128,96,8)
60 PRESET(128,96)
```

Now we have a blinking dot we can see at the center of the screen.

Changing colors

Using the nine color numbers make writing programs fun! Erase program memory and type in this text-mode program:

```
100 FOR C=0 TO 8
110 CLS C
120 FOR T=1 TO 200:NEXT T
130 NEXT C
140 GOTO 100
```

Run the program. You will see the nine colors appear in number order on the screen.

Line 100 and line 130 form a loop to take the value stored in memory location C from zero to eight. Line 110 uses the CLS instruction and the value stored in memory location C to color the blank screen. Line 120 is a time delay so you can see the displayed color. After the computer runs through all eight colors, line 140 pushes action back to line 100 where everything starts over. You press the BREAK key to end the run.

Here's the format for the CLS instruction:

CLS *color*

You add the color number you want, from zero to eight. If

you don't use a number, the computer assumes you want a green screen. It automatically switches into color number one.

By the way, in text mode, you can only type words on a green background. You don't have the option of typing red, for instance, against a blue background. Whenever you type on the screen or use PRINT or other command resulting in typing on the screen, the computer puts those words against a green background.

Now let's try displaying the screen colors in the graphics mode. First, we'll have to establish the graphics mode in our program. Type in lines 10 and 20:

```
10 PMODE 1,1
20 SCREEN 1,0
```

Line 10 tells the computer to use medium-resolution writing on graphics page 1. Line 20 switches output to the graphics display with color set number zero. Color set zero will give you combinations of green/yellow/blue/red in PMODE 1.

The colors green, yellow, blue and red are color numbers one through four. So, change line 100 like this:

```
100 FOR C=0 TO 4
```

That will make the computer run through the colors available in color set number zero. Since we are in the graphics mode we must change CLS in line 110 to PCLS like this:

```
110 PCLS C
```

The rest of the program remains the same. LIST the program and let's review what we have.

Lines 10 and 20 establish the graphics mode and screen. Lines 100 and 130 are the zero to four color-numbers loop. Line 110 actually does the clearing of the screen and the changing of the colors on the screen. Line 120 is a time delay so you have a moment to see each color. Line 140 makes the entire operation repeat endlessly. You press BREAK to end the run.

The format for PCLS is the same as for CLS. Here it is:

PCLS *color*

Again the color numbers are zero to eight but your selection is determined by the color set specified through the SCREEN instruction. Its format is:

SCREEN *type, color set*

You remember that type choices are zero (text) and one (graphics). Color set choices also are zero (green/yellow/blue/red) or one (buff/cyan/magenta/orange).

Suppose you want to look at the other four color choices for the graphics screen. Those would be buff (number 5), cyan (6), magenta (7), and orange (8). You need to change lines 20 and 100 like this:

```
20 SCREEN 1,1
100 FOR C=5 TO 8
```

The second number 1 in line 20 selects color set number one. The color numbers five through eight are used in line 100. Here's a handy chart of the SCREEN color sets:

SCREEN color set number	four-color set
0	green/yellow/blue/red
1	buff/cyan/magenta/orange

The higher the resolution, the more memory used by the computer to do its job. For instance, PMODE 0 can be thought of as using only one-page-worth of memory space. PMODE 1 and PMODE 2 take up memory space equivalent to two pages.

The difference is in the number of dots per screen and the number of colors available for those dots. PMODE 0 is lowest of the graphics resolutions with a relatively large dot composed of four points on the screen, all lit at the same time and only in two colors.

PMODE 1, on the other hand, creates a dot with four points but each dot has four colors available. The more colors available, the more memory used to do the job.

PMODE 2 forms a medium-sized dot from only two screen points in two colors. The higher the resolution, the more memory used.

If you want to save memory space, use a lower resolution (such as PMODE 0 or PMODE 1) and use fewer colors (such as PMODE 0 or PMODE 2). You can see that PMODE 0 uses the least memory.

PMODE number	SCREEN color set number	colors	X by Y size	equivalent pages
0	0	black/greeen	128x96	1
0	1	black/buff	128x96	1
1	0	green/yellow/blue/red	128x96	2
1	1	buff/cyan/magenta/orange	128x96	2
2	0	black/green	128x192	2
2	1	black/buff	128x192	2
3	0	green/yellow/blue/red	128x192	4
3	1	buff/cyan/magenta/orange	128x192	4
4	0	black/green	256x192	4
4	1	black/buff	256x192	4

You can see that PMODE 1 and PMODE 2 use the same amount of memory space, or pages. PMODE 1 offers four colors but less resolution. PMODE 2 gives higher resolution but fewer colors.

PMODE 3, on the other hand, offers the resolution of PMODE 2 but with four colors so it uses even more memory space. The higher the resolution, or finer the lines drawn, the more memory required. The more color used, the more memory used.

Is it on or off?

There is a convenient way to tell if a particular point on the graphics screen is lit. The PPOINT function can be used as a test to see what color has been assigned to a particular dot.

If a color, other than the background color, is found, the point is lit. If the color is the same as the background, then you won't be able to see that particular dot (even though it is there in the background color). Here's the format for PPOINT:

PPOINT *(X,Y)*

Using the PPOINT instruction, the computer looks at the point on the screen specified by the X,Y location. Here's a sample program.

```
 10 PMODE 1,1
 20 PCLS
 30 SCREEN 1,0
100 C = RND(4)
```

```
200 PSET(128,96,C)
210 P=PPOINT (128,96)
220 FOR L=1 TO 500:NEXT L
300 CLS
310 PRINT"RANDOM NUMBER=";C
320 PRINT"COLOR NUMBER=";P
```

This program lights a very small dot at center screen and tells you the color of that dot.

Lines 10 though 30 establish the graphics mode. We use PMODE 1,1 for a medium-resolution dot you will be able to see and create it on page 1. We switch on the graphics screen and use color-set zero with the SCREEN 1,0 instruction.

Line 100 generates a random number from zero to four and stores that number in location C.

Line 200 prints a dot at X,Y location 128,96 in the color corresponding to the number stored in C. For instance, if line 100 generated a number three, line 200 uses that number three as a color number and colors the dot blue. Here are those color numbers again:

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

Line 210 contains our PPOINT test. It looks at screen location 128,96 and determines the color in use there. If the random number generated was three and the color of the dot at 128,96 was blue, the PPOINT test will find blue and store the number three in memory location P.

Line 220 is a time delay so you can examine the dot for yourself and see what color you think it is. Then line 300 clears the text screen in preparation for displaying the test results for you to see.

Line 310 recalls the contents of C, the original random number used to determine the color in the first place. It displays that number.

Line 320 gets the result of the PPOINT test from memory location P and displays that number.

The random number and the PPOINT test-result number will be the same. If the number in C is 3, the number in P will be 3.

Here's a longer application of the PPOINT test in a program demonstrating how you can use PPOINT to tell what color is on a video page without looking.

```
 10 PMODE 1,1
 20 PCLS
 30 SCREEN 1,Ø
100 CIRCLE(128,96),80,4
110 DRAW"C4BM128,96NU80NE55NR80
    NF55ND80NG55NL80NH55"
200 A=RND(4):B=RND(4)
210 C=RND(4):D=RND(4)
220 E=RND(4):F=RND(4)
230 G=RND(4):H=RND(4)
300 PAINT(130,89),A,4
310 PAINT(140,94),B,4
320 PAINT(140,98),C,4
330 PAINT(130,101),D,4
340 PAINT(126,101),E,4
350 PAINT(116,98),F,4
360 PAINT(116,94),G,4
370 PAINT(126,89),H,4
400 P=P+PPOINT(130,89)
410 P=P+PPOINT(140,94)
420 P=P+PPOINT(140,98)
430 P=P+PPOINT(130,101)
440 P=P+PPOINT(126,101)
450 P=P+PPOINT(116,98)
460 P=P+PPOINT(116,94)
470 P=P+PPOINT(126,89)
500 IF P=32 THEN P=Ø:GOTO 20
510 P=Ø
520 GOTO 200
```

This program draws a pie and cuts it into eight pieces. Each piece is colored separately. The colors appear in an unpredictable order.

PPOINT is used to check the color of a dot in each of the eight slices of the pie. If all eight are red at the same time, the entire pie is consumed. A new pie is created and colored.

Lines 10 to 30 establish the graphics mode and screen. Lines 100 and 110 draw the circle and slice it into eight sections.

Lines 200 to 230 generate the random numbers used later (in lines 300 to 370) to select colors for the pie slices. Lines 300 to 370 PAINT the pie slices.

By the way, we'll get into the BASIC words CIRCLE, DRAW and PAINT later so don't worry about them for now.

Lines 400 to 470 look for the specific colors in the pie slices. As the eight slices are tested, the color numbers from 1 to 4 are added together. Since red is the highest number (4), we can use 8x4 = 32 in line 500 to test for the "all slices are red" condition. The only time the total stored in memory location P, in line 500, can be 32 is when all eight pie slices are red.

If the test in line 500 finds all eight pie slices are red, the value of P is set to zero and program action is kicked back up to line 20. At line 20, the screen is cleared so a new pie can be drawn.

If line 500 finds not all pie slices are red, the program moves on to line 510 where the value of P is set at zero. Line 520 shoots action to line 200 for a new set of random numbers and, thus, new pie slice colors.

Text mode POINT

A similar test of screen points is available for your use when you create low-resolution art in the text mode. The Color BASIC function POINT has this format:

POINT *(X,Y)*

The X,Y locations are on the 64x32 grid used in the low-resolution text mode. That is, X can be anywhere from 0 to 63 and Y can be from 0 to 31.

If the result of a POINT test of particular screen location is -1, the cell is in the character mode. That is, it is displaying a preformed text letter, number or symbol.

If PPOINT finds nothing at that location, it returns a zero.

If a color dot is found, the color number is returned. Try this simple program if you have non-extended Color BASIC:

```
10 CLS
20 PRINT @ 112,"A"
30 P = POINT(32,6)
40 PRINT @ 143,P
50 SET(38,6,8)
60 Q = POINT(38,6)
70 PRINT @ 146,Q
80 R = POINT(44,6)
90 PRINT @ 149,R
```

The program prints an A, an orange graphics dot, and nothing at one point. It then prints -1, 8 and zero below each dot.

Changing background colors

Your Color Computer will allow you to change the graphics-screen background and foreground colors. Use the instruction COLOR:

COLOR *foreground,background*

If you don't use the COLOR to specify foreground and background colors in the graphics mode, the computer automatically selects both. It checks which SCREEN color set you are using and then chooses the highest numbered color within that set for the foreground and the lowest numbered color within that set for the background.

For instance, if you are using SCREEN 1,1 the computer will select from buff (5), cyan (6), magenta (7) and orange (8). If you are using SCREEN 1,0 it will choose from green (1), yellow (2), blue (3), red (4).

In other words, if you don't tell it otherwise it will select orange on buff for SCREEN 1,1 or red on green for SCREEN 1,0.

Within the color sets you can change both foreground and background colors. Try this program:

```
10 PMODE 1,1
20 COLOR 3,2
30 PCLS
40 SCREEN 1,0
```

```
50 LINE(20,20)-(50,50),PSET,BF
60 GOTO 60
```

Lines 10 through 40 establish the graphics mode and screen, as before, but with one difference. Line 20 has been added to use the COLOR instruction to change the foreground and background colors.

In the program above we have selected color 3 (blue) for the foreground and color 2 (yellow) for the background. Line 50 draws a color-filled box on the screen. It is blue box (foreground) against a yellow graphics screen (background). Now switch those colors around by changing line 20:

```
20 COLOR 2,3
```

Running the program now results in a yellow box (foreground) against a blue screen (background). Now change lines 20 and 40:

```
20 COLOR 8,7
40 SCREEN 1,1
```

This selects the other color set and chooses colors orange and magenta. The box will be orange (foreground) and the screen will be magenta (background). Now change line 20 one more time:

```
20 COLOR 7,8
```

This switches the color so that the box (foreground) will be magenta and the screen (background) will be orange. Now that's what you call vivid!

Border around the screen

You will note that the border around the screen remains green throughout this run. In the other color set the border would be buff. The computer determines the color set you are using (either SCREEN 1,0 or SCREEN 1,1) and selects the lowest color number in that set for the border color. For SCREEN 1,0 it chooses green (color 1) for the border and for SCREEN 1,1 it takes buff (color 5) for the border.

Just for fun, here's a program to allow the foreground color to overflow the background color. Let's start in SCREEN 1,1 with orange and cyan:

```
10 PMODE 1,1
20 COLOR 8,6
30 PCLS
```

```
40  SCREEN 1,1
50  FOR Y=0 TO 199
60  LINE(0,Y)-(255,Y),PSET,BF
70  NEXT Y
80  GOTO 10
```

That will run over and over until you press BREAK. Now change colors by changing lines 20 and 40:

