TRS-80 COLOR COMPUTER GRAPHICS

Learn how to create interesting graphics by mastering the fantastic capabilities of Extended Color BASIC!

Don Inman with Dymax
TRS-80 Color Computer
Graphics
Inman, Don.
TRS-80 color computer graphics.

1. Computer graphics. 2. TRS-80 (Computer)
I. Dymax (Firm) II. Title. III. Title: TRS80 color com-
puter graphics. IV. Title: T.R.S.-80 color computer
graphics.
T385.I474 1982 001.55'3 82-7670
ISBN 0-8359-7865-6 AARC2
ISBN 0-8359-7864-8 (pbk.)

A Prentice-Hall Company
Reston, Virginia 22090

All rights reserved. No part of this book
may be reproduced, in any way or by any means,
without permission in writing from the publisher.

10 9 8 7 6 5 4 3 2

Printed in the United States of America
Contents

PREFACE ix

Chapter 1 FUNDAMENTAL COLORING 1
The Colors, 3
Setting Up the Screen, 5
Setting Color Points, 7
Setting Points to Draw Lines, 11
Erasing Points, 16
Summary, 16
Chapter Test, 17
Answers, 19

Chapter 2 COLORING LINES AND CIRCLES 21
Drawing Lines, 23
Changing Colors, 27
Connecting Lines, 30
Drawing Boxes, 33
Filling Boxes with Color, 35
Drawing Circles, 37
Summary, 39
Chapter Test, 40
Answers, 43

Chapter 3 MORE CIRCLES, PAINT, AND POINT 47
More About Circles, 47
Squeezing a Circle, 48
CONTENTS

Chapter 4

DRAW 73
The DRAW String, 73
DRAW Directions, 76
Drawing Spirolaterals, 77
More Directional Moves, 80
Many Lines from the Same Point, 84
Color Option, 85
Substrings, 87
Catenation, 89
Relative Motion, 91
Summary, 93
Chapter Test, 94
Answers, 97

Chapter 5

PAGE FUNCTIONS AND MORE DRAW 101
Scale Up or Down, 101
Turning Pages in Mode 0, 107
Act 1, 111
Act 2, 113
Act 3, 114
Turning Pages in Other Modes, 116
Summary, 122
Chapter Test, 123
Answers, 127

Chapter 6

PUT, GET AND JOYSTICKS 129
GET and PUT, 129
Draw It Small, 133
The Joysticks, 136
Push Buttons, 141
How to Use the Joysticks, 143
Using Joysticks in Games, 144
A Shooting Gallery, 147
Summary, 150
Chapter 7

**SOUND WITH PLAY** 157
SOUND, 157
Adding SOUND to Graphics, 159
The PLAY Function, 160
Other Noises, 165
Sounds by Joysticks, 171
Sound With Graphics, 173
Summary, 174
Chapter Test, 175
Answers, 178

Chapter 8

**DISPLAYING TEXT AND TIMING** 181
Switching From Graphics to Text Mode, 181
Drawing Text Characters, 184
Creating an Alphabet Array, 186
Using the Text Character Subroutine, 190
Time Out, 192
Timing More Interesting Events, 194
Time Your Target Practice, 198
Summary, 201
Chapter Test, 202
Answers, 205

Chapter 9

**THE USR FUNCTION** 207
Memory Use for Machine Language Subroutines, 208
The USR Function, 209
Machine Language Format, 212
Hex-Decimal Conversions, 215
Analyzing a Program, 219
Help with Those Mysterious Machine Codes, 221
Low Resolution Graphics, 222
Summary, 225
Chapter Test, 227
Answers, 230

Chapter 10

**ADDITIONAL PROGRAMS** 233

1A Fundamental Coloring, 234
   PCLS, PMODE, SCREEN, PSET, PRESET, COLOR

2A Coloring Lines and Circles, 241
   LINE, CIRCLE
CONTENTS

3A More Circles, PAINT, and POINT, 246  
PAINT, PPOINT

4A DRAW, 254  
DRAW

5A Page Functions and More DRAW, 261  
Turning Pages, PCLEAR, PCOPY

6A PUT, GET, And JOYsticks, 267  
GET, PUT, JOYSTK, PEEK(65280)

7A SOUND With PLAY, 274  
SOUND, PLAY

8A Displaying Text and Timing, 277  
TIMER

9A The USR Function, 282  
CLEAR, DEF USR, USR, HEX$, &H

Appendix A COLOR COMPUTER CONNECTIONS  287
Appendix B MEMORY MAP FOR 16K RAM—EXTENDED  
COLOR BASIC COMPUTER  289
Appendix C GRAPHICS SCREEN  291
Appendix D SOUND AND PLAY TABLES  293
Appendix E ASCII CHARACTER CODES  295
INDEX  297
To make full use of the TRS-80 Color Computer, you must have a good understanding of its color graphics capabilities and know how to use them. The purpose of this book is to help you discover how the ability to create interesting graphics can enhance your own computer programs.

As you probably know, the TRS-80 Color Computer comes in two versions. The 4K memory version with Color BASIC is capable of displaying color graphics on a 64-by-32 grid containing 2048 points that can be set to several different colors. The 16K memory version with Extended Color BASIC is capable of displaying color graphics on a 256-by-192 grid containing 49,152 points. This provides high resolution displays that can be manipulated in interesting ways.

In this book, the 16K Extended Color BASIC graphics features are fully explored. Through several graphics modes, the Extended Color BASIC allows you to explore and compare three different levels of graphics—low, medium, and high resolution.

Low-resolution graphics are displayed as a 128-by-96 grid of points. Medium-resolution graphics are displayed as a 128-by-192 grid of points; high-resolution graphics, as a 256-by-192 grid of points. You can also use the 64-by-32 mode implemented in the 4K Color BASIC machine.

The color graphics statements are presented gradually so that you will gain more control of the video display as you learn each new graphics statement. You will acquire a complete understanding of the graphics capabilities of the Extended BASIC by the time you finish this book. We encourage you to use this book while you are sitting before your color computer. Try each program and exercise in the book. In this way the computer will teach you how to use it.
Each chapter ends with a summary, a test covering essential information, and answers to the odd-numbered test exercises. Even-numbered exercises are left for you to ponder. If you really are baffled, write to the author at

The Dymax Gazette
P.O. Box 310
Menlo Park, CA 94025

Chapter 10 is composed of thoroughly documented programs that apply the graphics techniques learned in the first nine chapters. These applications supplement the material you learned earlier and are keyed to the preceding chapters. If you want to use them more than once, it would be helpful to save them on cassette tape. They may be used along with the applicable chapter, or they may serve as a reference. In addition to the application programs, Chapter 10 contains some useful subroutines that you may wish to add to your own programs.

As you proceed through the book, no doubt you will have some ideas that you will want to try. We encourage you to experiment with the color computer. You may make mistakes, but your computer is very friendly and forgiving. You will learn more by trying new things and making mistakes than if you do only those things for which you have rules and regulations.

Explore the TRS-80 Color Computer! Discover what it can do and what it cannot do.
TRS-80 Color Computer Graphics
Welcome to the world of color and graphics. Whether you're going to use your TRS-80 Color Computer at home, at school, at church, or at some other location, a new, fascinating experience awaits you.

The TRS-80 Color Computer is friendly and easy to use. It can provide hours of enjoyment for children and for adults through its many educational and recreational uses. You can also use it for serious work in math, science, business, or other applications if you wish. We will be most concerned with the educational and recreational uses in this book.

If you have had some experience with the computer language called BASIC, you will have little trouble using the Color Computer. This book is written for intermediate to advanced BASIC users. We assume that you know a good many computer terms, can understand computer literature, and enjoy conversing with others about all that you have read and learned about computers.

When you have finished reading this chapter, have worked all the exercises, and tried all the demonstrations, you will

- know how to set or to change the background color on the TV screen
- know how to choose from five available graphics modes
- know which colors can be used with each of the graphics modes
- be able to change foreground colors for each graphics mode
- be able to turn on points anywhere on the screen in any available color
- be able to combine points on the screen to form horizontal and vertical lines
- know how to use these graphics statements:
We assume that you have unpacked your TRS-80 Color Computer and any accessories purchased with it. Plunge right in.

You can start learning about computer graphics with just a TRS-80 Color Computer and a television set. However, for full enjoyment and full use of the computer's capabilities, we recommend the following equipment that we used in writing this book:

a. TRS-80 Color Computer with Extended Color BASIC
b. color television set
c. Radio Shack CTR-80A cassette recorder
d. pair of Radio Shack joysticks

Appendix A shows how to connect the four pieces of recommended equipment to make a complete system. These connections are also shown in the Radio Shack color computer operation manual. The Radio Shack color computer manuals should be studied thoroughly before you proceed in this book.

Two kinds of BASIC are available for the TRS-80 Color Computer. The 4K memory Color Computer uses a version known as Color BASIC. A second version of BASIC, which comes with the 16K memory Color Computer, is called Extended COLOR BASIC. This second version has more graphics capability. It can do all the things that the 4K Color BASIC can do but has additional statements and commands. The Extended Color BASIC is used in this book.

When your computer has been connected, turn on the computer and the TV. You should see a message that looks like this:

EXTENDED COLOR BASIC 1.0
COPYRIGHT (C) 1980 BY TANDY
UNDER LICENSE FROM MICROSOFT
OK

Notice that the background color is green and the text appears in dark letters. The cursor changes colors so fast that we couldn't catch all the colors (but we suspect that it is going through its full range of colors).

The TRS-80 Color Computer may be used in either of two display modes. When you first turn on the computer, it is automatically set to the
text mode. The background is always green and the text black when you are using the text mode. The blinking cursor indicates that the computer has finished its duties and is waiting for you to tell it to do something.

---

**THE COLORS**

The first thing that you may want to do is to see what colors are available for graphics backgrounds.

**Exercise 1-1**

Enter the program that follows. Then run it, and write the appropriate colors below the numbered screen drawings as the computer goes through the values 0 through 8 for the variable, X.

```basic
10   FOR X = 0 TO 8
20      CLS X
30   FOR Y = 1 TO 1500: NEXT Y
40      NEXT X
50     CLS
```

<table>
<thead>
<tr>
<th>X=0</th>
<th>X=1</th>
<th>X=2</th>
<th>X=3</th>
<th>X=4</th>
</tr>
</thead>
<tbody>
<tr>
<td>black</td>
<td>green</td>
<td>yellow</td>
<td>blue</td>
<td>red</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X=5</th>
<th>X=6</th>
<th>X=7</th>
<th>X=8</th>
</tr>
</thead>
<tbody>
<tr>
<td>buff</td>
<td>cyan</td>
<td>magenta</td>
<td>orange</td>
</tr>
</tbody>
</table>

(Answers are at the end of the chapter.)

The colors displayed will depend on the adjustment of your color TV set. Your TRS-80 Color Computer Operation Manual (Radio Shack catalog no. 3001/3002) has a program to adjust colors on your TV for low-resolution graphics. To test the colors for Extended Color BASIC, enter and run this program. (Don't worry about any new statements that you see. They will be explained later.)
Color Bar Program To Adjust TV

```
100 REM * SET GRAPHICS SCREEN *
110 PMODE 1,1
120 PCLS
130 SCREEN 1,0
200 REM * ROWS *
210 FOR Y = 0 TO 192 STEP 2
300 REM * COLOR COLUMNS *
310 FOR X = 0 TO 127 STEP 2
320 C = INT(X/32) +1
330 PSET (X+48,Y,C)
340 NEXT X
400 REM * GOTO NEW ROW *
410 NEXT Y
500 REM * ALTERNATE COLOR SETS *
510 FOR W = 1 TO 2000: NEXT W
520 SCREEN 1,1
530 FOR W = 1 TO 2000: NEXT W
540 SCREEN 1,0
550 GOTO 510
```

After you run the program, the colors should appear as follows. First,
As the two pictures alternate, adjust the color controls on your TV set until you get the desired result.

**SETTING UP THE SCREEN**

The first Extended Color BASIC statement that you'll need to use is the mode selector.

```
PMODE n,m
```

- **n** is an integer from 0 through 4
- **m** is an integer from 1 through 8

The integer **n** selects which of five graphics modes will be used. Zero (0) and one (1) select a low-resolution mode. Two (2) and three (3) select a medium-resolution mode. Four (4) selects the high-resolution mode.