```
20  COLOR 3,2
40  SCREEN 1,0
```

Get the idea? You have colors 1, 2, 3 and 4 available in color set 0 and colors 5, 6, 7 and 8 available in color set 1.

Circles and Other Shapes

Circles and Other Shapes

If you've been excited by where we've been so far, wait'll you see where we are going!

Remember the pie we cut into eight slices using DRAW, CIRCLE and PAINT? Here's where you find out how those super BASIC words work.

First, let's look at an even more versatile way to draw lines on the graphics screen. The instruction is DRAW and its format is:

DRAW *"line string"*

Remember, back in the beginning, how tedious it was to draw a simple line on the screen. You had to PRINT @ or POKE and, even when we got into the graphics mode, you were stuck in a straight LINE. Well, here's the way around those obstacles. This command lets you draw up, down, right, left, at angles, in different colors, in different sizes, and even blank lines.

Clear out your program memory by typing in NEW and ENTER. Set up the graphics mode with these program lines:

```
10  PMODE 1,1
20  PCLS
30  SCREEN 1,0
```

Now we'll add lines to compare the LINE drawing instruction you already know with the new DRAW statement:

```
40  LINE(80,110)-(180,110),PSET
50  DRAW"C3BM80,90R100"
60  GOTO 60
```

Line 40 uses LINE to create a straight line from X,Y position 80,110 to position 180,110. Line 50 uses DRAW to make a straight line at X,Y position 80,90 and moving right 100 points.

The LINE line is red and the DRAW line is blue.

Let's take a closer look at line 50:

```
50  DRAW"C3BM80,90R100"
```

Remember that the DRAW instruction must have, attached to it, a *string* of specifications enclosed in quotation marks.

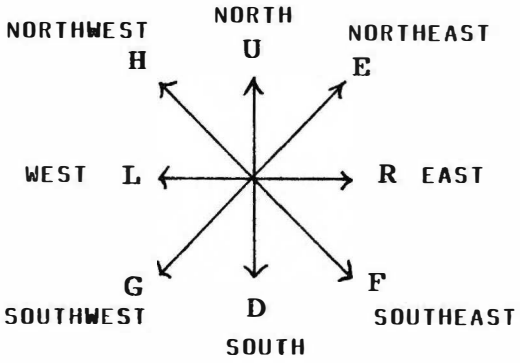
Between the quotes in our line 50 we find, first, a C3. C stands for color. C3 means we want the drawing to be in blue which is color number 3.

Next we find the letters BM. M is the command to start drawing. As a standard procedure, you must always place a B before this M or else you might get unwanted lines on your drawing. So, in effect, BM means "get going!"

But from where? Where do we start drawing? The answer is at X,Y position 80,90. Notice in line 50 that we specified position 80,90 immediately after the letters BM. You must remember to include the comma in the X,Y position at all times so the computer will recognize it as an X,Y location.

So we start at position 80,90. The next part of the string reads R100. This means, having started drawing at position 80,90, draw a line to the right for a distance of 100 screen points. R means draw toward the right and the number 100 indicates how far to go, how many screen points.

There are eight letters, including R, which let you move in any of the directions we think of as being compass points. Imagine moving north, northeast, east, southeast, south, southwest, west, or northwest.



U stands for up, or draw a line in an upward direction. R means draw toward the right. D stands for draw downward and L means draw toward the left.

E tells the computer to draw toward the northeast. That is, upward and toward the right by equal amounts, at a 45 degree angle from north. F says draw down and to the right, southeastward, at an angle of 135 degrees from north.

G means draw down and to the left, southwestward, at an angle of 225 degrees from north. And H stands for draw upward and to the left, northwestward, at an angle of 315 degrees from north.

Here's a list of the various letter abbreviations as used within the DRAW string:

letter	instruction
M	start moving position
B	move but draw blank
U	draw upward (north)
E	draw up and right (northeast)
R	draw rightward (east)
F	draw down and right (southeast)
D	draw downward (south)
G	draw down and left (southwest)
L	draw leftward (west)
H	draw up and left (northwest)
N	no position update
C	color choice
A	angle
S	scale
X	execute substring and return

Suppose you wanted to draw a line 50 points long to the left from the center of the screen. How would you set up a DRAW instruction?

DRAW"C4BM128,96L50"

This statement would cause the computer to draw a red line from the center of the screen toward the left for a distance of 50 points.

C4 calls for color number 4. BM tells the computer to get started with the drawing. 128,96 is the start point. And L50 means draw toward the left for 50 points.

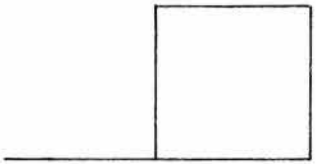
Now let's practice by changing the color in line 50 in our program:

50 DRAW"C2BM80,90R100"

The line will appear yellow. C2 tells the computer to draw using color number 2 which is yellow. Let's set a small box on the end of our line.

50 DRAW"C2BM80,90R100U50L50D50"

The yellow line goes 100 points to the right, then upward for 50 points, then to the left for 50 points, then down for 50 points. The result is a box on the line.



As you can see, you can take the line anywhere on the graphics screen. You can even use the line to make pictures or draw symbols which have meaning. Here's a long DRAW statement which actually creates the letters ABC in the middle of the screen:

```
50 DRAW"C2BM80,90U20R10D10L10R10D10
        BR20U20R10D10L8R8D10L10
        BR40L10U20R10"
```

Try it. You'll find a yellow line starting at X,Y position 80,90 moving upward 20 points, right 10 points, down 10 points, left 10, right 10 and down 10. That creates a letter A.

As the draw continues, it encounters BR20. That

means move right 20 points but don't draw anything. That's how we get to the start position on the letter B without drawing a line connecting the A and B.

To draw the B, we go up 20, right 10, down 10, left 8, right 8, down 10, and left 10. Then there's another BR, this time drawing the invisible or blank line for 40 spaces toward the right.

The C is composed by drawing left 10, up 20, and right 10.

No update

As you look at line 50, you'll see that the computer moves up 20 and then, from that point, moves right 10. And then from that point it moves down 10. And then from there left 10. Etc. The computer updates itself, after each short draw, so it can start the new piece of the drawing from wherever it left off.

You can cause the computer to *not* update by using the letter N before the direction letter.

Suppose you want to simulate the eight-pointed compass rose with eight lines radiating from the center of the compass. You could move the DRAW position out and back each arm of the compass like this:

```
DRAW"C2BM128,96U75D75E75G75R75L75
                F75H75D75U75G75E75
                L75R75H75F75"
```

Or we could shorten that by using the no-update letter, N, and have eight instead of 16 partial-lines to draw:

```
DRAW"C2BM128,96NU75NE75NR75NF75
                ND75NG75NL75NH75"
```

You can see that using N for no update saves plenty of program-memory space here. Type in and run the whole program:

```
10 PMODE 1,1
20 PCLS
30 SCREEN 1,0
40 DRAW"C2BM128,96NU75NE75NR75NF75
                ND75NG75NL75NH75"
50 GOTO 50
```

It draws eight lines, each from the center of the screen. Now use the NEW command to erase that program from memory and let's try +X and +Y.

Relative motion

No, not your mother-in-law chasing you. Relative motion means moving the draw position *relative* to where you last used it.

For instance, suppose you want to draw two boxes on the screen. You know where you want the first one located but the second one must be located relative to the position of the first box. You could spend a lot of time computing the exact X,Y locations for each box. Or you could use the computer's ability to locate *relative* positions.

When we use relative positions, we use points which are located in relation to the last points we used. For example, if you have a point at position 128,96 and you want another point at 120,90 you can see that both the X position and the Y position have changed relative to the first point. The X change is -8 and the Y change is -6.

Having made that change you now are at point 120,90. You want to change again, this time to 145,101. The change in the X direction is +25 and the change in the Y direction is +11.

Suppose we use the DRAW command to draw a yellow box like this:

DRAW"C2BM128,96R25U25L25D25"

This will draw a box starting at point 128,96. The box will be a square with each side 25 points long.

Next you need a box of the same dimensions starting at the upper right-hand corner of the first box. Rather than spend a lot of time calculating the start point of the second box, use the computer's ability to spot a position relative to its last position. Use +X or -X and +Y or -Y. In this example, use X+25 and Y-25. Write that into a DRAW statement like this:

DRAW"C2BM+25,-25R25U25L25D25"

When you ended the first box, your position was back at 128,96. So, in starting the second box, the computer moved X from 128 to a position 25 points in the postive(+) direction. It moved Y 25 points in the negative (-) direction. Then it started drawing the next box. Here's a program you can run:

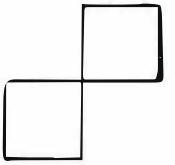
```
10 PMODE 1,1
20 PCLS
```

```
30  SCREEN 1,0
40  DRAW"C2BM128,96R25U25L25D25"
50  FOR L=1 TO 300:NEXT L
60  DRAW"C2BM+25,-25R25U25L25D25"
70  GOTO 70
```

Lines 10 to 30 establish the graphics mode. Line 40 starts at X,Y position 128,96 and draws a yellow square with 25 points on each side.

Line 50 is a momentary time delay so you can see the first box briefly.

Line 60 then draws a second box. The start position for this draw is determined by the +25,-25 in the X,Y places in the string. You must remember to use the comma there as we have done.

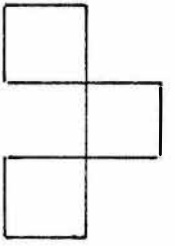


The X must have either a plus (+) sign or a minus (-) sign. Y should have a minus (-) sign or a plus (+) sign. In the case of +Y *only* you have the option of omitting the plus (+) sign.

Let's add some more yellow boxes to the display. Change line 70 and add lines 80 and 90:

```
70  FOR L=1 TO 300:NEXT L
80  DRAW"C2BM-25,-25R25U25L25D25"
90  GOTO 90
```

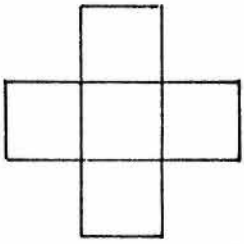
Now the program draws a third box using the position -25,-25 relative to where the second box has ended.



Can you figure out how to draw a fourth box to complete a cross shape? Change line 90 and add lines 100 and 110:

```
 90 FOR L = 1 TO 300:NEXT L
100 DRAW"C2BM-25, + 25R25U25L25D25"
110 GOTO 110
```

The program draws the fourth box, creating an interesting overall shape on the screen. Note that lines 50, 70 and 90 are time delays so you can see one box being created at a time.



To review the entire program, as modified, lines 10 to 30 establish the graphics mode. Line 40 draws the first box. Line 60 draws the second box. Line 80 draws the third and line 100 draws the fourth box. Now let's put some color in the boxes.

PAINT

You know how to color the lines created by the DRAW instruction. But how can we get a big splash of color to fill an entire box on the screen? Use PAINT which has this format:

PAINT *(X,Y),color, border*

The X,Y location specified inside the parentheses indicates where the PAINTing is to start. *Color* is where you put the number of the color you want to use to fill the box or circle or area. Be sure to use a color number which is in the particular SCREEN color-set you are using.

Border means the color number of the border you wish to PAINT to and stop. Suppose you are using SCREEN color set zero. You are in PMODE 1 which permits use of four colors on the display. The colors available to you are green/yellow/blue/red. The background color is green. You have just drawn a yellow square on that green background. Now you wish to fill in

that square with a solid blue color.

The color number you want to PAINT is 3 (blue) and you want to start at a point inside the square and PAINT in each direction until you hit the yellow (color number 2) line or border. You might have this instruction:

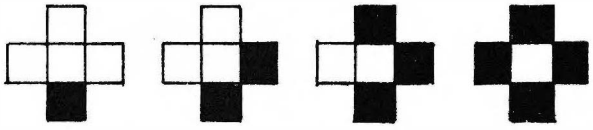
PAINT (130,94),3,2

Starting at X,Y location 130,94 the computer would PAINT blue until it reached a yellow border or until it reached the edge of the graphics screen.

Remember the four boxes we drew with the last program. Let's add some color to those boxes. Change line 110 and add lines 120 through 190.