The integer **m** selects the starting page of graphics memory to be used. The starting page will be discussed in more detail in Chapter 5. For the time being, always select page 1.

**TABLE 1-1. GRAPHICS MODES**

<table>
<thead>
<tr>
<th>PMODE Statement</th>
<th>Size of Graphics</th>
<th>Number of Colors</th>
<th>Relative Size of Graphics Grid Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMODE 4,1</td>
<td>256x192</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>PMODE 3,1</td>
<td>128x192</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>PMODE 2,1</td>
<td>128x192</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>PMODE 1,1</td>
<td>128x96</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>PMODE 0,1</td>
<td>128x96</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Another graphics statement used in combination with **PMODE** is

```
SCREEN n,m
```

- **n** is either zero (0) or one (1)
- **m** is either zero (0) or one (1)

The integer **n** selects whether text or graphics will be displayed on the screen. If zero (0) is selected for **n**, only text will be displayed. If one (1) is used for **n**, only graphics will be displayed.

The integer **m** selects one of two color sets available for the particular graphics mode called for by **PMODE**. Thus, the two statements **PMODE** and **SCREEN** work together to produce the desired formats. Table 1-2 shows the possible combinations that can be used.
TABLE 1-2. BACKGROUND AND FOREGROUND COLORS

<table>
<thead>
<tr>
<th>PMODE Statement</th>
<th>SCREEN Statement</th>
<th>Background Color</th>
<th>Foreground Colors Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMODE 4,1 or PMODE 2,1 or PMODE 0,1</td>
<td>SCREEN 1,0</td>
<td>black</td>
<td>green</td>
</tr>
<tr>
<td></td>
<td>SCREEN 1,1</td>
<td>black</td>
<td>buff</td>
</tr>
<tr>
<td>MODE 3,1 or PMODE 1,1</td>
<td>SCREEN 1,0</td>
<td>green</td>
<td>yellow, blue, red</td>
</tr>
<tr>
<td></td>
<td>SCREEN 1,1</td>
<td>buff</td>
<td>cyan, magenta, orange</td>
</tr>
</tbody>
</table>

Exercise 1-2

What foreground colors can be used with the following statements.

a. PMODE 2,1 with SCREEN 1,0
   - green

b. PMODE 3,1 with SCREEN 1,1
   - cyan, magenta, orange

c. PMODE 4,1 with SCREEN 1,1
   - buff

(Answers are at the end of the chapter.)

Before you start displaying graphics on the screen, you must prepare the screen for action. Do you remember the CLS command used in other TRS-80 BASICS? The Extended Color BASIC command to clear the screen for graphics is PCLS. CLS still works to clear the Color Computer for text. When you use PCLS, the current background color is used, unless you specify a color number.

PCLS n

- n may be any numeric expression whose value is 0 through 8.
  - Remember, the n is optional.

Run this program and note the changing colors.

100 REM * SET GRAPHICS *
110 PMODE 1,1
120 SCREEN 1,0  ← graphics background green
200 REM * CHANGE COLORS *
210 FOR X = 1 TO 4
220 PCLS X  ← graphics area colors 1-4
230 FOR Y = 1 TO 1500: NEXT Y
240 NEXT X
300 REM * CLEAR SCREEN FOR TEXT *
310 CLS  ← clear the screen for text

Square graphics area changes from green to yellow to blue to red.

Green on green is hard to see; therefore, the screen first appears to be completely green without a square area of another color.

At the end of the program, line 310 causes the screen to be cleared for the text mode. The OK prompt appears, followed by the blinking cursor on the next line.

OK

The three statements—PMODE, PCLS, and SCREEN—work together to select
a. the graphics mode 0-4 and starting page (PMODE)
 b. the background color to be used (PCLS)
   c. graphics or text and color set to be used (SCREEN)

Before you try something fancy, practice with the normal background and foreground colors shown in Table 1-2.

SETTING COLOR POINTS

You’re now ready to plot something on the screen. Let’s start with some points. The Extended Color BASIC statement to turn on a point is
Think of the graphics positions as starting in the upper left corner with column 0 and row 0. There are 256 columns and 192 rows.

Study the following program carefully. PSET is the only new statement. The color selected will depend on the PMODE and SCREEN statements. See if you can predict where the point will appear on the screen, what color it will be, and what color the background will be. Then run the program to confirm your predictions.

10 'SET MODE AND CLEAR SCREEN
20 PMODE 4, 1
30 PCLS
40 'SET SCREEN FOR GRAPHICS, COLOR
50 SCREEN 1, 0
60 'SET THE POINT
70 PSET (127, 95, 1)
80 GOTO 80

If you look hard, you can see the point near the center of the screen. The point is green on a black background. As shown in Table 1-2, in PMODE 4, 1 with SCREEN 1, 0 this is to be expected. When you run the program, the very tiny green point appears at row 95, column 127.
You were using the high-resolution mode called for by PMODE 4, 1. Therefore, the dot was very small.

Change line 20 of the program to

20 PMODE 2, 1

Run the program with this change, and see if you detect any difference. Then change line 20 again. This time try

20 PMODE 0, 1

Run the program once more, and see if you detect any further difference.

We hope that you noticed that the point was slightly bigger (or maybe appeared to be brighter) each time. As shown in Table 1-1, the three modes used in the previous programs produce the following size points:

<table>
<thead>
<tr>
<th>PMODE</th>
<th>Size of Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4, 1</td>
<td></td>
<td>□</td>
</tr>
<tr>
<td>2, 1</td>
<td></td>
<td>□□</td>
</tr>
<tr>
<td>0, 1</td>
<td></td>
<td>□□□□□□□□□□□□□□□□</td>
</tr>
</tbody>
</table>

Notice that the same X, Y positions were used for each of the modes in the three versions of the point-setting program. Yet, the point was enlarged as the resolution was lowered from mode 4 to mode 2 to mode 0. Look at the following blocks of four points to see how it was done.

In the high-resolution mode (4), the point 127, 95 was turned on.

In the medium-resolution mode (2), the points 127, 95 and 126, 95 were turned on even though the point specified was 127, 95.

In the low-resolution mode (0), points 126, 94; 127, 94; 126, 95; and 127, 95 all were turned on even though the point specified was still 127, 95.
Low-resolution modes always turn on four points, medium-resolution modes always turn on two points, and high-resolution modes turn on only one point.

Specifying any of the points—126, 94; 127, 94; 126, 95; or 127, 95—in mode 0 would turn on all four of the points. Specifying 127, 94 or 126, 94 in mode 2 would turn on both those points. Specifying 126, 95 or 127, 95 in mode 2 would turn on both those points. In mode 4, the point specified is the only point turned on.

Exercise 1-3

Shade in each of the points that you think would be turned on by the execution of the following statements:

a. 10 PMODE 4, 1
   20 PCLS
   30 SCREEN 1, 0
   40 PSET (126, 94, 1)

b. 10 PMODE 2, 1
   20 PCLS
   30 SCREEN 1, 0
   40 PSET (126, 94, 1)

c. 10 PMODE 0, 1
   20 PCLS
   30 SCREEN 1, 0
   40 PSET (126, 94, 1)

(Answers are at the end of the chapter.)
SETTING POINTS TO DRAW LINES

Of course, you can draw a line by setting several points. Now that you know how to set points, study this program, which draws a line in the high-resolution mode from point 127, 15 to 147, 15. The line is drawn with a buff color on a black background.

100 ' SET UP SCREEN
110 PMODE 4, 1 ←set the mode
120 PCLS ←clear screen—graphics
130 SCREEN 1, 1 ←set screen (buff/black)
200 ' SET POINTS
210 FOR X = 127 TO 147 ←FOR-NEXT loop to set the points
220 PSET (X, 15, 5) buff
230 NEXT X
300 ' LOOK WHILE LOOPING
310 GOTO 310

No matter what number you use for color in the PSET statement in line 220, PMODE and SCREEN parameters fix buff and black as the colors to be displayed (see Table 1-2).

After writing the program, we hope that you ran it to see the results like this:

```
\[\text{thin buff line}\]
\[\text{black background}\]
```

Now it's time to see if you can be a little fancier. There is a way to reverse the background and foreground colors in the previous program with the statement

```
COLOR, n, m
```

The color numbers used in the COLOR statement must conform to those
selected by the PMODE and SCREEN statements, in this case, buff and black. Insert line 115 and change line 220 of the previous program in this way:

```
115 COLOR 0, 5                      ↘ put the color statement before PCLS
  black — buff                      foreground first (black)
```

```
220 PSET (X, 15, 0)                 ↘ black
```

Run the revised program and notice the difference in the new display. The colors are reversed. A black line is drawn on a buff background.

You have been using the high-resolution mode (4) in the previous programs. Run the program again, but this time change to the following modes:

- **a. PMODE 2, 1**
  - 100 ' SET SCREEN—MODE 2
  - 110 PMODE 2, 1
  - 115 COLOR 0, 5
  - 120 PCLS
  - 130 SCREEN 1,1
  - 200 ' SET POINTS
  - 210 FOR X = 127 to 147
  - 220 PSET (X, 15, 0)
  - 230 NEXT X
  - 300 ' LOOP AROUND
  - 310 GOTO 310

- **b. PMODE 0, 1**
  - 100 ' SET SCREEN—MODE 0
  - 110 PMODE 0, 1
  - 115 COLOR 0, 5
  - 120 PCLS
  - 130 SCREEN 1,1
  - 200 ' SET POINTS
  - 210 FOR X = 127 to 147
  - 220 PSET (X, 15, 0)
  - 230 NEXT X
  - 300 ' LOOP AROUND
  - 310 GOTO 310

When these two versions are executed, look at the display carefully. There seems to be no difference caused by the change from PMODE 4, 1 to
PMODE 2, 1. PMODE 4, 1 draws □ points and PMODE 2, 1 draws □ points. The horizontal line is the same width in both cases. The line drawn by program b is definitely wider. PMODE 0, 1 draws □ points and is therefore twice as wide.

Modes 0, 2, and 4 allow only two colors at a time chosen from two color sets.

Let's move on to graphics modes 1 and 3. Modes 1 and 3 provide four colors at a time, chosen from two color sets as shown in Table 1-3.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Screen</th>
<th>Colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMODE 1, 1,</td>
<td>SCREEN 1, 0</td>
<td>1-green, 2-yellow, 3-blue, 4-red</td>
</tr>
<tr>
<td>or PMODE 3, 1</td>
<td>SCREEN 1, 1</td>
<td>5-buff, 6-cyan, 7-magenta, 8-orange</td>
</tr>
</tbody>
</table>

Exercise 1-4

a. Study this program, answer the questions, and then run the program.

100 'SET SCREEN
110 PMODE 1, 1
120 PCLS
130 SCREEN 1, 0
200 'SET POINTS
210 FOR Y = 85 TO 105
220 PSET (127, Y, 2)
230 NEXT Y
300 'LOOP HERE
310 GOTO 310

1. What level of resolution is used in the program? Low
2. What is the background color? Green
3. What is the foreground color? Yellow, Blue, Red
4. Is the line horizontal or vertical? Vertical

b. Add a COLOR statement and change the PSET statement to reverse the background and foreground colors.
Remember that in modes 1 and 3 you can use up to four colors. You have used yellow as background and green as foreground. You can change the program to add a vertical blue line at $X = 101$ and a vertical red line at $X = 153$. Make the new lines the same length as the green line. Put your additions at lines 225 and 227.

```
225  PSET (101, Y, 3)
227  PSET (153, Y, 4)
```

The program now looks like this:

```
100  SET SCREEN
110  PMODE 1, 1
115  COLOR 1, 2
120  PCLS
130  SCREEN 1, 0
200  SET POINTS
210  FOR Y = 85 TO 105
220  PSET (127, Y, 1)
225  PSET (101, Y, 3)
227  PSET (153, Y, 4)
230  NEXT Y
300  LOOP THE LOOP
310  GOTO 310
```

Draw a sketch of what you think you will see on the display. Use colored pens or crayons or just label the colors in your drawing.
When you run the program, it should look like this:

Let's get a little fancier with the program. Delete lines 300 and 310 and add a loop that will draw three horizontal lines across the existing vertical lines. Put a blue line on top, a green line in the middle, and a red line on the bottom.

```plaintext
100  ' SET SCREEN
110  PMODE 1, 1
115  COLOR 1, 2
120  PCLS
130  SCREEN 1, 0
200  ' DRAW VERTICAL
210  FOR Y = 85 TO 105
220   PSET (127, Y, 1)
225   PSET (101, Y, 3)
230   PSET (153, Y, 4)
230  NEXT Y
300  ' DRAW HORIZONTAL
310  FOR X = 101 TO 153
320   PSET (X, 85, 3)
330   PSET (X, 95, 1)
340   PSET (X, 105, 4)
350  NEXT X
400  ' NOW LOOK AT IT
410  GOTO 410
```
ERASING POINTS

Just as points can be displayed in colors by using PSET, they can be erased by using

PRESET (X, Y)

You don’t have to set a color for PRESET; the background color is used. When you erase a point, you want the background color to “show through.”