```
110 PAINT(130,94),3,2
120 FOR L=1 TO 300:NEXT L
130 PAINT(155,69),3,2
140 FOR L=1 TO 300:NEXT L
150 PAINT(130,44),3,2
160 FOR L=1 TO 300:NEXT L
170 PAINT(105,69),3,2
180 FOR L=1 TO 300:NEXT L
190 GOTO 10
```

The program, from line 10 through 190, will DRAW the four yellow boxes as before. Then it will color each box with blue. There is a time delay so you can see each box for a moment after it is colored, before the next box is colored. The time delay loops are lines 120, 140, 160 and 180.



By the way, if the computer is PAINTing toward a border color you have chosen but reaches some other color first, it will PAINT over the unwanted color as it moves on in search of the border you told it to go to.

Be sure to select colors which fit the PMODE number you have selected and the SCREEN color set you are using.

Angles and scale

A stands for *angle* and it has this format:

A *number*

The number must be either 0, 1, 2, or 3. A allows you to specify an angle at which a line is to be drawn. If you don't specify the angle, the computer assumes you want AØ.

Number	Degrees
0	0
1	90
2	180
3	270

The degrees of rotation are in a clockwise direction. If you use an A-number instruction in a DRAW statement, all subsequent lines will be drawn at the same angle or rotation.

The angle specification goes before the BM start-moving instruction. We'll build a program to draw an arrow on the screen and rotate it so it points in different directions:

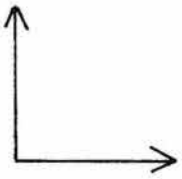
```
10 PMODE 1,1
20 PCLS
30 SCREEN 1,0
40 DRAW"C2A0BM128,96U50NF10NG10"
50 GOTO 50
```

The arrow extends upward 50 points from the center of the screen. The A0 information imbedded in the DRAW string told the computer to rotate the arrow zero degrees from straight up. In other words, in this case, don't rotate it at all.

Let's add another DRAW statement, this time with an A1 angle to rotate the arrow 90 degrees. Change line 50 and add lines 60 and 70:

```
50 FOR L=1 TO 300:NEXT L
60 DRAW"C2A1BM128,96U50NF10NG10"
70 GOTO 70
```

The first arrow still points up from the center of the screen but now there is a second arrow pointing toward the right from the center of the screen. In effect, the second arrow is rotated 90 degrees from the vertical.

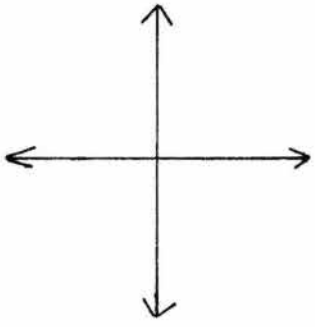


Obviously, we can add arrows so we have them pointing in four directions, north/east/west/south. Change line 70 and add lines 80 through 110:

```
 70 FOR L=1 TO 300:NEXT L
 80 DRAW"C2A2BM128,96U50NF10NG10"
 90 FOR L=1 TO 300:NEXT L
100 DRAW"C2A3BM128,96U50NF10NG10"
110 GOTO 110
```

Now the program works like this: lines 10 through 30 establish the graphics mode. Line 40 DRAWs a vertical arrow, pointing upward. Line 60 creates the rightward-pointing arrow. Line 80 makes the downward-pointing arrow and line 100 DRAWs the leftward-pointing arrow.

Lines 50, 70 and 90 are time delays so you can see the arrows progress around the "clock."



A spinning arrow comes to mind! Erase lines 40 to 110 of the old program and type in these new lines 100 to 230.

```
 10 PMODE 1,1
 20 PCLS
 30 SCREEN 1,0
100 FOR A=0 TO 3
110 A$=STR$(A)
120 GOSUB 200
130 NEXT A
140 GOTO 100
200 DRAW"C2A"+A$+"BM128,96U50NF10NG10"
210 FOR L=1 TO 200:NEXT L
220 PCLS
230 RETURN
```

Rather than writing the DRAW strings over and over, we put the DRAW instruction in a subroutine starting at line 200.

Lines 10 to 30 establish the graphics mode. Lines 100 and 130 create a loop to take the value of A from zero through three. Line 110 converts the current numerical value of A to a string and stores it in A$.

Line 120 forces the computer to jump to the subroutine at line 200.

Pay careful attention to the way we insert the A value into the DRAW statement in line 200. We use the computer's ability to *add* strings together to create longer strings.

```
200 DRAW"C2A"+A$+"BM128,96U50NF10NG10"
```

We actually add three strings together to create the final DRAW string. The first is the obvious "C2A" and the third is "BM128,96U50NF10NG10". The second, or middle, string in the group is A$.

Remember that the value of A$ changes with the FOR/NEXT loop in lines 100 to 130. Whatever the current value is for A$, it is inserted into the DRAW string at line 200 to tell the computer what *angle* number to use.

You could use the same trick to change colors, start locations, scale or other important information in the DRAW string.

Speaking of scale, here's the format for changing size of your drawings:

S *number*

The S-number can range from 1 to 62, indicating the relative scale of size of your drawings in quarter units.

The S-number can range from 1 to 62, indicating the relative scale of size of your drawings in quarter units.

S number	scale
1	1/4 scale
2	2/4 scale
3	3/4 scale
4	4/4 scale
5	5/4 scale
6	6/4 scale
etc.	

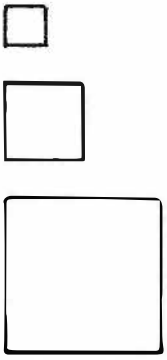
The 2/4 scale is the same as 1/2 scale. The 4/4 scale is full scale or one-to-one scale. The 5/4 scale is the same as 125% of the original. The 8/4 scale is double size. The 12/4 scale is triple size. And so on.

If you don't use an S-number, the computer assumes 4/4 scale or full size. After you use a scale change instruction, all drawings will remain in the increased or decreased size until you issue another S-number.

Here's a short program to draw a box in 4/4 (full size) scale and then repeat it in half size (2/4 scale) and double size (8/4 scale):

```
 10 PMODE 1,1
 20 PCLS
 30 SCREEN 1,0
100 DRAW"C2S2BM128,48R20U20L20D20"
110 DRAW"C3S4BM128,96R20U20L20D20"
120 DRAW"C4S8BM128,144R20U20L20D20"
200 GOTO 200
```

The program creates three boxes on the video screen. A small box is on top; a medium box is in the middle; a large box in on the bottom.



The small box is yellow. The medium box is blue. The large box is red.

All the boxes have the same dimensions on their sides. Each is drawn as R20U20L20D20. But, the yellow box is half-size 2/4 scale. Note the S2 information imbedded in line 100 which draws the yellow box.

The blue box is in full-size 4/4 scale. The specification S4 can be found in the DRAW string in line 110.

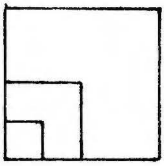
The red box is double size (8/4 scale). The S8 scale direction is in line 120.

Here's a variation on the same program which you might like. Change lines 100 through 120:

```
100 DRAW"C2S2BM128,96R20U20L20D20"
110 DRAW"C3S4BM128,96R20U20L20D20"
120 DRAW"C4S8BM128,96R20U20L20D20"
```

The program still draws three boxes but now each has the same origin. That is, the lower left-hand corner of each box is at the same point onthe screen. You can see how the sizes differ even though each still is described as R20 U20 L20 D20.

The yellow box is smallest. The blue box is medium sized. The red box is largest.



By the way, have you gotten the hang of making things disappear? Draw a line in any color other than the background and it will appear. Draw the same line over top of it in the background color and it will disappear. Here's a quick sample:

```
10 PMODE 1,1
20 PCLS
30 SCREEN 1,0
40 DRAW"C2BM128,96R50NH10NG10"
50 FOR L=1 TO 300:NEXT L
60 DRAW"C1BM128,96R50NH10NG10"
70 FOR L=1 TO 300:NEXT L
80 GOTO 40
```

A run results in an arrow which points rightward from the center of the screen. The arrow appears and disappears and appears and so on. It seems to blink on and off. Lines 10 to 30 establish the graphics mode and screen. Line 40 draws the arrow in yellow. Line 50 is a time delay so you can look at the arrow for a moment.

Line 60 draws the same size and shape arrow in the same place on the screen, but in green, the background color. It disappears into the rest of the green background.

Line 70 is a time delay so you can see the invisible arrow and line 80 pushes back to line 40 where the yellow arrow is drawn again. The blinking arrow continues until you press the BREAK key.

Around in CIRCLEs

So far we've used straight lines, sometimes bent into angles, to create shapes on the display. There is a command which will draw a circle for you.

CIRCLE *(X,Y),radius, color, heighth/width, start, end*

The center of the circle is set at a particular X,Y location on the face of the screen. To draw a circle on the face of the screen, you must at least locate the centerpoint and specify the radius.

Color, height-to-width ratio, start point and endpoint for partial circles all are optional.

```
10 PMODE 1,1
20 PCLS
30 SCREEN 1,0
40 CIRCLE (128,96),50
50 GOTO 50
```

Lines 10 to 30 establish the graphics mode. Line 40 draws a circle with center point at X,Y position 128,96. The circle has a radius equal to 50 points on the face of the graphics screen. The diameter of the circle is 100 points.

The color of the circle is red. Here's how to change that to yellow. Change line 40:

```
40 CIRCLE (128,96),50,2
```

Now the circle comes out drawn in a yellow line. To change it into a tall oval, change line 40 again:

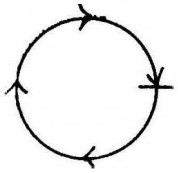
```
40 CIRCLE (128,96),52,2,2
```

To draw only half of the circle, change line 40 again:

```
40 CIRCLE (128,96),50,2,1,.25,.75
```

The possible start point and end point of a semicircle is in the range of zero to one. Here we selected .25 for the start point and .75 for the end point.

Remember that the start point and end point for a full circle is at the 3:00 position. So, when you specify a start at .25 that would be one quarter turn later or 6:00. When you specify an end point at .75 that would be three-quarters turn or 12:00.



Be sure to use the heighth-width ratio when drawing an arc. For a partial circle, use an heighth-width ratio of 1.

About that heighth-width ratio: if it is greater than 1, the circle will be taller than it is wide. If it is less than 1, the circle will be wider than it is tall. If it is 1, you will get a true circle.

If the heighth-width ratio equals zero, the circle becomes a horizontal line. On the other hand, the greater the number becomes above 1, the more the circle approaches a vertical line. Try this program:

```
10  PMODE 1,1
20  PCLS
30  SCREEN 1,0
40  CIRCLE (128,96),100,2,.4
50  GOTO 50
```

The program, at line 40, draws a circle with centerpoint at 128,96. The yellow circle has a radius of 100 points and is squashed. The .4 in line 40 gives it a heighth-width ratio of 4/10. It's 2 1/2 times as wide as it is tall. Sort of like a football or a halo.

Let's switch that to an oval that is taller than it is wide. Change line 40:

```
40  CIRCLE (128,96),15,2,5
```

Now the circle becomes very tall and narrow. You could make it so tall and narrow it would become a vertical line with this change to line 40:

```
40  CIRCLE (128,96),2,2,255
```

Or you could make it so short it would become a horizontal line with this change to line 40:

```
40  CIRCLE (128,96),15,2,0
```

If you want normal-shape, full circles you may omit the helghth-width ratio and the start point/end point numbers. Also, if you are satisfied with the current foreground color, you may eliminate the color number.

But, you must always have the X,Y center point and the radius. The largest radius which will fit on the screen is 95 and that requires the circle to have a center point at 128,96. Larger radii will cause part of the circle to flatten out against the edge of the graphics screen.

Ragged circles

Have you been a bit less than happy with the rough edges on the circles we have drawn so far? It's because we are in a medium resolution mode. Try this circle in the highest resolution, PMODE 4:

```
10  PMODE 4,1
20  PCLS
30  SCREEN 1,0
40  CIRCLE (128,96),50
50  GOTO 50
```

Now you see what the difference is! The points used to create the circle are much smaller in PMODE 4. The circle looks far less ragged and much more like a "real" circle. For further comparison of the resolutions, make this change to line 10:

```
10  PMODE 0,1
```

The difference between the higher and lower graphics resolutions is very noticeable.

Now for some fun. Let's leave the subject of circles with this easy piece of graphic artistry:

```
10  PMODE 1,1
20  PCLS
30  SCREEN 1,0
40  FOR X=25 TO 230 STEP 5
50  CIRCLE (X,96),25,2
60  NEXT X
70  GOTO 70
```

A neat art design of circles across the center of the screen. Can you figure out how to fill the screen with the same sort of pattern?

Making Things Move

Making Things Move

Movement on the computer display screen is an illusion. As in any television picture, the turning on and turning off of dots in a pattern across a screen can seem to provide motion to an object drawn on the face of the tube.

There are a number of ways to get the look of motion. Let's send a dot across the screen:

```
10  PMODE 1,1
20  PCLS
30  SCREEN 1,0
40  FOR X=0 TO 255
50  PSET (X,96)
60  PRESET (X,96)
70  NEXT X
80  GOTO 80
```

We use PSET and PRESET to turn on and off points in a line across the screen. It looks like a dot is moving. We can do the same with the LINE instruction. Change lines 40, 50 and 60:

```
40  FOR X=0 TO 253
50  LINE(X,96)-(X+2,98),PSET,BF
60  LINE(X,96)-(X+2,98),PRESET,BF
```

Again, a dot seems to move rightward across the screen. Let's try it with a CIRCLE. Change lines 50 and 60:

```
50  CIRCLE (X,96),5,2
60  CIRCLE (X,96),5,1
```

In each of these cases we start at the left edge of the graphics screen and create a dot. We then make that dot disappear by coloring it the same as the background. Then we light a new dot or circle and then turn it off by making it the same color as the background. The result is the apparent movement of the dot or circle rightward across the screen.

Give 'er some gas!

You will notice how slowly the dots move across the screen. The process of turning on and turning off and so on takes a lot of time. The computer, after all, only works on one instruction at a time. Even at the high speeds with which events happen inside the computer, it still seems slow in moving that dot across the screen. Here's how to speed things up considerably.

The GET and PUT instructions allow you to grab a picture from one place on the screen and take it directly to another spot on the display. This rapid changing of places is followed by the eye and seen as movement between the two places, even though the object on the screen only appeared in two places: the place where you GET the art and the place where you PUT it. Here's an example:

```
10 PMODE 3,1
20 PCLS
30 SCREEN 1,0
40 DIM X(7,9)
50 DRAW "BM113,191R15E5F5R15H10U30H10
                        NU5GI0D30G10"
60 PAINT(128,146),3,4
70 GET(113,135)-(153,191),X
80 PCLS
90 PUT (113,0)-(153,56),X
100 GOTO 100
```

A blue rocket ship with a red outline flies against a green background! It literally hops from bottom of screen to top. How does it work?

Lines 10 to 30 establish the graphics mode and screen. Note we went to PMODE 3 higher resolution, finer detail.

Line 40 sets up an array labeled X. That array will hold the picture of the rocket ship when the time comes for it to leap upward.

Line 50 draws the ship at the foot of the screen. It starts life as a red outline. Then like 60 PAINTs blue inside the red outline. Now we have a red and blue rocket ship sitting at the bottom of the screen.

The GET instruction in line 70 picks up the entire screen rectangle from X,Y location 113,135 to location 153,191 and stores the picture in that rectangle in the array labeled X which we set up back at line 40.

Position 113,135 is the upper left corner of the rectangle. Position 153,191 is the lower right corner of the screen rectangle. The rectangle, defined by the upper left and lower right corners, holds all of the picture which we want to move. We take the display from that rectangle and store it in memory, in a special array.

Having used GET to store away the display information in an array, we erase the display. The PCLS in line 80 clears the graphics screen. At this point the screen is blank but our original rocket ship is stored in memory in an array labeled X.

Now we accomplish the illusion of motion.

The PUT statement in line 90 recalls the display data from memory array X and places it in a same-size rectangle on the graphics screen, but in a different location from where it got that data.

The computer can GET a part of its display from one set of screen locations (first rectangle) and PUT that part of the display at some other location on the graphics screen.

The format for the GET statement is:

GET *(startpoint)-(endpoint),array,G*

Startpoint is the X,Y coordinates of the upper left corner of the part of the display to be stored in memory. End-

point is the lower right corner of that part of the display to be stored in memory.

Any part of the graphics display may be visualized as being in this rectangle but the larger the area to be picked up, the more memory required to hold it. So, the size of your rectangle is limited by how much memory you can afford to spend on it. Each element in the storage array takes five memory locations or bytes of memory. For instance, a 16K color computer is limited to arrays with fewer than a total of 1400 elements. (Width multiplied times length gives the total.)

The array is a group of memory locations you set up to receive the data from the rectangle on the screen. GET has to have a destination when it picks up the information and this array is that destination. GET takes the information and stores it in memory in this array.

The array is a two-dimensional array. The size of that array corresponds to the size of the screen rectangle which needs to be stored. The first array size is the width of the rectangle. The second array size is the heighth on the screen of the rectangle. For example, a 10 by 20 rectangle would fit in a 10 by 20 array.

Remember that in the Color Computer, array sizes start counting at zero. So a 10 by 20 array could be dimensioned as:

DIM X(9,19)

Such a 10 by 20 array would have a total of 200 elements. The total number of elements is rectangle width multiplied by rectangle heighth.

A 30 by 40 array might be dimensioned as DIM X(29,39) and would have a total of 1200 picture elements. The width of the screen rectangle to be stored (in this case, 30) is the first dimension in the array. The heighth of the rectangle on the screen (in this case, 40) is the second dimension in the array.

Here's the PUT statement format:

PUT*(startpoint)-(endpoint), array, action*

This startpoint and endpoint define a same-size rectangle on the screen where you plan to PUT the display data stored in the memory array.

The startpoint is the upper left corner of a rectangle. The endpoint is the lower right corner of that rectangle.

This rectangle must be the same size, in width and heighth, as the original rectangle obtained by the GET Instruction.

Naturally, the array is the set of memory locations previously used by GET to store the picture away. This array is the source of information for PUT just as it was the destination for GET.

Action is an optional part of the instruction. If you use action with PUT you must use the letter G with GET. The G goes at the end of the GET statement and the action instruction goes at the end of the PUT statement. Action, when used, tells the computer how to create the new display rectangle. The choices for action words are PSET, PRESET, AND, OR and NOT. Here's what they do:

Word	Function
PSET	Turns on same points as in the original rectangle.
PRESET	Turns off the points that were on in the original rectangle.
AND	Compares points in both old and new rectangles. If points are on in both, the screen point will be turned on. If not, turned off.
OR	Compares points. If either rectangle is on, screen remains on.
NOT	Reverses the on/off state of each point in the new rectangle regardless of the PUT array's contents.

It is important that you remember to use both GET and PUT in the same PMODE. And, of course, the rectangle you choose and the screen position you plan for it both must be large enough to hold all of the parts of the picture you want to move. If not, part of your picture will be lost.

If you tack the G on the end of the GET statement, you must use one of the action words on the end of the PUT statement. However, there is no need to use G and the action options in PMODE 0, PMODE 1 or PMODE 3. You should use them in PMODE 2 and PMODE 4.

Smooth flight

In our rocket ship program, we had it hop from one location to another. It was first drawn on the bottom of the screen. We used GET and then PUT to move it from the bottom to the top of the screen. Now let's give it a more smooth flight:

```
10 PMODE 3,1
20 PCLS
30 SCREEN 1,0
40 DIM X(7,9):DIM Z(7,9)
50 DRAW"BM113,191R15E5 F5R15H10U30H10
                        NU5G10D30G10"
60 PAINT(128,146),3,4
70 GET(113,135)-(153,191),X
80 FOR Y=134 TO 0 STEP -1
90 PUT(113,Y)-(153,Y+56),X
100 PUT(113,Y+57)-(153,Y+112),Z
110 NEXT Y
120 GOTO 120
```

In this program we PUT a series of rocket ships up the screen, erasing each old ship when a new one is created. The flow of rocket ships up the screen creates the illusion of smooth flight.

Lines 10 to 30 establish the graphics mode and turn on the graphics screen. We use PMODE 3 for better resolution.

At line 40, we dimension two arrays. The first array, labeled X, will be used to store the actual picture of the rocket ship. The second array, labeled Z, will have nothing placed in it. All of its memory locations will remain empty, or set to zero. We will need that empty array later, at line 100, to erase old rocket ships after new ones are drawn.

Line 50 draws our rocket ship at the bottom of the screen with a red outline. Line 60 colors the rocket ship blue inside the red outline.

Line 70 GETs the image of the rocket ship and background from a rectangle which has its upper left corner at X,Y position 113,135 and its lower right corner at position 153,191. The colors of all the dots in that rectangle are stored in memory in array X.

Lines 80 and 110 compose a FOR/NEXT loop to take the value of Y from 134 to zero (step -1). Each pass through the loop reduces Y by 1. That Y value is used to establish the down-to-up location of the rocket ship when it is printed on the screen by the PUT command in line 90.

Line 90 goes to array X and pulls out the information stored there concerning the color of all the dots in the rectangle being moved. It takes that information and uses

It to recreate the image of the rocket ship and its background in a new rectangle at a new screen location.

Line 100 is very important because it, in effect, erases an old rocket ship at a lower elevation when a new ship is drawn at a higher elevation above the bottom edge of the graphics screen.

Line 100 is a PUT instruction. It looks in the array we labeled Z. Finding nothing there, when it recreates its rectangle, nothing is printed. Only the background color remains. We position this rectangle so it always is immediately below the rectangle holding the picture of the rocket ship. Thus, you have two rectangles moving up the screen. One with a rocket ship drawing. One with nothing. The nothing covers or replaces the older, lower rocket ship, effectively erasing it.

If you leave out line 100, you get a solid color bar from the bottom of the rocket ship back down to the bottom edge of the graphics screen.

Line 120 is an action freezer.

Putting it all together

At this point, if you've been following the sequence of explanation through this book, you're ready for something really big!

Here's a program which will outdo any old video game graphics. This is not a game but an example of how to put together a great deal of the graphics knowledge you now have to create something to zonk the neighborhood kids. Even your family will think this one's cute.

We'll build on the blue rocket ship you already know about. However, to save confusion, better clear out all of your program memory by typing in NEW and pressing the ENTER key. Here's the program:

Ghost Martian
In Flying Saucer
Shoots Down Rocket Ship

```
10    PMODE 3,1
20    PCLS
30    SCREEN 1,0
```

```
100   DIM X(7,9):DIM Z(7,9)
200   DRAW"BM113,191R15E5F5R15H10U30HI0NU5
                          G10D30G10"
210   PAINT(128,146),3,4
300   DRAW"C2BM20,175R10U5L10U5R10"
310   SOUND 225,1
320   FOR L=1 TO 100:NEXT L
330   DRAW"C1BM20,175R10U5L10U5R10"
340   DRAW"C2BM20,175BR10U10D5L5U5"
350   SOUND 225,1
360   FOR  L=1 TO 100:NEXT L
370   DRAW"C1BM20,175BR10U10D5L5U5"
380   DRAW"C2BM20,175R10U5L5R5U5L10"
390   SOUND 225,1
400   FOR L=1 TO 100:NEXT L
410   DRAW"C1BM20,175R10U5L5R5U5L10"
420   DRAW"C2BM20,175R10L10U5R10U5L10"
430   SOUND 225,1
440   FOR L=1 TO 100:NEXT L
450   DRAW"C1BM20,175R10L10U5R10U5L10"
460   DRAW"C2BM20,175BR5U10"
470   SOUND 225,1
480   FOR L=1 TO 100:NEXT L
490   DRAW"C1BM20,175BR5U10"
500   GET(113,135)-(153,191),X
510   FOR Y=134 TO 0 STEP -1
520   PUT(113,Y)-(153,Y+56),X
530   PLAY"L255C"
540   PUT(113,Y+57)-(153,Y+112),Z
550   NEXT Y
600   CIRCLE(40,50),20,2,1,.6,.95
610   SOUND 250,1
620   CIRCLE(40,50),40,2,.2
700   FOR L=0 TO 9
710   LINE (L+60,50)-(L+61,48),PSET,BF
720   NEXT L
730   DRAW"C3BM70,49NU5NE5NR5NF5ND5R10
                          BR5R10BR5R10"
740   PLAY "L10C"
750   DRAW"C2BM116,49NH5NL5NG5"
760   FOR L=1 TO 50:NEXT L
800   PCLS
```