The previous program had a yellow background. You can add two PRESET statements, each in a FOR-NEXT loop to erase the two green lines.

Add

400 ' ERASE GREEN LINES
410 FOR X = 103 TO 151
420 PRESET (X, 95)
430 NEXT X
440 FOR Y = 87 TO 103
450 PRESET (127, Y)
460 NEXT Y
500 ' NOW LOOP
510 GOTO 510

Summary

In this chapter, you learned that the TRS-80 Color Computer can display nine colors: black, green, yellow, blue, red, buff, cyan, magenta, and orange. You also learned that

- PMODE n, m sets the graphics mode n (0-4) and the starting graphics memory page m (1-8)
- SCREEN n, m sets the screen display n (0 for text, 1 for graphics) and m (0 or 1, the color set to be used)
- CLS clears the screen for text
- PCLS n clears the screen for graphics; n (0-8) sets the color background; n is optional (green or buff is used for background color if no value is given for n)
- PSET (X, Y, C) sets a graphics point at screen position X (column), Y (row) with color C
- COLOR n, m sets color for foreground (n) and background (m)
- PRESET (X, Y) sets the point at column X and row Y to the background color (apparently turning it off)
You learned to use combinations of these commands to color the video screen with points, horizontal lines, and vertical lines. Now show how much you have learned in the chapter by answering the exercises in the chapter test.

Chapter Test

1. This book was written for the 16K memory TRS-80 Color Computer using extended color BASIC.

2. What are the four pieces of equipment recommended for using the Color Computer and this book?
   a. ______________________________
   b. ______________________________
   c. ______________________________
   d. ______________________________

3. Two integers are used in the statement: PMODE n, m. The integer m represents the starting graphics memory page.
   a. What does the n represent?
   ________________________________
   b. What numeric values may be used for n?
   ________________________________ through ________________________________

4. What do the letters n and m represent in the statement: SCREEN n, m?
   n represents ________________________________
   m represents ________________________________

5. The statement PSET (128, 96, 5) turns on a color point on the display. What does each parameter tell the computer?
   a. 128 ________________________________
   b. 96 ________________________________
   c. 5 ________________________________

6. The following FOR-NEXT loop is executed with PMODE 1, 1 and SCREEN 1, 1.

   60 FOR X = 5 TO 8
   70 PCLS X
   80 NEXT X
The colors displayed as background would change from ___________ to ___________ to ___________ to ___________.

7. These lines are executed:

```
10 PMODE 0, 1
20 SCREEN 1, 0
30 PCLS
40 PSET (124, 90, 1)
```

Shade in the points that would be turned on.

```
<table>
<thead>
<tr>
<th>123</th>
<th>124</th>
<th>125</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>89</td>
</tr>
<tr>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>90</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>91</td>
</tr>
</tbody>
</table>
```

8. What statement can be used to control the foreground and background display colors? ____________________________

9. Describe the display after these lines have been executed:

```
10 PMODE 3, 1
20 PCLS
30 SCREEN 1, 1
40 FOR Y = 85 TO 105
50 PSET (10, Y, 6)
60 NEXT Y
70 GOTO 70
```

A __________________________ line is drawn near the _____________ of the screen. The background color is _______________________; the foreground color is _____________________________.

10. A time delay is added to the program of exercise 9 at line 70:

```
70 FOR W = 1 TO 1000: NEXT W
```

You then want to erase the line drawn by the original program. What should be added?
Answers to Exercises Within the Chapter

Exercise 1-6

DITE/fil

Exercise 1-2

a. green
b. cyan, magenta, or orange
c. buff

Exercise 1-3

a. 

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>126</td>
</tr>
<tr>
<td>94</td>
</tr>
</tbody>
</table>

b. 

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>126</td>
<td>127</td>
</tr>
<tr>
<td>94</td>
<td>94</td>
</tr>
</tbody>
</table>

c. 

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>126</td>
<td>127</td>
</tr>
<tr>
<td>94</td>
<td>94</td>
</tr>
<tr>
<td>95</td>
<td></td>
</tr>
</tbody>
</table>
Exercise 1-4

a. 1. low
   2. green
   3. yellow
   4. vertical
b. 115 COLOR 1, 2  (1-green foreground, 2-yellow background)
   220 PSET (127, Y, 1)  (green points)

Answers to Odd-Numbered Exercises of Chapter Test

1. Extended Color
3. a. one of five graphic modes
   b. 0 through 4
5. a. 128-column location for the point
   b. 96-row location for the point
   c. 5-buff colored point

7. X→

<table>
<thead>
<tr>
<th>123</th>
<th>124</th>
<th>125</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>89</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>Y</td>
</tr>
<tr>
<td>91</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9. vertical, left, buff, cyan
In the first chapter, you were setting points of various colors on backgrounds of other colors. You also set a group of points next to each other to form a line.

This chapter explains how to form colored lines in quicker and more efficient ways. You'll even discover how to draw a circle quickly on the screen. By the time you have finished Chapter 2, you will have learned

- to display a line by telling the color computer where its end points are
- to add quickly a second line that starts from the end of the first line and goes to a new point
- to connect several lines together
- to display a rectangle by giving the coordinates of two corners
- to fill a rectangle with color
- to draw a circle quickly
- to do other exciting things

You already know how to display a line on the screen by setting points. Use this method once more to see how points are displayed by the three levels of resolution:

<table>
<thead>
<tr>
<th>PMODE</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 1</td>
<td>✖️</td>
<td>low resolution</td>
</tr>
<tr>
<td>2, 1</td>
<td>☐☐</td>
<td>medium resolution</td>
</tr>
<tr>
<td>4, 1</td>
<td>☐</td>
<td>high resolution</td>
</tr>
</tbody>
</table>

Mode 3 would set points in the same way as mode 2. Mode 1 would set points in the same way as mode 0. Therefore, modes 3 and 1 are not demonstrated in the following program.
The low-resolution mode is used first.

100 ' SET LOW RESOLUTION SCREEN
110 PMODE 0, 1
120 PCLS
130 SCREEN 1, 0
200 ' DRAW VERTICAL
210 FOR Y = 0 TO 127 STEP 2
220 PSET (4, Y, 1)
230 NEXT Y
300 ' LOOP
310 GOTO 310

The first point set (when Y=0) fills columns 4 and 5 at rows 0 and 1. Y is then increased by two in successive steps. See Figure 2-1 to see how this is done. Then run the program to see how mode 0 sets a double column, two rows at a time.

Now change line 110 to

110 PMODE 2, 1

Rerun the program. The line drawn by PMODE 2, 1 seems to have a

Figure 2-1. Points PSET in PMODE 0.
space between rows since each point is set as a two-position block (■□). The line drawn previously by PMODE 0, 1 was more solid since each point set a four position block (□□□□). See Figure 2-2 for a diagram of the formation of blocks for PMODE 2, 1. The medium-resolution mode (PMODE 2) draws the first PSET (Y=0) in columns 4 and 5 at row 0 only. The second PSET (Y=2) draws a point in columns 4 and 5 at row 2. Therefore, a space will exist between each row of blocks.

Change line 110 again. This time make it

```
110 PMODE 4, 1
```

Rerun the program. Can you tell the difference between PMODE 2 and PMODE 4? You must have good eyesight, but PMODE 4 appears to make a finer line. It is actually setting single points as shown in Figure 2-3.

---

**Figure 2-2. Points PSET in PMODE 2.**

Lines 210-230 of the previous program can be replaced by a single statement:

```
210 LINE (4, 0) - (4, 127), PSET
```
Figure 2-3. Points PSET in PMODE 4.

Can you see how this works? If you can, go directly to Exercise 5. If you cannot, read on. The LINE statement gives the X, Y coordinates (in parentheses) where the line is to begin, then a dash, then the X, Y coordinates (in parentheses) where the line is to end, then a comma, and finally the word PSET.

\[
\text{LINE} \ (X_1, \ Y_1) - (X_2, \ Y_2), \ PSET
\]

The LINE statement quickly draws the entire line.

Exercise 2-1

Write a program using the LINE statement that will do the same thing as the program in exercise 1-4 of Chapter 1. You can save two lines.
Here is a program with the same PMODE, SCREEN, and PCLS statements used earlier. It uses LINE to draw a line half-way across the top of the screen, starting at point 0, 4. It shows again how quickly and easily you can use the LINE statement to draw a line rather than setting each point in the line.

```
100 ' SET SCREEN
110 PMODE 1, 1
120 PCLS
130 SCREEN 1, 0
200 ' DRAW LINE
210 LINE (0, 4) - (128, 4), PSET
300 ' LOOP
310 GOTO 310
```

Notice that the program produced a red line on a green background. If no color is specified when drawing a line, the lowest numbered color in the color set (1 = green in this case) is used as the background color. The highest numbered color in the color set (4 = red in this case) is used as the foreground color. Both the foreground and background colors can be changed by using the statement

```
COLOR n, m
```

where n is the foreground color and m is the background color. The values for n and m must be in the color set selected by the SCREEN statement.

Many of the previous programs have been run in PMODE 1, 1 and SCREEN 1, 0. Remember that the legal color codes to use for n and m in the COLOR statement for these conditions are 1 = green, 2 = yellow, 3 = blue, and 4 = red.

To change the previous program to draw a blue line on the green background, you could insert a COLOR statement at line 125. It would be

```
125 COLOR 3, 1
```

The revised program would be

```
100 ' SET SCREEN
110 PMODE 1, 1
```
When the program is run, you see

```
200 'DRAW BLUE LINE
210 LINE (0, 4) - (128, 4), PSET
```

Exercise 2-2

Insert lines in the revised program to show each of the four colors (blue, yellow, green, and red) on separate lines spaced two rows apart.

```
100 'SET SCREEN
110 PMODE 1, 1
120 PCLS
125 COLOR 3, 1
130 SCREEN 1, 0
200 'DRAW COLORFUL LINES
210 LINE (0, 4) - (128, 4), PSET "1st line blue"
```

```
220 _______________ "2nd line yellow"
230 _______________
240 _______________
250 _______________ "3rd line green"
260 _______________
270 _______________ "4th line red"
```
CHANGING COLORS

300 ' LOOP
310 GOTO 310

(Answers are at the end of the chapter.)

Notice that the COLOR changes used in the program of exercise 2-2
could have been done in a neat FOR-NEXT loop like this:

100 ' SET SCREEN
110 PMODE 1, 1
120 PCLS
130 SCREEN 1, 0
200 ' DRAW COLORFUL LINES
210 FOR Z = 1 TO 4
220 W = 4*Z
230 COLOR Z, 1
240 LINE (0, W) - (128, W), PSET
250 NEXT Z
300 ' LOOP
310 GOTO 310

CHANGING COLORS

It would be better still if you also could see each color on all four back­
grounds. We have used only green for the background so far. Can you think
of a way to see all four backgrounds (Hint: Maybe another FOR-NEXT loop;
also, a time delay to observe each background with its colored lines before
going on to the next one)? To do this add

115 FOR B = 1 TO 4
260 FOR N = 1 TO 500: NEXT N
270 NEXT B

and change

120 PCLS B
230 COLOR Z, B

The program now looks like this:

100 ' SET MODE AND COLOR
110 PMODE 1, 1
When the program is run, you first see a green background with colored lines appearing in this order: yellow, then blue, then red. The green line does not show up on the green background:

The background then changes to yellow, and you see
The background then changes to blue, and you see

![Diagram of blue background with colorful lines]

Last, the background changes to red, and you see

![Diagram of red background with colorful lines]

This program demonstrates all the color combinations for PMODE 1, 1 and SCREEN 1, 0. You can change to SCREEN 1, 1 and see all the combinations of buff, cyan, magenta, and orange.

You now can draw vertical and horizontal lines anywhere on the screen. What happens with diagonal lines? The positions of the four corners of the screen are

```
(0, 0) - (255, 0)
(0, 191) - (255, 191)
```

Here is a program using PMODE 0, 1 with SCREEN 1, 1. It uses LINE statements to draw the diagonals from the lower left to the upper right and from the upper left to the lower right corners.
30 COLORING LINES AND CIRCLES

100 ' DRAW DIAGONALS
110 PMODE 0, 1
120 PCLS
130 SCREEN 1, 1
200 ' DRAW 2 LINES
210 LINE (0, 191) - (255, 0), PSET
220 LINE (0, 0) - (255, 191), PSET
300 ' LOOP
310 GOTO 310

The program draws this display:

![Red diagonal lines and green background](image)

You also might try the program with

SCREEN 1, 0

or

120 PCLS 5
125 COLOR 0, 5
130 SCREEN 1, 1

or

120 PCLS 1
125 COLOR 0, 1
130 SCREEN 1, 0

or one of the many other color combinations.

---

CONNECTING LINES

You have seen that horizontal, vertical, and diagonal lines can be drawn using the LINE statement. You also can connect the lines to make a geometric figure. The color computer lets you use the LINE statement in a different way when drawing lines that connect.
It is used like this:

```
100  ' SET SCREEN
110  PMODE 1, 1
120  PCLS
130  COLOR 3, 1
140  SCREEN 1, 0
200  ' DRAW CONNECTING LINES
210  LINE (98, 5) - (158, 5), PSET
220  LINE -(158, 96}, PSET - draw from (158,5) to (158, 96)
300  ' LOOP
310  GOTO 310
```

Exercise 2-3

Run the preceding program, then

a. describe the display colors, lines, placement, etc.

b. insert this line:

```
215  COLOR 2, 1
```

Rerun the program and describe the change in the display caused by line 215.

(Answers are at the end of the chapter.)

You could add more lines to the program that you ran in exercise 2-3 to complete a rectangle. That program has already drawn the top and right sides of the rectangle. Delete line 215 from part b of exercise 2-3.
Add these lines:

230 LINE -(98, 96), PSET
240 LINE -(98, 5), PSET

Let's use the lower half of the screen to draw the rectangle in this program.

100 ' SET SCREEN
110 PMODE 3, 1
120 PCLS
130 COLOR 3, 1
140 SCREEN 1, 0
200 ' CONNECT LINES
210 LINE (98, 5) - (158, 5), PSET
220 LINE -(158, 96), PSET
230 LINE -(98, 96), PSET
240 LINE -(98, 5), PSET
300 ' LOOP
310 GOTO 310

Exercise 2-4

Write a program to draw this figure, using \( \text{LINE } (X_1, \ Y_1) - (X_2, \ Y_2) \) once and \( \text{LINE } - (X, \ Y) \) several times.

Don't lift the crayon off the paper. Use any graphics mode and colors that you want. Use these row and column numbers: rows 15 and 80, columns 80 and 170.

100 ' SET SCREEN
110 PMODE

DRAWING BOXES

You've seen how the LINE statement can be used to draw boxes (or rectangles) more quickly than setting individual points with PSET. There is even a quicker way to draw a box using an additional parameter with the LINE statement. Think of a box with these coordinates for corners:

\[(X_1, Y_1)\] \[(X_2, Y_2)\] \[(X_4, Y_4)\] \[(X_3, Y_3)\]

Now use the LINE statement

\[
\text{LINE } (X_1, Y_1) - (X_3, Y_3), \text{ PSET, B}
\]

Points \(X_2, Y_2\) and \(X_4, Y_4\) can also be used. Whatever choice is made, the
coordinates must be at opposite corners. That one LINE statement will draw the entire box.

**Example:** LINE (20, 10) - (60, 50), PSET, B

Here is a program using the BOX option with rows 10 and 50 and columns 20 and 60. It draws the box pictured in Figure 2-4.

```plaintext
100 ' SET SCREEN
110 PMODE 3, 1
120 PCLS
130 SCREEN 1, 0
200 ' DRAW BOX
210 LINE (20, 10) - (60, 50), PSET, B ---the BOX option
220 GOTO 220
```

Other LINE statements that could be used to draw the same box are

```plaintext
210 LINE (60, 10) - (20, 50), PSET, B
or
210 LINE (60, 50) - (20, 10), PSET, B
or
210 LINE (20, 50) - (60, 10), PSET, B
```

You now have a one-liner that will draw a box. Here is a program that randomly places boxes on the screen.

```plaintext
100 ' SET SCREEN
110 PMODE 3, 1
120 PCLS
130 SCREEN 1, 0
```

![Figure 2-4. Drawing a box.](image)
FILLING BOXES WITH COLOR

200  'DRAW BOXES RANDOMLY
210  FOR Z = 1 TO 5
220    I = RND (50): J = RND (128)
230    LINE (J, I) - (J+I, J+I), PSET, B
240  NEXT Z
300  'LOOP
310  GOTO 310

Run the program several times and see the many patterns produced. Change the FOR-NEXT ending value (now 5) to create the number of rectangles pleasing to you. Then add these two lines to the program to produce one of four random colors when drawing the rectangles.

213  C = RND (4)
216  COLOR C, 1

Notice that sometimes you don’t see five rectangles because a green rectangle doesn’t show on a green background. You might add

215  IF C = 1 THEN 213

You can change PMODE, PCLS, C, and COLOR to produce a wide variety of color patterns on the screen.

FILLING BOXES WITH COLOR

You should learn one more option before leaving the LINE statement:

LINE (X1, Y1) - (X3, Y3), PSET, BF

The **F** option lets you fill the box with the current foreground color.

You know that the middle of the screen is approximately at position 128, 96. Here is a program that draws a box and fills it with color. The corner positions of the box are 64 positions above, below, left, and right of the center of the screen.

110  PMODE 4, 1
120  PCLS
130  SCREEN 1, 0
Let's return to the program that placed boxes randomly on the screen. Add the F option to line 150. Before you run the program, revise it to use the second set of colors (SCREEN 1, 1).

```
100 'SET SCREEN
110 PMODE 3,1
120 PCLS
130 SCREEN 1,1 --> changed to second color set
200 'DRAW BOXES AND COLOR
210 FOR Z = 1 TO 5
213 C = RND(4) + 4 --> to give correct color
215 IF C = 5 THEN 213 --> trap background color
216 COLOR C,5
220 I = RND(50): J = RND(128)
230 LINE(J,I) - (J+I, J+I), PSET, BF --> fill it up
240 NEXT Z
300 'LOOP
310 GOTO 310
```

Randomly filled, colored boxes appear. Sometimes they overlap; sometimes they erase all or parts of other boxes. Change the end value of the FOR-NEXT loop for some variation of the picture.

**Figure 2-5. Filling a box with color.**
Exercise 2.5

Now it's time for you to put the LINE statement to use in your own ways. Write a program that will put colored rectangles in different areas of the screen in approximately this way:

![Diagram of colored rectangles in different areas of the screen]

(Sample answer is at the end of the chapter.)

**DRAWING CIRCLES**

Straight lines and rectangles aren't all that the color computer can draw. It also can draw circles. The CIRCLE statement can take many different forms. Let's consider the simplest form first:

\[
\text{CIRCLE}(X,Y), R
\]

- the position of the center
- the circle's radius
- comma

The X coordinate may be any number from 0 through 255. The Y coordinate may range from 0 through 191. One unit for R is equal to one point, or screen position.

**Example:**

\[
\text{CIRCLE (128, 96), 20}
\]

→ would draw a circle with center at the center of the screen with a radius 20 points long.

The following short program draws a circle at the center of the screen with a radius of 40:

100 'SET SCREEN
110 PMODE 0, 1
120 PCLS
130 SCREEN 1 0
COLORING LINES AND CIRCLES

200 'DRAW CIRCLE
210 CIRCLE (128,96),40 ← draw it round
300 'LOOP
310 GOTO 310

Run the program and study the circle.

On the screen we see

![Figure 2-6. Drawing a circle.](image)

1. change line 110 to PMODE 2,1. Run the program again. See if you can see a difference in the circle's appearance.

2. change line 110 to PMODE 4,1. Run the program again. Can you see a further difference?

We noticed that in case 1 the lines appear to be thinner than in the original program. The circle was not very smooth. In case 2 (the high-resolution mode) the lines forming the circle are thinner yet, and the circle is smoother.

If the center of the circle is moved to 64,48 and the radius changed to 60, the top of the circle is flattened like this:

![Image of flattened circle](image)

If the center of the circle is changed to 240,144, the right and lower parts of the circle are flattened like this:
Exercise 2-6

Write a program to place 15 circles in randomly chosen places on the screen. Make the choice of radii and color random also.

100 'SET SCREEN
110 PMODE 3,1
120 PCLS 5
130 SCREEN 1,1
200 'DRAW 15 COLORED CIRCLES
210 FOR X = ________ TO ________
220   C = ____________________ → a random color
230   A = ____________________ → a random column
240   B = ____________________ → a random row
250   R = ____________________ → a random radius
260   COLOR _________________
270   CIRCLE _________________
280   NEXT X
300 'WAIT AWHILE THEN GO BACK
310 FOR W = 1 TO 500: NEXT W
320 GOTO 120

(Answers are at the end of the chapter.)

Summary

In this chapter you learned all about drawing straight colored lines and rectangles. You even learned how to draw colored circles of various sizes and colors. Lines were drawn with the new statements that you learned more quickly than before and were much easier to program. You learned that

- horizontal, vertical, and diagonal lines can be drawn by specifying their endpoints, such as
COLORING LINES AND CIRCLES

LINE($X_1, Y_1$) - ($X_2, Y_2$),PSET

- a second line can be connected to the end of a previous line by specifying the endpoint of the second line, such as

LINE - ($X_3, Y_3$),PSET

- a complete rectangle can be drawn by specifying the position of opposite corners, such as

LINE($X_1, Y_1$) - ($X_3, Y_3$),PSET,B

- the contents of a box (or rectangle) can be filled with a specified color, such as

LINE($X_1, Y_1$) - ($X_3, Y_3$),PSET,BF

- a circle can be drawn by specifying the position of its center and the length of its radius, such as

CIRCLE(X,Y),R

In addition to all this, you had more practice in changing graphics modes, changing foreground and background colors, and moving geometric figures around the screen.

Just for the fun of it, show yourself how much you learned by working the exercises in the chapter test.

Chapter Test

1. Describe the lines drawn by the execution of

LINE(0,96) - (255,96),PSET

2. Fill in the coordinates in the LINE statement that would draw a line from the upper right corner to the lower left corner of the video display.

LINE( _____ , _____ ) - ( _____ , _____ ),PSET
3. Give the LINE statement that would draw a rectangle using rows 35 and 99 with columns 75 and 105.

LINE ________________________________

4. Show a second way to arrange the coordinates to do the same thing as exercise 3 above.

LINE ________________________________

5. Suppose a program has just drawn a line by the statement

LINE(50,10) - (90,10),PSET

You want to add another line from the end of this line so that it will go down 40 positions. Fill in line 160 to do it.

160 LINE ________________________________

6. Add two more lines to 160 to complete the rectangle started in exercise 5.

170 ________________________________
180 ________________________________

7. Give one LINE statement that would draw the same rectangle as the four statements in lines 150, 160, 170, and 180 of exercises 5 and 6.

LINE ________________________________

8. Complete the COLOR and LINE statements in this program to fill the box red.

100 PMODE 3,1
110 PCLS 1
120 SCREEN 1,0
130 COLOR ________________________________
140 LINE ________________________________
150 GOTO 150

9. Which lines would you change and how would you change them in
order to create a green rectangle on a blue background in the program of exercise 8?