```
810  PLAY "L100F"
820  DRAW"C4BM153,56 D15F5G5D15E10R30E10
                     NR5H10L30H10"
830  PCLS
840  PLAY "L100E"
850  DRAW"C4BM150,120D15F5G5D15E10R30E10
                     NR5H10L30H10"
860  PCLS
870  PLAY "L100D"
900  LINE(0,191)-(255,191),PSET
910  DRAW"C4BM150,191H5U15F10E15D20R5U20
                     F35E9F5E5"
920  PAINT(151,190),3,4
930  DRAW"C2BM135,191NE2NU5NH5BR100NH2
                     NU5NE5"
940  PLAY "L1C"
1000 IF INKEY$ = ""THEN 1000 ELSE CLEAR:GOTO 10
```

Type in the program and run it. At the end of action and sound, the computer will hold the last image on the screen and await instructions to start over. You press any key on the keyboard and the action will start over from the beginning. Here's how it works:

We've divided the program lines into 10 blocks so you can more readily analyze the structure. The lines from 10 to 30 establish the graphics mode and turn on the graphics screen. We choose PMODE 3 and the green/yellow/blue/red color set.

Line 100 sets up two memory arrays. The array labeled X will hold dot color information from the rectangle containing the rocket ship. The Z array will hold nothing and is used to erase the rocket ship from the screen as it flies upward.

Lines 200 and 210 create the rocket ship at the bottom of the screen. Line 200 draws it in red outline and line 210 fills it in with blue color.

What good is a rocket ship without a countdown to launch? Lines 300 to 490 make the countdown. We do that by drawing on the graphics screen the numbers 5, 4, 3, 2 and 1 in sequence with an appropriate beep after each number appears.

Line 300 draws a yellow number 5. Line 310 beeps. Line 320 holds the 5 on the screen for a moment so you

can see it. Then line 330 erases the number by drawing the same 5 in the same location in green. The green 5 can't be seen, since green is the overall background color, so the number disappears.

The cycle is repeated as we draw a 4 at the same screen location. Line 340 draws a yellow 4. Line 350 beeps. Line 360 pauses. Line 370 erases the 4 by drawing a green 4.

Line 380 draws a yellow 3 and line 410 erases it. Line 420 draws a 2 and line 450 erases it. Line 460 draws a 1 and line 490 erases it.

Couldn't the number of program lines from 300 to 490 be reduced by using subroutines? Yes! We've strung things out just a bit so the logical sequence of operations would be more clear.

Program lines 500 to 550 fly the rocket ship from the bottom of the screen to the top, with sound effects. Line 500 gets the dot color information for the rectangle containing the rocket ship and stores that data in memory array X.

The FOR/NEXT loop in lines 510 and 550 cause the PUT statement in line 520 to move the rocket ship picture up the Y axis one notch at a time. At the same time, the "nothing" stored in array Z is used by line 540 to erase the lower rocket ship pictures. This is the same trick we used earlier to make the rocket seem to fly smoothly up the screen.

As the rocket reaches the top of the screen, a ghostly martian flying saucer suddenly appears from nowhere and fires a laser beam at the rocket, hitting the ship in the tail section. Lines 600 to 620 draw the flying saucer outline, with a sound effect.

Lines 700 to 760 create the laser cannon on the side of the saucer, the bolt of laser light in a line across to the rocket, and the effect of the hit on the rocket's tail. Line 760 freezes the action momentarily so you can see what has happened.

The rocket ship is damaged and crashes. Lines 800 to 870 are the falling ship. Here, we simply repeat the rocket shape twice, at lines 820 and 850 to show it on the way down. Again, we take the long way to show how the program works. You may be able to shorten the program here

by using a FOR/NEXT loop to display the falling ship.

There must be some ground for the rocket to go "splat" on. Lines 900 to 940 create that and display the downed ship. Line 900 draws a ground line. Line 910 draws the broken rocket ship on the ground in red outline and line 920 colors it in blue. Line 930 shows dust kicked up by the rocket hitting the ground.

Finally, after everything has come to rest and the music generated by line 940 has ended, the computer awaits your key press. Line 1000 loops back to itself forever until you press one of the keyboard keys. Press BREAK to end the run or press any other key to repeat the flight and crash.

That's a lot of action in only 56 program lines. And, using some ingenuity in trimming program lines by using subroutines and FOR/NEXT loops, you might be able to reduce the overall program to fewer than 50 lines. That's a lot of graphic animation and sound in a small package!

It demonstrates the extreme power in the TRS-80 Color Computer's microprocessor and Extended Color BASIC language. Words like DRAW, LINE, PAINT, PLAY, SOUND, GET, PUT, CIRCLE, PMODE, PCLS, SCREEN, DIM, LINE and INKEY$ set the color computer apart from most others. These words gain their power from their ability to combine in one BASIC word, instructions which might require as much as 10 or 20 program lines in other computers.

Color Graphics Programs

Color Graphics Programs

This section includes a wide range of different color graphics programs for your enjoyment. These programs are designed to demonstrate the various BASIC words used to create unusual screen displays and sounds. All are short and easy to type in and run. You should try each one. You'll like what you find.

Canadian Maple Leaf

Canadians are proud of their national symbol, the maple leaf. Let's see what it takes to draw one on the color computer video screen:

```
10 DATA 1552,1584,1613,1615,1616,1617,1619
20 DATA 1645,1646,1647,1648,1649,1650,1651
30 DATA 1677,1678,1679,1680,1681,1682,1683
40 DATA 1709,1710,1711,1712,1713,1714,1715
50 DATA 1737,1741,1742,1743,1744,1745
60 DATA 1746,1747,1751
70 DATA 1766,1767,1768,1769,1770,1773,1774
80 DATA 1775,1776,1777,1778,1779
90 DATA 1782,1783,1784,1785,1786
100 DATA 1798,1799,1800,1801,1802,1803
110 DATA 1805,1806,1807,1808,1809,1810,1811
120 DATA 1813,1814,1815,1816,1817,1818
130 DATA 1830,1831,1832,1833,1834,1835
140 DATA 1836,1837,1838,1839,1840,1841,1842
150 DATA 1843,1844,1845,1846,1847,1848,1849,1850
160 DATA 1861,1862,1863,1864,1865,1866,1867
170 DATA 1868,1869,1870,1871,1872,1873,1874,1875
180 DATA 1876,1877,1878,1879,1880,1881,1882,1883
190 DATA 1894,1895,1896,1897,1898,1899,1900
200 DATA 1901,1902,1903,1904,1905,1906,1907
210 DATA 1908,1909,1910,1911,1912,1913,1914
220 DATA 1927,1928,1929,1930,1931,1932,1933
230 DATA 1934,1935,1936,1937,1938,1939
240 DATA 1940,1941,1942,1943,1944,1945
250 DATA 1960,1961,1962,1963,1964,1965,1966
260 DATA 1967,1968,1969,1970,1971
270 DATA 1972,1973,1974,1975,1976
280 DATA 1993,1994,1995,1996,1997,1998,1999
290 DATA 2000,2001,2002,2003
300 DATA 2004,2005,2006,2007
310 DATA 2026,2027,2028,2029,2030,2031,2032
320 DATA 2033,2034,2035,2036,2037,2038
330 DATA 2057,2058,2059,2060,2061,2062,2063
340 DATA 2064,2065,2066,2067
350 DATA 2068,2069,2070,2071
360 DATA 2088,2089,2090,2091,2092,2093
```

```
370 DATA 2094,2095,2096,2097,2098,2099
380 DATA 2100,2101,2102,2103,2104
390 DATA 2128,2160,2192,2224,2256,2288
400 PMODE 1,1:PCLS:SCREEN 1,0
420 FOR Z=1 TO 240
430 READ L
440 POKE L,255
450 NEXT Z
460 RESTORE
490 GOTO 420
```

Let's change things a bit. Here's the same solid-color maple leaf but now you can control its position on the screen by pressing the up-arrow or down-arrow keys on the keyboard. The leaf stays in the middle of the screen until you press one of those arrow keys. It moves up when you press the up-arrow key and down when you press the down-arrow key.

```
10 DATA 1552,1584,1613,1615,1616,1617,1619
20 DATA 1645,1646,1647,1648,1649,1650,1651
30 DATA 1677,1678,1679,1680,1681,1682,1683
40 DATA 1709,1710,1711,1712,1713,1714,1715
50 DATA 1737,1741,1742,1743,1744,1745
60 DATA 1746,1747,1751
70 DATA 1766,1767,1768,1769,1770,1773,1774
80 DATA 1775,1776,1777,1778,1779
90 DATA 1782,1783,1784,1785,1786
100 DATA 1798,1799,1800,1801,1802,1803
110 DATA 1805,1806,1807,1808,1809,1810,1811
120 DATA 1813,1814,1815,1816,1817,1818
130 DATA 1830,1831,1832,1833,1834,1835
140 DATA 1836,1837,1838,1839,1840,1841,1842
150 DATA 1843,1844,1845,1846,1847,1848,1849,1850
160 DATA 1861,1862,1863,1864,1865,1866,1867
170 DATA 1868,1869,1870,1871,1872,1873,1874,1875
180 DATA 1876,1877,1878,1879,1880,1881,1882,1883
190 DATA 1894,1895,1896,1897,1898,1899,1900
200 DATA 1901,1902,1903,1904,1905,1906,1907
210 DATA 1908,1909,1910,1911,1912,1913,1914
220 DATA 1927,1928,1929,1930,1931,1932,1933
230 DATA 1934,1935,1936,1937,1938,1939
240 DATA 1940,1941,1942,1943,1944,1945
```

```
250 DATA 1960,1961,1962,1963,1964,1965,1966
260 DATA 1967,1968,1969,1970,1971
270 DATA 1972,1973,1974,1975,1976
280 DATA 1993,1994,1995,1996,1997,1998,1999
290 DATA 2000,2001,2002,2003
300 DATA 2004,2005,2006,2007
310 DATA 2026,2027,2028,2029,2030,2031,2032
320 DATA 2033,2034,2035,2036,2037,2038
330 DATA 2057,2058,2059,2060,2061,2062,2063
340 DATA 2064,2065,2066,2067
350 DATA 2068,2069,2070,2071
360 DATA 2088,2089,2090,2091,2092,2093
370 DATA 2094,2095,2096,2097,2098,2099
380 DATA 2100,2101,2102,2103,2104
390 DATA 2128,2160,2192,2224,2256,2288
400 PMODE 1,1:PCLS:SCREEN 1,0
420 FOR Z=1 TO 240
430 READ L
440 POKE L+P,255
450 NEXT Z
460 RESTORE
500 KY$=INKEY$
510 IF KY$="" THEN 500
520 IF ASC(KY$)=94 THEN P=P-128:GOTO 550
530 IF ASC(KY$)=10 THEN P=P+128:GOTO 550
540 GOTO 500
550 SOUND 1,1:PCLS:GOTO 420
```

Here's our very finest Canadian Maple Leaf! With this program you have true graphics artistry. There is an ever-changing pattern and color in the leaf:

```
10 DATA 1552,1584,1613,1615,1616,1617,1619
20 DATA 1645,1646,1647,1648,1649,1650,1651
30 DATA 1677,1678,1679,1680,1681,1682,1683
40 DATA 1709,1710,1711,1712,1713,1714,1715
50 DATA 1737,1741,1742,1743,1744,1745
60 DATA 1746,1747,1751
70 DATA 1766,1767,1768,1769,1770,1773,1774
80 DATA 1775,1776,1777,1778,1779
90 DATA 1782,1783,1784,1785,1786
100 DATA 1798,1799,1800,1801,1802,1803
```

```
110 DATA 1805,1806,1807,1808,1809,1810,1811
120 DATA 1813,1814,1815,1816,1817,1818
130 DATA 1830,1831,1832,1833,1834,1835
140 DATA 1836,1837,1838,1839,1840,1841,1842
150 DATA 1843,1844,1845,1846,1847,1848,1849,1850
160 DATA 1861,1862,1863,1864,1865,1866,1867
170 DATA 1868,1869,1870,1871,1872,1873,1874,1875
180 DATA 1876,1877,1878,1879,1880,1881,1882,1883
190 DATA 1894,1895,1896,1897,1898,1899,1900
200 DATA 1901,1902,1903,1904,1905,1906,1907
210 DATA 1908,1909,1910,1911,1912,1913,1914
220 DATA 1927,1928,1929,1930,1931,1932,1933
230 DATA 1934,1935,1936,1937,1938,1939
240 DATA 1940,1941,1942,1943,1944,1945
250 DATA 1960,1961,1962,1963,1964,1965,1966
260 DATA 1967,1968,1969,1970,1971
270 DATA 1972,1973,1974,1975,1976
280 DATA 1993,1994,1995,1996,1997,1998,1999
290 DATA 2000,2001,2002,2003
300 DATA 2004,2005,2006,2007
310 DATA 2026,2027,2028,2029,2030,2031,2032
320 DATA 2033,2034,2035,2036,2037,2038
330 DATA 2057,2058,2059,2060,2061,2062,2063
340 DATA 2064,2065,2066,2067
350 DATA 2068,2069,2070,2071
360 DATA 2088,2089,2090,2091,2092,2093
370 DATA 2094,2095,2096,2097,2098,2099
380 DATA 2100,2101,2102,2103,2104
390 DATA 2128,2160,2192,2224,2256,2288
400 PMODE 1,1:PCLS:SCREEN 1,0
410 FOR C=1 TO 255
420 FOR Z=1 TO 240
430 READ L
440 POKE(L+512),C
450 NEXT Z
460 RESTORE
470 SOUND 1,1
480 NEXT C
490 GOTO 410
```

Even perfection can be boring so let's change the leaf into something else. Type in our first maple leaf program

and look at its non-changing color and non-moving image. Now edit line 420 to change the number 240 to 117. The resulting picture, after a RUN, will look like a castle sitting on ground or a tramp steamer plying the high seas.

```
420 FOR Z=1 TO 117
```

You can keep the leaf/castle/boat the same color but change the background to yellow. Try this change to line 400:

```
400 PMODE 1,1:PCLS 2:SCREEN 1,0
```

Are The Stars Out Tonight?