10. Change lines in the program of exercise 8 to create a cyan rectangle on a buff background.

11. Complete the program to create this figure at the center of the screen.

```
100 PMODE
110 PCLS
120 SCREEN
130 COLOR
140 LINE
150 COLOR
160 LINE
170 GOTO 170
```

12. Give the necessary changes to your program of exercise 11 to create this change in the figure. Use the same rows and columns.
13. Complete this program to draw 12 concentric circles in yellow on a green background with random radii up to 60 positions long. Put the circles in the center of the screen.

100 PMODE 3,1
110 PCLS
120 SCREEN
130 FOR X = TO
140 R =
150 COLOR
160 CIRCLE
170 NEXT X
180 FOR W = 1 TO 500: NEXT W
190 GOTO 120

Answers to Exercises Within the Chapter

Exercise 2-1

10 PMODE 1,1
20 PCLS
30 SCREEN 1,0
40 LINE(127,85) - (127,105),PSET
50 GOTO 50
Exercise 2-2

220 COLOR 2,1
230 LINE(0,8) - (128,8),PSET
240 COLOR 1,1
250 LINE(0,12) - (128,12),PSET
260 COLOR 4,1
270 LINE(0,16) - (128,16),PSET

Exercise 2-3

a. A horizontal blue line was drawn close to the top of the screen near the middle columns. A connecting vertical blue line was drawn downward on the right side to about the middle row.

b. The vertical line drawn was yellow instead of blue.

Exercise 2-4

This is only one way to do it. Yours may be entirely different. You should try several versions.

110 PMODE 4,1
120 PCLS
130 SCREEN 1,0
210 LINE(80,15) - (170,15),PSET
220 LINE -(170,80),PSET
230 LINE -(80,15),PSET
240 LINE -(80,80),PSET
250 LINE -(170,15),PSET
260 LINE -(170,80),PSET
270 LINE -(80,80),PSET

Exercise 2-5

Sample answer

100 PMODE 0,1
110 PCLS
120 SCREEN 1,1
130 LINE(30,20) - (45,80),PSET,BF
140 COLOR 7,5
150 LINE(20,150) - (100,160),PSET,BF
160 LINE(140,60) - (200,90), PSET,BF
170 COLOR 6,5
Exercise 2-6

210  FOR X = 1 TO 15
220    C = RND(4) + 4  (a random color)
230    A = RND(127) + 64 (a random column)
240    B = RND(96) + 48 (a random row)
250    R = RND(40) (a random radius)
260    COLOR C,5
270    CIRCLE(A,B),R

Answers to Odd-Numbered Exercises of Chapter Test

1. A line would be drawn halfway down the screen from one side (left) to the other side (right).

3. LINE(75,35) - (105,90),PSET,B
   or
   LINE(75,90) - (105,35),PSET,B
   or other combinations

5. 160 LINE -(90,50},PSET

7. LINE(50,10) - (90,50),PSET,B

9. 110  PCLS 3
    130  COLOR 1,3

11. 100  PMODE 3,1 (or PMODE 1,1)
    110  PCLS 5
    120  SCREEN 1,1
    130  COLOR 7,5
    140  LINE(88,76) - (168,116),PSET,BF
    150  COLOR 8,5
    160  LINE(108,56) - (148,136),PSET,BF

13. 100  PMODE 3,1
    110  PCLS 1
    120  SCREEN 1,0
    130  FOR X = 1 TO 12
    140    R = RND(60)
    150    COLOR 2,1
    160    CIRCLE (128,96),R
More Circles, PAINT, and POINT

You ended Chapter 2 by learning a little about the CIRCLE statement, using the center position and the length of the radius. You changed colors for the circle with the COLOR statement.

In this chapter, you'll meet an expanded CIRCLE statement and learn new ways to color all, or parts, of geometric figures. You will learn

• to set a circle's color within the CIRCLE statement
• to stretch the circle into an ellipse with the CIRCLE statement
• to draw arcs (or parts) of a circle
• to PAINT portions of geometric figures that you have drawn on the screen
• to detect the color of a point on the screen with the PPOINT statement

MORE ABOUT CIRCLES

In the last chapter, you were introduced to the CIRCLE statement. In addition to specifying the position of the center of the circle and the length of the radius, you can also specify the color of the circle.

![CIRCLE(X,Y),R,C](same as in last chapter) specifies one of the available colors

If the C parameter is omitted (as in the last chapter), the foreground color is used to draw the circle. In the last chapter, we used this sequence to draw a red circle on a green background:
You can omit line 130 when including the circle's color in the CIRCLE statement. The following line would duplicate the results of lines 130 and 140 in the program.

```
140 CIRCLE(128,96),20,4
```

Other colors could be produced by

```
140 CIRCLE(128,96),20,3
140 CIRCLE(128,96),20,2
```

If 1 is used for the circle's color, it would blend into the green background and not be visible.

---

**SQUEEZING A CIRCLE**

In addition to setting the circle's color within the CIRCLE statement, you can also specify the eccentricity of the circle. In other words, you can stretch the circle out of its round shape

```
  this way          or          this way
```

To do this, a new parameter is used:

```
CIRCLE(X,Y),R,C,E
```

same as before

eccentricity (called hw in the TRS-80 Color Computer Manual)
The height/width ratio (E) may be any number from 0 through 255. You will not want to go nearly as high as 255, but it is possible to do so without a syntax error. If E is not specified (as in previous circles), a ratio of 1 is used. The height is equal to the width.

When the color parameter (C) is not given in the CIRCLE statement and the height/width ratio is used, an extra comma must be used to tell the computer that the color has been omitted. If this was not done, the computer couldn’t tell the color from the eccentricity desired.

Example:

\[
\text{CIRCLE (X, Y), R, , E}
\]

two commas here to indicate that the color parameter has been omitted—the foreground color is used to draw the circle

Enter and run the following program. The result is shown in Figure 3-1.

```
100  ' SET SCREEN
110  PMODE 4, 1
120  PCLS
130  SCREEN 1, 0
200  ' SQUEEZE THE CIRCLE
210  CIRCLE (20, 96), 15, , 0.5
220  CIRCLE (80, 96), 15, , 1
230  CIRCLE (140, 96), 15, , 2
240  CIRCLE (220, 96), 15, , 5
300  ' LOOP
310  GOTO 310
```

Figure 3-1. Squeezing a circle.
Exercise 3-1

Now let's go to PMODE 3, 1 and use 0.5, 1, and 2 for the height/width ratio. Write a program to display cyan, magenta, and orange circles on a buff background, using these ratios. Put all the circles on the screen at the same time.

```
100 ' SET SCREEN
110 PMODE ____________
120 PCLS
130 SCREEN ____________
200 ' COLORED CIRCLES
210 ____________________
220 ____________________
230 ____________________
300 ' LOOP
310 GOTO 310
```

(Answer is at the end of the chapter.)

You can use the high-resolution mode to write a program to draw circles that all have the same center, all have a radius of 20, and have height/width ratios of 0, .1, .2, .3, .4, .5, .6, .7, .8, .9, 1, 1.1, 1.2, 1.3, 1.4, and 1.5. Let's leave each circle on the screen during a time delay (W = 1 TO 200). Then we'll erase the current circle before drawing a new one and have the program loop back to repeat the sequence over and over.

```
100 ' SET IT UP
110 PMODE 4, 1
120 PCLS
130 SCREEN 1, 0
200 ' CHANGE COLOR, THEN ERASE
210 FOR X = 0 TO 1.5 STEP 0.1
220 CIRCLE (128, 96), 20, , X
230 FOR W = 1 TO 200: NEXT W
240 CIRCLE (128, 96), 20, 0, X
250 NEXT X
300 ' DO AGAIN
310 GOTO 210

use foreground color to draw
use background color to erase
```
The following program draws 12 circles with the following random parameters:

- location of center
- color
- radius
- height/width ratio

Our choice of colors are buff, cyan, magenta, and orange. We are using cyan as the background color.

```
100  'SET UP
110  PMODE 3, 1
120  PCLS 6
130  SCREEN 1, 1
200  'GET RANDOM VALUES
210  FOR Z = 1 TO 12
220      X = RND (128) + 64  \( \text{chooses the center} \)
230      Y = RND (96) + 48
240      R = RND (48)
250      C = RND (4) + 4  \( \text{5, 6, 7, or 8 for color} \)
260      IF C = 6 THEN 250  \( \text{refuses cyan} \)
270      E = RND (20)/10  \( \text{gives range 0.1–2} \)
300  'DRAW CIRCLE AND GO BACK
310      CIRCLE (X, Y), R, C, E
320      NEXT Z
400  'LOOP AWHILE
410  GOTO 410
```

The patterns produced are unpredictable. The following addition to the program will occupy you for long periods of time. With this addition, you can repeat the sequence with changing patterns.

```
410  FOR W = 1 TO 500: NEXT W
420  GOTO 120
```

**DRAWING ARCS**

The versatile CIRCLE statement has two more parameters that enable you to draw only portions of circles (arcs) if desired. Here are all the CIRCLE options!
B and S may be numerical expressions from 0 through 1. If the B parameter is omitted, the computer uses 0. If the S parameter is omitted, the computer uses 1. What do the 0 and 1 mean?

To locate a position of zero, think of the 3 o'clock position on the face of a clock.

Other positions (decimals between 0 and 1) are located as follows:

![Figure 3-2. Arc end point locations.](image)

**Exercise 3-2**

On the following circles, mark an o for the starting position and an x for the ending position for these CIRCLE statements.

a. CIRCLE (128, 96), 20, 2, 1, 0.2, 0.5
b. CIRCLE (128, 96), 10, 3, 1, 0.5, 0.8
c. CIRCLE (128, 96), 15, 4, 0.4, 1

(Answers are at the end of the chapter.)

When the starting point (B) is equal to the ending point (S), the computer draws a complete circle. It also draws a complete circle if the starting and end points are omitted (as in earlier circles). If the starting point is greater than
the ending point, the computer will draw the circle from the starting point, going right through the zero point to the ending point.

Example:

```
CIRCLE (128, 96), 30, 1, 0.5, 0.2
```

To use the start and end points in order to draw an arc, you must specify the height/width ratio (for a normal circle, this value is 1).

Exercise 3-3

Examine the following statements. If they are executable, fill in the arc that will be drawn in the accompanying sketch. If the statement is not executable, mark a large X over the sketch.

a. `CIRCLE (128, 96), 10, , 1, 0.2, 0.5`
b. `CIRCLE (128, 96), 25, , 1, 0.4, 0.3`
c. `CIRCLE (128, 96), 20, , 0.3, 0.9`
d. `CIRCLE (128, 96), 10, , 2, 0, 0.9`

Study this program. A sketch of the screen is shown as it looked when the program was executed. Run the program to see if you get the same result.

```
100  ' SET UP
110  PMODE 4, 1
120  PCLS
130  SCREEN 1, 0
140  B = 0.65: S = 0.85
200  ' DRAW ARCS
210  FOR R = 20 TO 50 STEP 10
```
MORE CIRCLES, PAINT, AND POINT

Figure 3-3. Drawing arcs.

220 CIRCLE (128, 96), R, 1, B, S  → top arcs
230 CIRCLE (128, 96), R, 1, B-.5, S-.5 → bottom arcs
240 NEXT R
300 ' LOOK AT THEM
310 GOTO 310

PAINTING

Get out your crayons or paint brushes. The PAINT statement follows. You can color geometric figures in some imaginative ways by using

PAINT (X, Y), C, B

start painting from this point

use this color on your brush

PAINT until you come to this border color

The "paint brush" starts at point X, Y and "paints" in all directions, using color C until it comes to the border color B. If the computer comes to

Figure 3-4. Painting a picture.
border colors that differ from the specified border color, it will paint over and beyond that border. Suppose that you have these rectangles on your screen:

![Diagram of rectangles with border colors]

You then execute

```
PAINT (110, 105), 8, 7
```  

The display in Figure 3-4 would be produced.