```
10 PMODE 1,1
20 PCLS
30 SCREEN 1,1
40 FOR X=1 TO 8
50 Y=RND(8)
60 Z=RND(256):W=RND(192)
70 CIRCLE(Z,W),X,Y
80 NEXT X
90 GOTO 30
```

Rainbow Popcorn Strings

```
10 PMODE 1,1
20 PCLS
30 SCREEN 1,1
40 Y=RND(8)
50 CIRCLE(Z,W),6,Y
60 IF PEEK(341)=247 THEN W=W-1
70 IF W<0 THEN W=0
80 IF PEEK(342)=247 THEN W=W+1
90 IF W>191 THEN W=191
100 IF PEEK(343)=247 THEN Z=Z-1
110 IFZ<0 THEN Z=0
```

```
120 IF PEEK(344)=247 THEN Z=Z+1
130 IF Z>255 THEN Z=255
140 IF PEEK(345)=247 THEN PCLS
150 GOTO 30
920 '
930 '
940 '
950 'CIRCLE(Z,W),X,Y
960 'X MAX = 96 (0 TO 95)
970 'Y MAX = 8 (0 TO 8)
980 'W MAX = 192 (0 TO 191)
990 'Z MAX = 256 (0 TO 255)
```

The Tunnel

A mysterious, dark, long tunnel, as deep as your eye can see. Bats, mice, rats, hidden doors.

Press B and a bat flies out toward you. Press M and a mouse runs. Press R and a rat runs. Press D and a hidden door opens and bats fly out. Press C and the door closes. Roll your own fun. Imagine whatever you think might happen in such a deep, dark tunnel and then make up a game to go along with this video artform. The combinations of possibilities are limited only by your imagination.

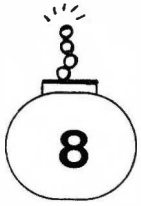
```
 10 PMODE 2,1
 20 PCLS
 30 SCREEN 1,0
 40 FOR A=1 TO 115 STEP 2
 50 CIRCLE(128,96),A,1,.60
 60 NEXT A
 70 DRAW"BM43,187U20L6R11" 'T
 80 DRAW"BM58,187U20D10R10U10D20" 'H
 90 DRAW"BM78,187R10L10U10R10L10
    U10R10" 'E
100 DRAW"BM103,187U20L6R11" 'T
110 DRAW"BM118,187U20D20R10U20" 'U
120 DRAW"BM138,187U20F20U20" 'N
130 DRAW"BM163,187U20F20U20" 'N
```

```
140 DRAW"BM188,187R10L10U10R10L10
    U10R10" 'E
150 DRAW"BM208,187R10L10U20" 'L
160 A$=INKEY$
170 IF A$="" THEN 160
180 IF A$="B" THEN 300
190 IF A$="M" THEN 400
200 IF A$="R" THEN 500
210 IF A$="D" THEN 600
220 IF A$="C" THEN 800
230 GOTO 160
300 FOR L=4 TO 22 STEP 2 'FLYING BAT
310 S$=STR$(L)
320 DRAW"S"+S$+"CØBM128,96E5F5E5F5"
    'DRAWS BAT
330 DRAW"S"+S$+"C1BM128,96E5F5E5F5"
    'ERASES BAT
340 DRAW"S"+S$"+CØBM128,96F5E5F5E5"
    'WING FLAPS
350 DRAW"S"+S$+"C1BM128,96F5E5F5E5"
    'ERASES FLAP
360 NEXT L
370 GOTO 160
400 FOR L=200 TO 61 STEP -5  'MOUSE MOVES
410 M$=STR$(L)
420 DRAW"S4CØBM"+M$+",140H5G5F5E5"
    'DRAWS BODY
430 PAINT(L-2,140),Ø,Ø
440 DRAW"C1BM+M$+",140H5G5F5E5"
    'ERASES BODY
450 PAINT(L-2,140),1,1
460 NEXT L
470 GOTO 160
500 FOR L=61 TO 200 STEP 10 'RAT MOVES
510 R$=STR$(L)
520 DRAW"S4CØBM"+R$+",140H5G5F5E5"
    'DRAWS BODY
530 PAINT(L-2,140),Ø,Ø
540 DRAW"C1BM"+R$+",140H5G5F5E5"
    'ERASES BODY
550 PAINT(L-2,140),1,1
```

```
560 NEXT L
570 GOTO 160
600 LINE(42,93)-(80,130),PRESET,BF   'DOOR
610 FOR JJ=1 TO 125:NEXT JJ
620 DRAW"S4C1BMBM70,113E5F5E5F5"   'DRAWS
    BAT
630 FOR JJ=1 TO 125:NEXT JJ
640 DRAW"CØBM70,113E5F5E5F5"
    'ERASES BAT
650 DRAW"C1BM55,113F5E5F5E5"
    'WINGS FLAP
660 FOR JJ=1 TO 125:NEXT JJ
670 DRAW"CØBM55,113F5E5F5E5"
    'ERASES FLAP
680 DRAW"C1BM45,113E5F5" 'DRAWS PART
690 FOR JJ=1 TO 125:NEXT JJ
700 DRAW"CØBM45,113E5F5" 'ERASES PART
710 GOTO 160
800 PAINT(62,95),1,1   'DOOR CLOSES
810 GOTO 160
```

Yellow Time Bomb Explodes

Yes, it's yellow. And, yes, it's a time bomb. But it has one thing different: a built-in digital timer. You see 10 seconds count down and then the bomb explodes like a supernova. Yes, the fuse does flicker!




```
 10 PMODE 3,1
 20 PCLS
 30 SCREEN 1,Ø
 40 CIRCLE(190,130),50,4,.6
 50 PAINT(190,108),2,4
 60 DRAW "BM190,102R15U10L30D10R15"
 70 PAINT(190,100),2,4
 80 CIRCLE(190,89),6,4.6
```

```
 90 SOUND 200,1
100 PAINT(190,89),2,4
110 CIRCLE(192,84),6,4,.6
120 SOUND 205,1
130 PAINT(192,84),2,4
140 CIRCLE(194,79),6,4,.6
150 SOUND 210,1
160 PAINT(194,79),2,4
170 CIRCLE(192,74),6,4,.6
180 SOUND 215,1
190 PAINT(192,74),2,4
200 FOR LL=1 TO 5
210 DRAW "C2BM192,70NL5NH5NU5NE5NR5"
220 FOR L=1 TO 150:NEXT L
230 DRAW "C4BM192,70NL5NH5NU5NE5NR5"
240 FOR L=1 TO 150:NEXT L
250 NEXT LL
260 DRAW "C2BM192,70NL5NH5NU5NE5NR5"
270 SOUND 200,1
280 DRAW "C4BM180,140U20BR10D20R10
    U20L10"'TEN FOR L=1 TO 500:NEXT L
300 DRAW "C2BM180,140U20BR10D20R10U20L10"
    'ERASE TEN
310 SOUND 180,1
320 DRAW "C4BM195,140U20L10D10R10"'NINE
330 FOR L=1 TO 500:NEXT L
340 DRAW "C2BM195,140U20L10D10R10"'
    ERASE NINE
350 SOUND 160,1
360 DRAW "C4BM195,140U20L10D10R10L10
    D10R10"'EIGHT
370 FOR L=1 TO 500:NEXT L
380 DRAW "C2BM195,140U20L10D10R10L10
    D10R10"'ERASE EIGHT
390 SOUND 140,1
400 DRAW "C4BM195,140U20L10"'SEVEN
410 FOR L=1 TO 500:NEXT L
420 DRAW "C2BM195,140U20L10"'
    ERASE SEVEN
430 SOUND 120,1
440 DRAW "C4BM185,140R10U10L10U10D20"'
    SIX
```

```
450 FOR L=1 TO 500:NEXT L
460 DRAW "C2BM185,140R10U10L10U10
    D20"'ERASE SIX
470 SOUND 100,1
480 DRAW "C4BM185,140R10U12L10U8R10"'
    FIVE
490 FOR L=1 TO 500:NEXT L
500 DRAW "C2BM185,140R10U12L10U8R10"'
    ERASE FIVE
510 SOUND 90,1
520 DRAW "C4BM195,140U20D10L10U10"'FOUR
530 FOR L=1 TO 500:NEXT L
540 DRAW "C2BM195,140U20D10L10U10"'ERASE
    FOUR
550 SOUND 80,1
560 DRAW "C4BM195,140L10R10U10L10R10
    U10L10"'THREE
570 FOR L=1 TO 500:NEXT L
580 DRAW "C2BM195,140L10R10U10L10R10
    U10L10"'ERASE THREE
590 SOUND 60,1
600 DRAW "C4BM195,140L10U10R10U10L10"
    'TWO
610 FOR L=1 TO 500:NEXT L
620 DRAW "C2BM195,140L10U10R10U10L10"'
    ERASE TWO
630 SOUND 40,1
640 DRAW "C4BM190,140U20"'ONE
650 FOR L=1 TO 500:NEXT L
660 DRAW "C2BM190,140U20"'ERASE ONE
670 SOUND 20,15
680 PCLS
690 SCREEN 1,1
700 FOR R=1 TO 500
710 CIRCLE(195,130),R,4
720 NEXT R
730 GOTO 730
```

Spring Art

Here's a very, very nifty way to draw sketches with

lines which look for all the world like coiled springs. Using this sketching program is a lot like the popular child's erasable tablet for sketching. Draw designs, letters, and tricky-looking things on the screen.

The spring starts in the center of the screen but you can move its start point. Press space bar to have it start in the upper left-hand corner of the video display. Press the letter X key on your computer's keyboard to make it start at the upper right-hand corner. Press Y for lower right-hand corner and Z for lower left-hand corner. Press two arrow keys at the same time to make it move at special angles or in curves. Move the tips of the spring by pressing up, down, right or left arrow keys on the keyboard.

```
 10 PMODE 1,1
 20 PCLS
 30 SCREEN 1,0
 40 X=128:Y=96:M=247
 50 IF Y<1 THEN Y=1
 60 IF Y>191 THEN Y=191
 70 IF X<1 THEN X=1
 80 IF X>255 THEN X=255
 90 CIRCLE(X,Y),10,4
100 PAINT(X,Y),4,4
110 CIRCLE(A,B),10,1
130 IF PEEK(341)=M THEN A=X:B=Y:Y=Y-5
    :GOTO 50
140 IF PEEK(342)=M THEN A=X:B=Y:Y=Y+5
    :GOTO 50
150 IF PEEK(343)=M THEN B=Y:A=X:X=X-5
    :GOTO 50
160 IF PEEK(344)=M THEN B=Y:A=X:X=X+5
    :GOTO 50
170 IF PEEK(345)=M THEN PCLS:X=10:Y=10
    :GOTO 50
180 IF PEEK(338)=M THEN PCLS:X=245:Y=10
    :GOTO 50
190 IF PEEK(339)=M THEN PCLS:X=245:Y=181
    :GOTO 50
200 IF PEEK(340)=M THEN PCLS:X=10:Y=181
    :GOTO 50
```

```
210 GOTO 130
999 END
```

Kaleidoscope

Remember those round tubes you looked into? The colors danced and changed as you looked. Here's an electronic version.

```
 10 PMODE 1,1
 20 PCLS
 30 SCREEN 1,Ø
100 CIRCLE(128,96),80,4
110 DRAW"C4BM128,96NU80NE55NR80
    NF55ND80NG55NL80NH55"
200 A=RND(4):B=RND(4)
210 C=RND(4):D=RND(4)
220 E=RND(4):F=RND(4)
230 G=RND(4):H=RND(4)
300 PAINT(130,89),A,4
310 PAINT(140,94),B,4
320 PAINT(140,98),C,4
330 PAINT(130,101),D,4
340 PAINT(126,101),E,4
350 PAINT(116,98),F,4
360 PAINT(116,94),G,4
370 PAINT(126,89),H,4
400 P=P+PPOINT(130,89)
410 P=P+PPOINT(140,94)
420 P=P+PPOINT(140,98)
430 P=P+PPOINT(130,101)
440 P=P+PPOINT(126,101)
450 P=P+PPOINT(116,98)
460 P=P+PPOINT(116,94)
470 P=P+PPOINT(126,89)
500 IF P=32 THEN P=Ø:GOTO 20
510 P=Ø
520 GOTO 200
```

Porthole

Very cute. Very colorful. There's a porthole with blue water visible outside. Fish swim by. Press any key to close the porthole. Press any key to open it. Very neat!