Study this program and the sketch of the result produced when it is executed. Note: The BREAK key is used to stop the program. It has no effect while the PAINT statement is being executed. Hold down the BREAK key until the PAINTing is completed.

```
100 'SET UP
110 PMODE 4, 1
120 PCLS
130 SCREEN 1, 0
200 'DRAW DIAGONALS
210 LINE (0, 0) - (255, 191), PSET
220 LINE (255, 0) - (0, 191), PSET
250 'DRAW AND PAINT
260 CIRCLE (128, 96), 40
```
Exercise 3-4

Show which line of the previous program should be changed (and how to change it) to produce these displays:

a. 

b. 

c. 

black 

green
If you start painting inside a figure that is not closed, the paint will "leak" and spill over everything.

Example:

Leave the top of a bucket of paint open as in this program.

```
100  ' SET UP
110  PMODE 1, 1
120  PCLS
130  SCREEN 1, 1
140  COLOR 6, 5
200  ' DRAW BUCKET
210  LINE (100, 64) - (100, 126), PSET
220  LINE - (156, 126), PSET
230  LINE - (156, 66), PSET
250  ' FILL WITH PAINT
260  PAINT (102, 124), 8, 6
300  ' DON'T CRY OVER SPILT PAINT
310  GOTO 310
```
Watch the paint fill the bucket and overflow. The whole screen will be filled with paint because the top is not on the bucket.

RANDOM PAINTINGS

Concentric circles all have the same center. Here is a program using PMODE 3, 1 that draws concentric circles with their centers near the center of the screen. The circles will be five units apart (i.e. radii 5, 10, 15, 20, ..., 90). The circles are drawn in random colors from the choices: buff, cyan, magenta, and orange. A buff background is used.

100 'SET UP
110 PMODE 3, 1
120 PCLS
130 SCREEN 1, 1
200 'VARY RADIUS AND COLOR
210 FOR R = 5 TO 90 STEP 5
220 C = RND (4) + 4
230 CIRCLE (128, 96), R, C
240 NEXT R
300 'LOOP
310 GOTO 310

After you have run this program several times to see the random patterns, you may wish some additional variation. It would be interesting to add the PAINT statement to the program. To do so, delete line 310 and add these lines to fill the circles from the center outward. We'll PAINT with the same color set, choosing the PAINT and border colors randomly.

Add or change these lines:

310 C = RND (4) + 4
320 B = RND (4) + 4
330 PAINT (128, 96), C, B
340 GOTO 310

Now you get some really colorful patterns with which to play. When you tire of this circular randomness, proceed to exercise 3-5 and try something a little more square.
Exercise 3-5

Alter the previous program to draw and paint rectangles instead of circles. Use the LINE statement with the box option to do it. Use R in the FOR-NEXT loop to vary the corner positions of the boxes.

100 'SET UP
110 PMODE 3, 1
120 PCLS
130 SCREEN 1, 1
200 'DRAW AND PAINT
210 ...start FOR-NEXT
220 ...random color
230 ...specify color
240 ...draw box
250 ...end FOR-NEXT
260 ...random paint
270 ...random border
280 ...PAINT from center
300 'GET MORE PAINT
310 GOTO 260

(Answers are at the end of the chapter.)

After all that serious work, it's time to play awhile. You know all about arcs and circles. See if you can create a cartoon character in the successive steps that follow:

Large CIRCLE with h/w ratio = 1.5

Add two small circles.

Add two small circles with h/w ratio = 1.8; use arcs.
Add two arcs. Add two more arcs. Add anything else you want.

Write your program here:

Here's the way we did it.

100 ' SET SCREEN
110 PMODE 4, 1
120 PCLS
130 SCREEN 1, 1
200 ' DRAW ARCS AND CIRCLES
210 CIRCLE (128, 96), 50, , 1.2
220 CIRCLE (110, 80), 3
230 CIRCLE (136, 80), 3
240 CIRCLE (120, 99), 6, , 1.3, .15, .75
250 CIRCLE (120, 120), 10, , 1, .1, .4
260 CIRCLE (110, 75), 8, , 1, .65, .85
270 CIRCLE (136, 75), 8, , 1, .65, .85
280 CIRCLE (70, 96), 8, , 1.8, 0, .9
290 CIRCLE (186, 96), 8, , 1.8, .55, .45
300 ' SIT ON BOX
310  LINE (100, 155) - (156, 170), PSET, B ← a box to sit on
320  GOTO 320

GETTING THE POINT

Another statement useful in programming the color computer is

PPOINT (X, Y)

This statement tests the color of the specified point X, Y.

Example:
Suppose that you had colored a box with

100  PMODE 3, 1
110  PCLS
120  SCREEN 1, 0
130  LINE (20, 80) - (120, 100), PSET, BF

When those lines were executed, the display would show

![Red box with green background]

The following lines would print the color value of the specified points.

140  PRINT PPOINT (10, 80) ← This would print 1 since the point (10, 80) is in the green area.
150  PRINT PPOINT (30, 96) ← This would print 4 since the point (30, 96) is in the red area.

When the PRINT statements in lines 140 and 150 are executed, the computer returns to the text screen to show the results. This is done automatically when a PRINT statement is executed regardless of the current mode of the display screen.
Example:

100 'SET UP
110 PMODE 3, 1
120 PCLS
130 SCREEN 1, 1

200 'FILL 2 BOXES, DIFFERENT COLORS
210 LINE (0, 0) - (255, 95), PSET, BF
220 COLOR 6, 8
230 LINE (0, 96) - (255, 191), PSET, BF

300 'WAIT, CLEAR SCREEN
310 FOR W = 1 TO 500: NEXT W
320 CLS

400 'PRINT COLOR AT POINTS
410 PRINT PPPOINT (128, 40)
420 PRINT PPPOINT (128, 120)

If this program is executed, the graphics briefly appear on the screen.

If this program is executed, the graphics briefly appear on the screen.

Line 320 then clears the screen for text, and the PRINT statements in lines 410 and 420 print the color codes of the points specified.

Do you think that the previous color graphics are still in the computer's memory? You can find out by adding the following lines and executing the program again. The computer will alternately display the text screen and the graphics screen.

430 FOR W = 1 TO 500: NEXT W
440 SCREEN 1, 1
450 FOR W = 1 TO 500: NEXT W
460 GOTO 320

---

- Point (128, 40) is orange
- Point (128, 120) is cyan
Yes, both the graphics and text are in the computer. The graphics data are kept in a special section of memory (see Appendix B for a TRS-80 Color Computer memory map).

PPOINT can be used in other ways also. Suppose that you have three color swatches on the screen.

You could test the color of points on the screen for a given color and have the computer perform some specified action if a given color is found. For instance, you could have the computer change all the red points to green (effectively erasing the points).

Enter and run this program.

```
100 ' SET UP
110 PMODE 1, 1
120 PCLS
130 SCREEN 1, 0
200 ' DRAW AND COLOR
210 LINE (20, 10) - (80, 20), PSET, BF ←red box
220 COLOR 2, 1
230 LINE (100, 80) - (140, 100), PSET, BF ←yellow box
240 COLOR 3, 1
250 LINE (200, 130) - (220, 150), PSET, BF ←blue box
300 TEST POINTS AND RESET
310 FOR Y = 10 TO 191 STEP 2
320 FOR X = 20 TO 255 STEP 2
```
The program takes a long time to search the complete screen. Press the BREAK key after the red box disappears in order to end the search.

To change colors of all the boxes, make these changes.

1. Alter line 330 of the program so that the red box is changed to yellow.

   330  IF PPOINT (X, Y) = 4 THEN PSET (X, Y, 2)

2. Add line 333 to change the yellow box to blue.

   333  IF PPOINT (X, Y) = 2 THEN PSET (X, Y, 3)

3. Add line 336 to change the blue box to red.

   336  IF PPOINT (X, Y) = 3 THEN PSET (X, Y, 4)

Again, it takes a long time for the program to search for the color points. Sit back and wait until all three rectangles have changed colors. You also could change the program to loop back to line 310 to make ever-changing color boxes.

Exercise 3-6

To finish this chapter, write a program that will draw 10 concentric circles with randomly chosen colors from cyan, magenta, and orange. Use a
buff background. Then check the color of the points on the screen. If a point is cyan, change it to magenta. If a point is magenta, change it to orange. If a point is orange, change it to cyan.

100 'SET UP
110 _____________________________
120 _____________________________
130 _____________________________
200 'CIRCLES WITH RANDOM COLORS
210 _____________________________
220 _____________________________
230 _____________________________
240 _____________________________
250 _____________________________
300 'CHECK POINTS AND RESET
310 _____________________________
320 _____________________________
330 _____________________________
340 _____________________________
350 _____________________________
360 _____________________________
370 _____________________________
400 'LOOP
410 _____________________________

(Answers are at the end of the chapter.)

Summary

Using only a handful of new statements in this chapter, you learned many new color graphic programming techniques. You learned

- to specify the color for a circle to be drawn with

\[
\text{CIRCLE (X, Y), R, C}
\]

center \quad \text{radius} \quad \text{color}
• to specify the eccentricity (or height/width ratio) of a circle with

\[
\text{CIRCLE (X, Y), R, C, E}
\]

*any numerical expression (0-255)*

**Examples:**

\[
\begin{array}{cccc}
\text{E = .5} & \text{E = 1} & \text{E = 2} & \text{E = 5} \\
\text{\includegraphics[width=0.2\textwidth]{circle05}} & \text{\includegraphics[width=0.2\textwidth]{circle10}} & \text{\includegraphics[width=0.2\textwidth]{circle20}} & \text{\includegraphics[width=0.2\textwidth]{circle50}}
\end{array}
\]

• to specify a portion of a circle (an arc) to be drawn with:

\[
\text{CIRCLE (X, Y), R, C, E, B, S}
\]

\[
\begin{array}{c}
\text{begin here} \\
\text{stop here}
\end{array}
\]

\[
\text{B and S may be in the range of 0 through 1 on the circle as:}
\]

\[
\text{\includegraphics[width=0.2\textwidth]{arc05}} \quad \text{\includegraphics[width=0.2\textwidth]{arc50}} \quad \text{\includegraphics[width=0.2\textwidth]{arc75}}
\]

• to use commas to show parameters that are not expressed

\[
\text{CIRCLE (X, Y), R, , E, B, S}
\]

\[
\text{color not expressed; last foreground color used}
\]

You also learned that

• the eccentricity parameter must be given if the arc starting and end points are used in the CIRCLE statement

• areas can be "painted" by specifying the color to be used for painting and the border color where the painting is to stop:

\[
\text{PAINT (X, Y), C, B}
\]

\[
\begin{array}{c}
\text{start here} \\
\text{color for paint} \\
\text{paint to this border}
\end{array}
\]
• the color of a specified point on the screen can be determined by the PPOINT statement.

See if you have remembered how to use these new tools by answering the questions in the chapter test.

Chapter Test

1. State the color of the circle drawn by the execution of these statements:

   100 PMODE 1, 1
   110 PCLS
   120 SCREEN 1, 0
   130 CIRCLE (128, 96), 10, 3

   The circle is ___________ color on a ___________ background.

2. If line 130 in the program in exercise 1 were changed to

   130 CIRCLE (20, 40), 15, 2

   the circle drawn would be ______ ________.
       (larger, smaller)

3. If you wanted to draw a green circle on a black background, complete these lines to do so:

   100 PMODE __________
   110 PCLS
   120 SCREEN __________
   130 CIRCLE (128, 96) __________

4. Draw the shape that would be displayed if the CIRCLE statement in line 130 of exercise 3 had been

   130 CIRCLE (128, 96), 10, , 0.5
5. To stretch the circle into an ellipse that is higher than wide, the eccentricity parameter should be _______ than one.
   (greater, less)

6. Sketch the “circle” that would be produced by

   100  PMODE 4, 1
   110  PCLS
   120  SCREEN 1, 0
   130  CIRCLE (128, 96), 20,, 2

![Circle Diagram]

7. Suppose that you want to draw an arc of a circle whose radius is 10 in a blue color on a green background. However, you only want to show this three-quarters of the circle.

![Arc Diagram]

Complete these statements to draw the specified arc.

   100  PMODE 3, 1
   110  PCLS
   120  SCREEN (128, 96)
   130  CIRCLE (128, 96)
   140  GOTO 140

8. How would you complete line 130 of exercise 7 if you wanted to draw only the one-quarter of the circle not drawn by the program in exercise 7?