```
 10 PMODE 1,1
 20 PCLS
 30 SCREEN 1,Ø
 40 FOR L=1 TO 1000000
 50 CIRCLE(100,96),40,2
 60 PAINT(100,134),3,2
 70 LINE(50,46)-(150,146),PSET,B
 80 PAINT(155,195),2,4
 90 PAINT(145,140),4,2
100 DRAW"C4BM25,176U20R10D10L10" 'P
110 DRAW"C4BM45,176U20R10D20L10" 'O
120 DRAW"C4BM65,176U20R10D10L10F10" 'R
130 DRAW"C4BM90,176U20L6R11" 'T
140 DRAW"C4BM105,176U20D10R10U10D20" 'H
150 DRAW"C4BM125,176U20R10D20L10" 'O
160 DRAW"C4BM145,176R10L10U20" 'L
170 DRAW"C4BM165,176R10L10U10R10L10
    U10R10" 'E
180 DRAW"C2BM125,96H12G8F8E12"
    'FISH
190 FOR LL=1 TO 125:NEXT LL
200 DRAW"C3BM125,96H12G8F8E12"
    'ERASE FISH
210 DRAW"C2BM105,96H12G8F8E12" 'FISH
220 FOR LL=1 TO 125:NEXT LL
230 DRAW"C3BM105,96H12G8F8E12" 'ERASE FISH
240 DRAW"C2BM70,96H4E4" 'FISH TAIL
250 FOR LL=1 TO 125:NEXT LL
260 DRAW"C3BM70,96H4E4" 'ERASE FISH TAIL
270 A$=INKEY$
280 IF A$="" THEN NEXT L
290 PAINT(100,58),4,2
300 B$=INKEY$
310 IF B$="" THEN 300
320 GOTO 40
```

Red-n-Green Muncher

A fat red-and-green Muncher. When you type in this program and RUN it, you'll find Muncher burping quietly in the upper left-hand corner of his cage. He's just sitting there, fat and happy having recently eaten all the little Yellow Squeekies you fed him.

You can take Muncher for a walk around his cage—a very large room actually—by pressing the arrow keys on the keyboard. Press the key with the arrow pointing the direction you want Muncher to move.

All that exercise will make Muncher hungry again so you must prepare to feed him. When you lift the feeding door—by pressing the space bar on the computer keyboard—the Muncher's favorite food will run in. Now, press the space bar. First one, then two, then four, soon a dozen Yellow Squeekies appear all over the screen. The longer you hold down the space bar, the more Squeekies you give to Muncher.

Notice how the Squeekies attack Muncher and cart off parts of his body. No need to worry. Press any arrow key to make Muncher move away from the Squeekies and you'll see his whole body again.

When you have enough Squeekies in Muncher's cage, release the space bar. You will have made Muncher very happy. He always likes to have a fresh supply of Squeekies to chase around the room. Use the various arrow keys to drive Muncher around his cage, cleaning up the remaining Yellow Squeekies.

By the way, you'll notice Muncher can't get out of his cage. He beeps a lot when bouncing against the edges but he can't get out. And, speaking of beeping, Muncher honks while driving around his cage and the Yellow Squeekies do...well, they squeek.

Pressing any other keys during the game won't result in anything. Except when you want to quit the game. Then press the BREAK key. A great game to teach your preschoolers colors, sounds, directions.

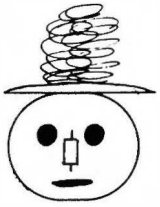
```
10 CLEAR:CLS
20 P=P+A
30 IF P=-1 THEN SOUND 150,1:P=0
```

```
 40 IFP<0 THEN SOUND 150,1:P=P+32
 50 IF P=412 THEN SOUND 150,1:P=411
 60 IF P>411 THEN SOUND 150,1:P=P-32
 70 FOR PX=29 TO 381 STEP 32
 80 IF P=PX THEN SOUND 150,1:P=PX-1
 90 NEXT PX
100 FOR PY=31 TO 383 STEP 32
110 IF P=PY THEN SOUND 150,1:P=PY+1
120 NEXT PY
130 PRINT @ P,STRING$(4,182)
140 PRINT @ P+32,STRING$(4,182)
150 PRINT @ P+64,STRING$(4,182)
160 IF PEEK(341)=247 THEN 400
170 IF PEEK(342)=247 THEN 500
180 IF PEEK(343)=247 THEN 600
190 IF PEEK(344)=247 THEN 300
200 IF PEEK(345)=247 THEN 700
210 GOTO 160
300 PRINT @ P,STRING$(4,143)
310 PRINT @ P+32,STRING$(4,143)
320 PRINT @ P+64,STRING$(4,143)
330 SOUND 1,1
340 A=1
350 GOTO 20
400 PRINT @ P,STRING$(4,143)
410 PRINT @ P+32,STRING$(4,143)
420 PRINT @ P+64,STRING$(4,143)
430 SOUND 1,1
440 A=-32
450 GOTO 20
500 PRINT @ P,STRING$(4,143)
510 PRINT @ P+32,STRING$(4,143)
520 PRINT @ P+64,STRING$(4,143)
530 SOUND 1,1
540 A=32
550 GOTO 20
600 PRINT @ P,STRING$(4,143)
610 PRINT @ P+32,STRING$(4,143)
620 PRINT @ P+64,STRING$(4,143)
630 SOUND 1,1
640 A=-1
650 GOTO 20
```

```
700 RS=RND(411)
710 PRINT @ RS,CHR$(159)
720 SOUND 235,1
730 GOTO 160
```

Blinking Man

He blinks his eye, closes his eye, smiles, frowns. And he wears an interesting hat.



Press C on the keyboard to close the eye. Press O to open it. Press B to make him wink. Press S for a quick smile and press F for a momentary frown.

```
 10 PMODE 3,1
 20 PCLS
 30 SCREEN 1,Ø
 40 CIRCLE(190,130),50,4,.6
 50 CIRCLE(190,99),50,4,.05
 60 PAINT(190,99),2,4
 70 CIRCLE(170,120),9,4
 80 CIRCLE(205,120),9,4
 90 PAINT(170,120),3,4
100 PAINT(205,120),3,4
110 CIRCLE(190,130),4,4,3
120 CIRCLE(190,146),25,4,.1
130 PAINT(190,146),4,4
140 CIRCLE(188,95),25,4,.2
150 PAINT(188,95),2,4
160 CIRCLE(188,90),25,4,.2
170 PAINT(188,90),2,4
180 CIRCLE(186,85),25,4,.2
190 PAINT(186,85),2,4
200 CIRCLE(188,80),25,4,.2
210 PAINT(188,80),2,4
220 CIRCLE(190,75),25,4,.2
230 PAINT(190,75),2,4
240 CIRCLE(192,70),25,4,.2
250 PAINT(192,70),2,4
```

```
260 CIRCLE(194,65),25,4,.2
270 PAINT(194,65),2,4
280 CIRCLE(192,60),25,4,.2
290 PAINT(192,60),2,4
300 CIRCLE(190,55),25,4,.2
310 PAINT(190,55),2,4
320 PAINT(190,135),2,4
330 PAINT(190,155),2,4
340 A$=INKEY$
350 IF A$="" THEN 340
360 IF A$="0" THEN PAINT(205,120),3,4
    :GOTO 340
370 IF A$="C" THEN PAINT(205,120),2,4
    :GOTO 340
380 IF A$="B" THEN PAINT(205,120),2,4
    :FOR L=1 TO 125:NEXT L
    :PAINT(205,120),3,4:GOTO 340
390 IF A$="S" THEN DRAW "C4BM170,144H5"
    :DRAW "C4BM210,144E5":FOR L=1 TO 500
    :NEXT L:DRAW "C2BM170,144H5"
    :DRAW "C2BM210,144E5":GOTO 340
400 IF A$="F" THEN DRAW "C4BM170,144G5"
    :DRAW "C4BM210,144F5":FOR L=1 TO 500
    :NEXT L:DRAW "C2BM170,144G5"
    :DRAW "C2BM210,144F5":GOTO 340
410 GOTO 340
420 GOTO 420
```

Civilization

We like to contemplate the history of mankind and civilization as we watch this artform develop.

```
10 PMODE 1,1
20 PCLS
30 SCREEN 1,Ø
40 X=RND(255):Y=RND(191):C=RND(4)
50 X$=STR$(X):Y$=STR$(Y):C$=STR$(C)
60 DRAW"C"+C$+"BM"+X$+","+Y$+"
   U20R20D20L20"
70 GOTO 40
```

Red Worm

The blob starts in the center of your video screen. As you move it around—at any angle you wish—it inches along like a fat, red worm.

Use the red worm to create new and unusual graphics designs or lettering. Even script.

If you prefer to start in the upper left-hand corner of the screen, press the space bar. The blob will jump to the new start location. Or press the letter X on the keyboard for a start in the upper right-hand corner of the screen. Press Y for the lower-right hand corner. Press Z for the lower left-hand corner. You can push two arrow keys at the same time and use them to move at angles and in curves.

Draw shapes. Draw letters. Write names. Whatever you imagine. The red inchworm on green background is very cute as he slowly moves along. To take your worm for a walk, hold down either the left or right arrow key. Then repeatedly press and release either the up or down arrow key. The worm appears to sway forward.

The worm can't go off the screen as error traps are built into the program for all four edges. You can make the worm narrower in size by making the radius of his original circle smaller. That's the number 10 in line 90. Make the radius larger and the worm will grow even fatter.

Call it a worm, a sketcher, whatever. It looks like toothpaste being squeezed out of a tube onto the face of your TV set. Create your own maze. Write messages.

```
 10 PMODE 1,1
 20 PCLS
 30 SCREEN 1,0
 40 X=128:Y=96:M=247
 50 IF Y<1 THEN Y=1
 60 IF Y>191 THEN Y=191
 70 IF X<1 THEN X=1
 80 IF X>255 THEN X=255
 90 CIRCLE(X,Y),10,4
100 IF PEEK(341)=M THEN Y=Y-1:GOTO 50
110 IF PEEK(342)=M THEN Y=Y+1:GOTO 50
120 IF PEEK(343)=M THEN X=X-1:GOTO 50
```

```
130 IF PEEK(344)=M THEN X=X+1:GOTO 50
140 IF PEEK(345)=M THEN PCLS:X=10:Y=10
    :GOTO 50
150 IF PEEK(338)=M THEN PCLS:X=245:Y=10
    :GOTO 50
160 IF PEEK(339)=M THEN PCLS:X=245:Y=181
    :GOTO 50
170 IF PEEK(340)=M THEN PCLS:X=10:Y=181
    :GOTO 50
180 GOTO 100
999 END
```

Waving Woman

Wonderful Wanda is happy. She likes the fact that you took the time to run her little program.

```
 10 PCLS
 20 SCREEN 1,Ø
 30 PMODE 1,1
 40 CIRCLE(128,50),30,4  'HEAD
 50 CIRCLE(120,40),5,4  'LEFT EYE
 60 CIRCLE(136,40),5,4  'RIGHT EYE
 70 DRAW"C4BM127,50D3F3G3"  'NOSE
 80 LINE(110,65)-(146,65),PSET  'MOUTH
 90 LINE(128,80)-(128,60),PSET  'TORSO
100 DRAW"C4BM128,160G40L20"  'LEFT LEG
110 DRAW"C4BM128,160F40R20"  'RIGHT LEG
120 DRAW"C4BM128,105L20D40NE10
    NR10NF10ND10NG10"  'LEFT ARM
130 DRAW"C1BM128,105R60NH10NU10
    NE10NR10NF10"  'ERASE LOWERED ARM
140 DRAW"C4BM128,105R20E40NH10
    NU10NE10NR10NF10"  'RIGHT ARM HI
150 FOR LL=1 TO 200:NEXT LL
160 DRAW"C1BM128,105R20E40NH10NU10
    NE10NR10NF10"  'ERASE RIGHT ARM HI
170 DRAW"C4BM128,105R60NH10NU10NE10
    NR10NF10"  'LOWER ARM
180 FOR LL=1 TO 200:NEXT LL
190 GOTO 130
```

Color Popper

Here's great fun for the very young! Watch bright colors and listen to their special sounds all day.

The active computer-keyboard keys are B, G, R, Y, CLEAR-key and BREAK-key. Nothing happens if other keys are pressed.

Type it into your computer and RUN it. Press the letter B on the keyboard and blue popcorn appears all over the screen. Beeping as it pops.

Press R for red popcorn and Y for yellow popcorn. Pressing G gives green popcorn but it is invisible against the green screen background.

You can change the background color to whatever the last popcorn color was. Pressing the CLEAR key removes everything from the screen and changes the screen color to whatever the color of the last popcorn was.

Each key press causes a beep, whether popping corn or clearing the screen. Pressing the BREAK key ends the game.

Noise and bright colors are fascinating to watch. Each popcorn key and the CLEAR key have slightly different sounds. Popcorn appears at random positions across and up and down the screen.

By the way, if the background is any color other than green, the popcorn will appear at the end of long arms giving an unusually different look to the screen. To remove this appearance, press G to get green as the last popcorn color and then press CLEAR to get a green background.

```
 10 CLEAR:CLS
 20 IF PEEK(340)=254 THEN 100
 30 IF PEEK(345)=254 THEN 200
 40 IF PEEK(340)=251 THEN 300
 50 IF PEEK(339)=247 THEN 400
 60 IF PEEK(339)=191 THEN 500
 70 GOTO 20
100 RB=RND(511)
110 C$="BLUE"
120 PRINT @ RB,CHR$(175)
130 SOUND 125,1
```

```
140 GOTO 20
200 RG=RND(511)
210 C$="GREEN"
220 PRINT @ RG,CHR$(143)
230 SOUND 150,1
240 GOTO 20
300 RR=RND(511)
310 C$="RED"
320 PRINT @ RR,CHR$(191)
330 SOUND 175,1
340 GOTO 20
400 RY=RND(511)
410 C$="YELLOW"
420 PRINT @ RY,CHR$(159)
430 SOUND 200,1
440 GOTO 20
500 IF C$="GREEN" THEN CLS 1
510 IF C$="YELLOW" THEN CLS 2
520 IF C$="BLUE" THEN CLS 3
530 IF C$="RED" THEN CLS 4
540 SOUND 100,2
550 GOTO 20
```

Chinese Water Torture

Bonkers! That's what you'll be if you watch and listen attentively to this program run for five minutes.

```
 10 CLS
 20 FOR X=Ø TO 63
 30 Y=15
 40 RESET (X,Y)
 50 NEXT X
 60 FOR Y=Ø TO 31
 70 X=31
 80 RESET (X,Y)
 90 NEXT Y
100 SOUND 1,1
110 PRINT
120 GOTO 20
```

Appendix

Functions

Color BASIC Functions

ABS	Finds absolute value
ASC	Finds ASCII number of first character of string
CHR$	Finds character from ASCII, control or graphics code
EOF	Finds whether end of a file has been read
INKEY$	Finds keyboard key being pressed
INT	Changes a number to its integer
JOYSTK	Finds coordinates of joysticks
LEFT$	Finds left portion of string
LEN	Finds length of a string
MEM	Finds amount of memory left
MID$	Finds a portion of a string
POINT	Tells whether graphics point is on or off and its color
RIGHT$	Finds right-hand portion of a string
SGN	Tells whether number is negative, positive or zero
SIN	Finds sine of an angle
STR$	Changes a number to a string
VAL	Changes a string to a number
VARPTR	Finds pointer address

Extended Color BASIC Functions

ATN	Finds arctangent
COS	Finds cosine of an angle
EXP	Finds natural exponential of a number
FIX	Truncates decimal number to whole number
HEX$	Converts decimal number to hexadecimal
LOG	Finds natural logarithm
PEEK	Finds contents of a memory location
POS	Finds print position
PPOINT	Tells whether graphics-screen point is on and its color
STRING$	Finds string of characters as specified
SQR	Finds square root of a number
TAN	Finds tangent of an angle
TIMER	Sets and reads time
USRn	Starts machine-language subroutine

Statements

Color BASIC Statements

AUDIO	Connects cassette recorder to television
CLEAR	Erases variables and reserves string storage
CLOAD	Loads program from cassette
CLOSE	Closes an open file
CLS	Clears display
CONT	Continues program run after BREAK or STOP
CSAVE	Saves program on cassette tape
DATA	Stores data in a program

DIM	Dimensions arrays
END	Concludes the program
EXEC	Gives control to machine-language program
FOR/ TO/ STEP/ NEXT/	Creates a loop
GOSUB	Moves program execution to a subroutine
GOTO	Moves program execution to specified line number
IF/ THEN/ ELSE	Performs test, executes one action if true, another if false
INPUT	Computer stops and waits for keyboard input
INPUT#-1	Brings in data from cassette tape recorder
LIST	Displays program lines
LLIST	Prints program lines on printer
MOTOR	Turns cassette recorder on or off
NEW	Erases all programs in memory
ON/ GOSUB	Moves program execution to specified subroutines
ON/ GOTO	Moves program execution to various line numbers
OPEN	Opens a file
POKE	Deposits a value into a memory location
PRINT	Prints letters, numbers, symbols on TV screen
PRINT#-1	Writes data on a cassette tape
PRINT#-2	Prints letters, numbers, symbols on a printer
PRINT @	Prints message at particular text-screen location
READ	Reads DATA line
REM	Permits comments in program lines
RESET	Resets a point
RESTORE	Sets computer back to first data-line item
RETURN	Brings execution back from subroutine
RUN	Executes program
SET	Sets a text-screen point
SKIPF	Moves to next cassette-tape program
SOUND	Makes tone of specific length
STOP	Stops program execution

Extended Color Basic Statements

CIRCLE	Draws a circle
CLOADM	Loads machine-language program from tape
COLOR	Sets foreground and background color
CSAVEM	Files a machine-language program on tape
DEF FN	Creates numeric function
DEFUSR	Selects entry point for USR function
DEL	Deletes program lines
DLOADM	Loads machine-language program at baud rate specified
DRAW	Draws a line
EDIT	Permits editing of program lines
GET	Stores graphics in a screen rectangle into an array
INSTR	Searches for a target string

LET	Assigns value to a variable
LINE	Draws a line and boxes
LINE INPUT	Computer waits for keyboard input
MID$	Different from Color-BASIC MID$. Replaces a string
PAINT	Colors graphic screen
PCLEAR	Reserves memory for graphic pages
PCLS	Clears graphic screen
PCOPY	Copies from one graphics page to another
PLAY	Performs music
PMODE	Selects graphics resolution and memory page
PRESET	Resets a graphic-screen point to background color
PRINT USING	Prints numbers, letters, symbols in specified format
PSET	Sets a graphic-screen point to a color
PUT	Displays graphics array obtained through GET statement
RENUM	Renumbers program lines
SCREEN	Selects color set and graphics or text screen
TROFF	Permits tracing of program execution
TRON	Stops tracing of program execution

Operators

↑	Exponent
-	Negative
+	Positive
*	Multiplication
/	Division
+	Addition
-	Subtraction
<	Less than
>	More than
=	Equal to
<=	Less than or equal to
>=	More than or equal to
<>	Not equal to
NOT	
AND	
OR	

Video Control Codes

Decimal	Hexadecimal	PRINT CHR$ (Code)
8	Ø8	Backspaces and erases current character
13	ØD	Line feed with carriage return
32	2Ø	Space

Graphic-Character Codes

Using the *color* number, 1 to 8, and the *pattern* number, zero to 15, here's how to generate the correct code:

code = 128 + 16 * (*color* - 1) + *pattern*

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Control Keys

←	Deletes last character; moves cursor back one space
SHIFT ←	Erases current line
BREAK	Interrupts process and returns control to you
CLEAR	Clears the screen
ENTER	Indicates end of current line
SPACEBAR	Enters a blank character and moves cursor one space
SHIFT @	Causes program RUN to pause; press any key to continue
SHIFT Ø	Changes keyboard to upper-and-lowercase

Special Characters

'	Abbreviation for REM
$	Makes a variable a string variable
:	Separates statements on the same program line
?	Abbreviation for PRINT
,	PRINT punctuation; spaces over to mid-screen
;	PRINT punctuation; prints separate items with no spaces between

Error Messages

/Ø	Division by zero
AO	File already open
BS	Subscript out of range
CN	Cannot continue
DD	Redimensioned array
DN	Device-number error
DS	Direct statement in file
FC	Illegal function call
FD	Bad file data
FM	Bad file mode
ID	Illegal direct
IE	Input past end of file
I/O	Input/output error
LS	String too long
NF	NEXT without FOR
NO	File not open
OD	Out of data
OM	Out of memory
OS	Out of string space
OV	Overflow
RG	RETURN without GOSUB
SN	Syntax error
ST	String formula too complex
TM	Type mismatch
UL	Undefined line

Index

books from ARCsoft Publishers

For the TRS-80 PC-1, PC-2, Sharp PC-1211, PC-1500:

99 Tips & Tricks for the New Pocket Computers (PC-2/PC-1500)
128 pages $7.95 ISBN 0-86668-019-5

101 Pocket Computer Programming Tips & Tricks
128 pages $7.95 ISBN 0-86668-004-7

Pocket Computer Programming Made Easy
128 pages $8.95 ISBN 0-86668-009-8

Murder In The Mansion and Other Computer Adventures
96 pages $6.95 ISBN-0-86668-501-4

50 Programs in BASIC for Home, School & Office
96 pages $9.95 ISBN 0-86668-502-2

50 MORE Programs in BASIC for Home, School & Office
96 pages $9.95 ISBN 0-86668-003-9

35 Practical Programs for the Casio Pocket Computer
96 pages $8.95 ISBN 0-86668-014-4

Pocket-BASIC Coding Form programming worksheet tablets
40-sheet pad $2.95 ISBN 0-86668-801-3

For the TRS-80 Color Computer:

101 Color Computer Programming Tips & Tricks
128 pages $7.95 ISBN 0-86668-007-1

55 Color Computer Programs for Home, School & Office
128 pages $9.95 ISBN 0-86668-005-5

55 MORE Color Computer Programs for Home, School & Office
112 pages $9.95 ISBN 0-86668-008-X

Color Computer Graphics
128 pages $9.95 ISBN 0-86668-012-8

The Color Computer Songbook
96 pages $7.95 ISBN 0-86668-011-X

My Buttons Are Blue and Other Love Poems
96 pages $4.95 ISBN 0-86668-013-6

Color Computer BASIC Coding Form program worksheet tablets
40-sheet pad $2.95 ISBN 0-86668-802-1

For the APPLE Computer:

101 APPLE Computer Programming Tips & Tricks
128 pages $8.95 ISBN 0-86668-015-2

33 New APPLE Computer Programs for Home, School & Office
96 pages $8.95 ISBN 0-86668-016-0

APPLE Computer BASIC Coding Form program worksheet tablets
40-sheet pad $2.95 ISBN 0-86668-803-X

For the ATARI 400 and 800 computers:

101 ATARI Computer Programming Tips & Tricks
128 pages $8.95 ISBN 0-86668-022-5

31 New ATARI Computer Programs for Home, School & Office
96 pages $8.95 ISBN 0-86668-018-7

ATARI Computer BASIC Coding Form program worksheet tablets
40-sheet pad $2.95 ISBN 0-86668-806-4

books from ARCsoft Publishers

For the Sinclair ZX-81 and TIMEX/Sinclair 1000 computers:

101 TIMEX 1000 and Sinclair ZX-81 Programming Tips & Tricks
128 pages $7.95 ISBN 0-86668-020-9

37 TIMEX 1000 & Sinclair ZX-81 Programs for Home, School & Office
96 pages $8.95 ISBN 0-86668-021-7

TIMEX 1000 & Sinclair ZX-81 BASIC Coding Form program worksheets
40-sheet pad $2.95 ISBN 0-86668-807-2

For the electronics hobbyist:

25 Quick-N-Easy Electronics Projects
96 pages $4.95 ISBN 0-86668-023-3

25 Electronics Projects for Beginners
96 pages $4.95 ISBN 0-86668-017-9

25 Easy-To-Build One-Night & Weekend Electronics Projects
96 pages $4.95 ISBN 0-86668-010-1

Computer program-writing worksheets:
Each in tablet of 40 sheets with stiff backing

Universal BASIC Coding Form	$2.95
IBM Personal Computer BASIC Coding Form	$2.95
ATARI Computer BASIC Coding Form	$2.95
Pocket Computer BASIC Coding Form	$2.95
APPLE Computer BASIC Coding Form	$2.95
Color Computer BASIC Coding Form	$2.95
TIMEX 1000/Sinclair ZX-81 BASIC Form	$2.95

ISBN: International Standard Book Number

ARCsoft Publishers

Post Office Box 132
Woodsboro, Maryland 21798

Color Computer Book VI $9.95

Color Computer Graphics

by Ron Clark

If you have a personal computer and want to do video graphics, you must have this handbook. It is the one complete introduction, written in an easy-to-read format for folks of all ages and all walks of life. The simple down-to-earth language in this learn-by-doing instruction guide starts at the very beginning and covers all the elements you need to know to be able to design and execute exciting new graphics for your computer video screen.

Written in *Color Basic* and *Extended Color BASIC* for the **TRS-80 Color Computer**, this complete coverage includes a thorough introduction to CIRCLE, PAINT, DRAW, LINE, COLOR, SCREEN, POKE, SET, RESET, PCLEAR, PSET, PPOINT, PCLS and many more of the powerful BASIC graphics words. All programs have been thoroughly tested on the **TRS-80 Color Computer**.

Learn all about low, medium and high-resolution graphics, and how to create each. See the difference between text and graphics modes. Find out how to set up your computer for graphics programming. Change screen colors, make animation, turn on sound and music too!

Draw curves, ovals, arcs, angles, boxes, figures, games, fine arts, even flowers. You'll quickly understand exactly how to use the powerful BASIC-language words built into your computer to write software to reproduce beautiful graphics in nine colors on your computer's video screen.

Basic instruction in this book is applicable to graphics, with only minor changes in program lines, on TRS-80 Model I/II/III/16, Apple II/III, Atari 400/800, Commodore, Vic, Pet, Sinclair ZX-81, Micro Ace, Timex 1000, and many others. IBM Personal Computer owners will find graphics commands here almost identical to those used by their PCs so these programs will easily run on the IBM PC. Everything you need to know to get started in color computer graphics.

ARCsoft Publishers
WOODSBORO, MARYLAND

ISBN 0-86668-012-8