   130  CIRCLE (128, 96)
9. Darken the arc drawn by the execution of these statements.

```
100  PMODE 3. 1
110  PCLS
120  SCREEN 1, 1
130  CIRCLE (128, 96), 10, 7, 1, .5, .25
140  GOTO 140
```
12. What line would you add to change the program in exercise 11 to give this display?

Add

13. What would the final printed display look like if line 140 of exercise 10 were

140 PRINT PPOINT (125, 90)

14. Write a program to put multicolored vertical and horizontal lines on the display and then PAINT from random points to random, colored borders.
Answers to Exercises Within the Chapter

Exercise 3-1

110  PMODE 3, 1
120  PCLS
130  SCREEN 1, 1
210  CIRCLE (20, 96), 16, 6, .5
220  CIRCLE (105, 96), 15, 7, 1
230  CIRCLE (210, 96), 15, 8, 2
310  GOTO 310

Exercise 3-2

Exercise 3-3

Exercise 3-4

a. 270 PAINT (128, 50), 1, 1
   - less than 56
b. 270 PAINT (128, 138), 1, 1: PAINT (85, 96), 1, 1
   - greater than 136
   - less than 88
c. 270 PAINT (128, 50), 1, 1: PAINT (128, 138), 1, 1:
   PAINT (170, 96), 1, 1
   - greater than 168

Exercise 3-5

210  FOR R = 5 TO 90 STEP 5
220  C = RND (4) + 4
230  COLOR C, 5
240  LINE (128-R, 96-R) - (128+R, 96+R), PSET, B
250 NEXT R  
260   C = RND (4) + 4  
270   B = RND (4) + 4  
280   PAINT (128, 96), C, B

Exercise 3-6

110   PMODE 1, 1  
120   PCLS  
130   SCREEN 1, 1  
210   FOR R = 10 TO 55 STEP 5  
220   C = RND (4) + 4  
230   IF C = 5 THEN 140  
240   CIRCLE (128, 96), R, C  
250   NEXT R  
310   FOR X = 72 TO 184 STEP 2  
320   FOR Y = 40 TO 152 STEP 2  
330   IF PPOINT (X, Y) = 6 THEN PSET (X, Y, 7): GOTO 230  
340   IF PPOINT (X, Y) = 7 THEN PSET (X, Y, 8): GOTO 230  
350   IF PPOINT (X, Y) = 8 THEN PSET (X, Y, 6)  
360   NEXT Y  
370   NEXT X  
410   GOTO 410

Answers to Odd-Numbered Problems in Chapter Test

1. Circle is blue color on green background.

3. 100   PMODE 4, 1 or PMODE 2, 1 or PMODE 0, 1  
110   PCLS  
120   SCREEN 1, 0  
130   CIRCLE (128, 96), 10 whatever radius you want

5. greater than one

7. 110   SCREEN 1, 0 gives green, yellow, blue, red  
130   CIRCLE (128, 96), 10, 3, 1, 0, .75

9. 

11. donut painted orange

   buff

13. 8 OK
In past chapters you have learned to create colorful video displays in several ways (e.g. by the PSET, LINE, and CIRCLE statements). This chapter is devoted to one powerful statement that provides an even easier way to DRAW screen displays.

In this chapter, you will learn to use this DRAW statement in order to

- specify a point on the screen to start drawing
- specify in what direction and how far to draw
- change the direction of successive lines to be drawn
- return to the point of origin to draw a new line
- choose a color for drawing
- include string variables as definitions for motion commands
- include substrings for intermediate drawings within the main DRAW string
- catenate strings for drawing

---

**THE DRAW STRING**

The DRAW statement may be used to draw lines by stating the starting point, the direction that you want the line to go, and how far you want it to go. All the information is contained in a string following the word DRAW:

```
DRAW"line-defining string"
```

This defines the conditions.
In the first part of the statement, you move to the starting point on the screen without drawing anything.

```
DRAW"BM128, 96"
```

B for blank; don't draw  
M for Move to the position that follows  
X, Y coordinates of the point where you want to start

Suppose that you want to draw upward from the starting point. You would then add to the string following the starting point like this:

```
150 DRAW"BM128, 96; U30"
```

```
start at 128, 96
```

```
draw up this many positions
```

The semicolon before the letter U is optional. It helps to separate visually the commands given.

If the DRAW statement of line 150 were executed, you would see

```
a line going from the center of the screen upward a distance of 30 screen positions
```

A sketch of the display produced by the execution of the following program shows three lines with different starting points drawn upward.

```
100 'SET UP THE SCREEN
110 PMODE 0, 1
120 PCLS
130 SCREEN 1, 0
140 'DRAW THREE LINES
150 DRAW"BM128, 96; U30"
160 DRAW"BM108, 96; U30"
170 DRAW"BM148, 96; U10"
180 'STAY RIGHT HERE
190 GOTO 190
```

from line 150
from line 160
from line 170
The B included at the beginning of the string in the DRAW statement is very important. The following program omits the B in one of the DRAW statements. Study the program and see if you can predict how the screen would look if the program were executed. Then, enter and run the program to confirm your prediction.

```
100  'SET THE SCREEN
110  PMODE 0, 1
120  PCLS
130  SCREEN 1, 0
140  'DRAW 2 LINES; OMIT ONE BLANK
150  DRAW"BM15, 150; U100"
160  DRAW"M240, 150; U50"  ← B omitted here
170  'LOOK AT IT NOW
180  GOTO 180
```

Here is what happens:

```
```
line drawn when you moved the "pen" from the end of the first line to the start of the second line

You can see from this example that the B BLANKS the MOVEment made by the M command in preparation for a DRAW execution. If the B is omitted, an unwanted line may appear.

The DRAW statement can make consecutive lines in different directions. You can DRAW up and then left as illustrated by this statement:

```
DRAW"BM240, 160; U50; L75"
```

This display would be produced:

```
Exercise 4-1

Complete this program to start at the center of the screen, go up 25 positions, then left 25 positions, then to the top of the screen, and then left to position 0,0. The display should look like this:

It will be hard to see that top line as the green color blends with the green border around the black background.

The program is

100 ' SET THE SCREEN
110 PMODE 0, 1
120 PCLS
130 SCREEN 1, 0
140 ' YOU PUT IN THE DRAW STATEMENT
150 DRAW
160 ' LOOP HERE AND STUDY DISPLAY
170 GOTO 170

(Answer is at the end of the chapter.)

DRAW DIRECTIONS

If the computer can DRAW up and left, it must also be able to DRAW down and right. Sure enough, the DRAW statement may be used for any of these directions:

M for move to a new position
U for draw up
L for draw left
D for draw down
R for draw right

Of course, you also must specify the number of units that you want to draw in the chosen direction. Now that you can move in four directions, it should be easy to draw a box with one DRAW statement.
Exercise 4-2

Write a program in PMODE 0, 1 that will draw a green box on a black background. Start at the lower left corner of the box (at position 10, 185). Make the box 90 units high and 150 units wide.

100 ' SET THE SCREEN
110
120
130
140 ' DRAW THE BOX
150
160 ' LOOP FOREVER
170

(Answers are at the end of the chapter.)

You could add a line in the program of exercise 4-2 that would draw a second rectangle inside the first one. We made the sides of the inner rectangle 15 units from the sides of the outer rectangle by adding:

155 DRAW“BM25, 170; U60; R120; D60; L120”

This is how our final display looked:

![Diagram of a box with a smaller box inside it.]

DRAWING SPIROLATERALS

It's play time again. Using the four directional moves of the DRAW statement, you can create spirolaterals. A spirolateral is, in some ways, similar to a spiral. However, the spirolateral is constructed with straight lines rather than a smooth curve.

You can construct an order-3 spirolateral on the screen by
1. choosing a starting point
2. selecting three distances to travel
3. draw in four different directions (we used right, down, left, and up) in sequence using the three distances of step 2 until you reach the starting point.

*Example:*

1. starting point: 60, 30
2. distances: 20, 30, and 50
3. DRAW“BM60, 30; R20; D30; L50” ← 1st 3 moves
   DRAW“U20; R30; D50” ← 2nd 3 moves
   DRAW“L20; U30; R50” ← 3rd 3 moves
   DRAW“D20; L30; U50” ← 4th 3 moves

At the end of the fourth series of draws, you should be at the beginning point.
The draw statements would produce

The final display:

*Figure 4-1. An order-3 spirolateral.*
This program lets you input the three distances for an order-3 spiro-
lateral. Experiment for awhile with the program.

100    "INPUT 3 DISTANCES"
110    CLS
120    INPUT "FIRST DISTANCE"; A1$
130    INPUT "SECOND DISTANCE"; A2$
140    INPUT "THIRD DISTANCE"; A3$
150    "SET GRAPHICS SCREEN"
160    PMODE 0, 1
170    PCLS
180    SCREEN 1, 1
200    "ASSIGN STRING VALUES"
210    A$ = "R"+A1$+"D"+A2$+"L"+A3$ ← for 1st 3 moves
220    B$ = "U"+A1$+"R"+A2$+"D"+A3$ ← for 2nd 3 moves
230    C$ = "L"+A1$+"U"+A2$+"R"+A3$ ← for 3rd 3 moves
240    D$ = "D"+A1$+"L"+A2$+"U"+A3$ ← for 4th 3 moves
250    "DRAW SPIROLATERAL"
260    DRAW"BM60, 30"+A$
270    FOR W = 1 TO 500: NEXT W
280    DRAW B$
290    FOR W = 1 TO 500: NEXT W
300    DRAW C$
310    FOR W = 1 TO 500: NEXT W
320    DRAW D$
350    "DO AGAIN?"
360    Z$ = INKEY$: IF Z$ = "" THEN 360
370    IF LEFT$(Z$,1)< > "S" THEN 110 ← type S to stop; any
380    END other key will start

Try various combinations for the three distances. Spirolaterals can be of
three types:

- each rectangle touching
- hole in the center
- overlapping
MORE DIRECTIONAL MOVES

In addition to drawing up, left, down, and right, the same statement can also DRAW at angles of $45^\circ$, $135^\circ$, $225^\circ$, and $315^\circ$ as measured from north.

Figure 4-2. Angular moves.

Drawing at angles:

100 ' SET THE SCREEN
110 PMODE 0, 1
120 PCLS
130 SCREEN 1, 0
140 ' DRAW AT ANGLES
150 DRAW"BM128, 96; E30; F30; G30; H30"  // First 45, then
160 ' LOOK AT THAT
170 GOTO 170

Here is a sketch of the result:
Exercise 4-3

Use a DRAW statement in the following program to draw a square with lower left corner at 40, 140. Make the sides 40 units long.

a. Use a separate DRAW statement to draw a diagonal from the lower left corner to the upper right corner of the square.

100 ' SET THE SCREEN
110 PMODE 0, 1
120 PCLS
130 SCREEN 1,0
140 ' DRAW SQUARE, THEN DIAGONAL
150
160
170 ' NOW LOOP AROUND
180 GOTO 180

b. Delete line 160 and change line 150 so that the diagonal is included in the string that draws the square.

(Answers are at the end of the chapter.)

The lengths of the lines drawn by the angle commands may surprise you. You must interpret the E, F, G, and H commands to mean:

The specified value for the E, F, G, and H commands will be long enough to draw a diagonal of the square whose sides are the specified values. In other words, give the length of the projection of the diagonal on the horizontal.

The length of the diagonal lines are pictured in Figure 4-3.
You drew a diamond-shaped figure earlier with this program:

```
100  ' SET THE SCREEN
110  PMODE 0, 1
120  PCLS
130  SCREEN 1, 0
140  ' DRAW AT ANGLES
150  DRAW"BM128, 96; E30; F30; G30; H30"
160  ' LOOK AT THAT
170  GOTO 170
```

You could change line 150 of this program so that a line will also be drawn from point A to B, as in this sketch:
The following sketch shows why 60 units are needed for the move in the right direction.

Consider the lines drawn by E30 and F30 as diagonals of squares that are 30 units on a side. The sum must be used as the value for R.

Exercise 4.4

To emphasize the lengths of lines drawn at 45°, 135°, 225°, and 315°, write a program that draws a series of lines, all from the point 128, 96. Make the lines go up 50, right 50, down 50, left 50, then 50 at 45°, 50 at 135°, 50 at 225°, and 50 at 315°. The display should look something like this when the program is run:

Note: The diagonals are longer than the vertical and horizontal lines.

100 ' SET SCREEN
110 PMODE 0, 1
120 PCLS
130 SCREEN 1,0
140 ' DRAW LINES
150 DRAW"BM128, 96; 30; G30; H30; R60"
160 DRAW"BM128, 96; 30; G30; H30; R60"
170
180
190
200
210
220

230 ' STUDY WHILE IT LOOPS
240 GOTO 240

(Answers are at the end of the chapter.)

MANY LINES FROM THE SAME POINT

If you haven't already studied the Radio Shack Extended Color BASIC manual and found a shortcut to the last problem, here is an option for the program in exercise 4-4.

The DRAW statement has an N option that will automatically return to the draw position used to start the last DRAW command. Lines 160-220 of the program in exercise 4-4 could be replaced by one DRAW statement using the N option.

The program would be shortened to

100 ' SET SCREEN
110 PMODE 0, 1
120 PCLS
130 SCREEN 1, 0
140 ' DRAW 8 LINES FROM CENTER
150 DRAW BM128, 96; USO; NRSO; NDSO; NL50; NESO; NFSO; NGSO; NHSO" Each N tells the computer to return to the original point to draw the next line.
230 ' STUDY WHILE IT LOOPS
240 GOTO 240

If you wanted to erase some of the lines drawn by the program, you could use the N option also. You might add these two lines to erase the vertical and horizontal lines previously drawn.

170 COLOR 0, 1 reverse colors
180 DRAW"BM128, 96; U50; NR50; ND50; NL50; NES50; NF50; NG50; NH50"

only these left
COLOR OPTION

Another mode that you can use with DRAW lets you choose the color of the figure drawn (similar to the COLOR statement used before). This is done by using the letter C as the first command in the string.

\[ \text{DRAW} \left \{ \text{CX}; \ldots \ldots \ldots \ldots \right \} \]

where \( X \) is a color value (0-8)

Suppose you entered and executed the following lines:

```
100 ' SET SCREEN
110 PMODE 1, 1
120 PCLS
130 SCREEN 1, 0
150 ' DRAW LINES BLUE
160 DRAW"C3; BM128, 96; U50; NR50; NU50; NL50"
170 ' LOOK AT THE BLUE LINES
180 GOTO 180
```

The display would show

- green background
- blue lines

Exercise 4-5

Remember what happens when you use the background color to draw? To see, add these program lines to lines 100-160 to draw and erase lines alternately.

a. “Erase” the vertical and horizontal lines drawn by line 160.
   170 ___________________________

b. DRAW the diagonals (as in exercise 4-4) blue.
   180 ___________________________

c. “Erase” the diagonal lines drawn in part b.
   190 ___________________________

d. Go back and redraw the horizontal and vertical lines (thus repeating the process over and over).
   200 ___________________________
Since the draw statement uses a string to describe the specified conditions, string variables can be used to define the string to be used for drawing. We did this in the spirolateral program earlier.

Example:
Define the variable\textbf{→130} A$=\text{"BM128, 96; U50; R50; D50; L50"}$
Draw a square\textbf{→150} DRAW A$

Whenever a given DRAW statement is used in a program several times, it is convenient to define a variable with a string and include the variable when the DRAW statement is used. In the following program, A$ and C$ are assigned strings to use at a later time in the program. Study the program carefully. Then answer the questions in exercise 4-6.

```plaintext
100  ' SET SCREEN
110  PMODE 1, 1
120  PCLS
130  SCREEN 1, 1
150  ' DEFINE THE STRINGS
160  A$ = "C6; BM10, 10; R20; D20; L20; U20"
170  C$ = "C6; BM120, 90; R16; D12; L16; U12"
180  ' DRAW BOX NEAR CENTER OF SCREEN
190  DRAW C$
200  ' PAINT BOX AND PERFORM MYSTERY MOVES
210  FOR A = 5 TO 8
220  PAINT (128, 96), A, 6
230  FOR W = 1 TO 300: NEXT W
240  IF PPOINT (128, 96) = 5 THEN DRAW A$
250  IF PPOINT (15, 15) = 6 THEN PAINT (15, 15), 5,5:
      DRAW A$
260  IF PPOINT (128, 96) = 6 THEN PAINT (15, 15), 6, 6
270  IF PPOINT (128, 96) = 7 THEN PAINT (15, 15), 7, 6
280  IF PPOINT (128, 96) = 8 THEN PAINT (15, 15), 8, 6
290  IF PPOINT (128, 96) = 5 THEN PAINT (15, 15), 5, 6
300  FOR W = 1 TO 300: NEXT W
310  PAINT (121, 91), 5, 5
320  DRAW C$
330  NEXT A
350  ' DO THE MYSTERY PART AGAIN
360  GOTO 210
```
Line 190 draws a box 16 by 12 units in cyan near the center of the screen. Line 220 colors the inside of the box drawn by line 190 in one of four colors depending on the value of A. Line 240 causes A$ to be drawn only if the box in the center of the screen is buff. Line 250 causes A$ to be redrawn if the box in the upper left is cyan.

**Exercise 4-6**

Run the program and describe the results.

a. When the center rectangle is buff, the upper rectangle turns

b. When the center rectangle is cyan, the upper rectangle turns

c. When the center rectangle is magenta, the upper rectangle turns

d. When the center rectangle is orange, the upper rectangle turns

(Answers are at the end of the chapter.)

**SUBSTRINGS**

You probably know how a subroutine works in a computer program. You are able to perform a similar feat within the string of a DRAW statement. The X motion command lets you execute a substring within a string.

**Example:**

150 A$ = "E10;F10;G10;H10"

250 DRAW"BM60,90;R30;XA$;D30;L30;U30"
This program draws the top and right side of a square. Then it uses a substring to put a diamond at the upper right corner of the square. Then it goes back and finishes the square.

```
100 'SET SCREEN
110 PMODE 4,1
120 PCLS
130 SCREEN 1,0

200 'DEFINE SUBSTRING
210 A$ = "BM110,90;E10;F10;G10;H10"

300 'DRAW STRING
310 DRAW"BM60,90;R50;D50;XA$;BM110,140;L50;U50"

400 'LOOK AT IT
410 GOTO 410
```

When you use the DRAW statement, you do not always have to separate the motion commands by semicolons. We have done so to indicate clearly each separate motion command. Even if you omit the semicolons as separators for most commands, you must follow the substring command X with a semicolon.

**Examples:**

A$ = "BM110,90E10F10G10H10" ➔ no semicolons

DRAW"BM60,90R50XA$;BM110,40L50U50" ➔ this semicolon is necessary

The semicolon following an X motion must be given even if the X command comes at the end of a DRAW statement.
Let's change the last program that we discussed to draw the following display using as few semicolons as possible.

1. Start and end here, drawing the complete square.
2. Then go to lower left and draw a diamond with a substring.

\[ (110,90) \]

100 'SET SCREEN
110 PMODE 4,1
120 PCLS
130 SCREEN 1,0

200 'DEFINE SUBSTRING
210 A$ = "BM60,140H10G10F10E10"

300 'DRAW STRING
310 DRAW"BM110,90L50D50R50U50X"A$;" 

400 'LOOK AT IT
410 GOTO 410

---

**CATENATION**

From previous use of BASIC, you probably remember that you can catenate strings.

If A$ = "CAT", and B$ = "EN", and C$ = "ATE", then PRINT A$+B$+C$ would print the word:

**CATENATE**

Since the DRAW command contains a string definition, that string can be a catenation of other strings. Study this program.

100 'SET SCREEN
110 PMODE 3,1
120 PCLS
130 SCREEN 1,1
200 'DEFINE A STRING
210 A$ = "C7"  ← a color command

300 'DRAW WITH CATENATION
310 DRAW A$+"BM120,90420D30L20U30"
    ← catenate color with motion

400 'LOOK HERE
410 GOTO 410

Line 310 contains a catenation of A$ with the string in quotes. A$ provides the color to be used (magenta) in the DRAW statement. A single statement that would produce the same effect as lines 210 and 310 is

DRAW"C7BM120,90R20D30L20U30"
    ← same as A$

One advantage of catenating strings in a DRAW statement can be seen in the following program. It allows the color in the DRAW statement to be varied. (Numeric variables cannot be used within a quoted string of a DRAW statement.)

100 'SET THE SCREEN
110 PMODE 3,1
120 PCLS
130 SCREEN 1,0

200 'DEFINE STRING VARIABLES
210 F$ = "BM120,90U40R40D40L40"
220 G$ = "C1BM120,90U40R40D40L40"

300 'ASSIGN COLOR STRINGS
310 FOR Z = 2 TO 4
320 A$(Z) = "C"+STR$(Z)
    ← assigns: A$(2)="C2"
            A$(3)="C3"
            A$(4)="C4"
330 NEXT Z

400 'DRAW THEN ERASE
410 FOR X = 2 TO 4
420 DRAW A$(X)+F$  ← FOR X = 2 TO 4
430 FOR W = 1 TO 30: NEXT W
440 DRAW G$
450 FOR W = 1 TO 30: NEXT W
460 NEXT X

500 'KEEP REPEATING
510 GOTO 410
Exercise 4-7

a. Describe what the display shows when line 420 is executed.

b. Describe what the execution of line 440 does.

(Answers are at the end of the chapter.)

RELATIVE MOTION

You have been using the M command to move to a specified point on the screen in order to begin drawing a geometric figure. You can also use the M command to move a specified distance relative to the last position used. Suppose that you have drawn a square with the DRAW statement:

150 DRAW"BM120,90R30D30L30U30"

You could now draw a square to the left of the original square using the relative motion command

160 DRAW"BM-40,0;R30D30L30U30"

Suppose that you wanted to put the second square to the right of the original square instead of to its left. The relative motion command in line 160 would be:

160 DRAW"BM+40,0;R30D30L30U30"

Semicolon is optional.
The relative motion for the X coordinate must always specify the sign (+ or -). This tells the computer that a relative motion is desired rather than an absolute position.

If the X-offset of a relative motion command is preceded by a plus (+) sign, the last used X position is incremented by the specified amount (the drawing "pen" moves to the right). If the X-offset is preceded by a minus (-) sign, the last used X position is decremented by the specified amount (the drawing "pen" moves to the left).

A positive Y-offset (the "pen" moves down) does not need to be preceded by a plus (+) sign, but a negative Y-offset (the "pen" moves up) does need to be preceded by a minus (-) sign.

The relative movement may be used to draw successive squares in different positions relative to the last position drawn.

Exercise 4-8

Complete the following program to draw the specified original square, then another square 40 units to the left and 40 units above the original. Then add a square 40 units to the right and 40 units below the original as in this sketch. Use relative motion in the DRAW statements.

```
100 'SET SCREEN
110 PMODE 3,1
120 PCLS
130 SCREEN 1,0
200 'DRAW SQUARES
210 DRAW"BM110,80R30D30L30U30"
220 DRAW
230 DRAW
300 'LOOP AROUND
310 GOTO 310
```

(Answers are at the end of the chapter.)
It might be interesting to make the square appear to move down and to the right by drawing a square, erasing it, moving it to the next position, etc. Enter and run the following program to see this possibility.

```
100 'SET SCREEN
110 PMODE 4,1
120 PCLS
130 SCREEN 1,0

200 'DRAW ORIGINAL SQUARE
210 A$ = "R30D30L30U30"
220 DRAW "BM10,10" + A$

300 'WAIT THEN ERASE IT
310 FOR W = 1 TO 100: NEXT W
320 COLOR 0,1: DRAW "BM10,10" + A$

400 'NOW MOVE, ERASE, MOVE, ETC.
410 FOR Z = 1 TO 12
420 COLOR 1,0
430 B$ = "BM+10,10"  "move to right and down"
440 DRAW B$ + A$
450 FOR W = 1 TO 100: NEXT W
460 COLOR 0,1
470 DRAW "BM+0,0" + A$  "erase"
480 FOR W = 1 TO 100: NEXT W
490 NEXT Z

500 'PUT LAST ONE ON TO STAY
510 COLOR 1,0: DRAW B$ + A$
520 GOTO 520
```

There are still more DRAW features, but it is time for you to take a break and review the many things that you have already learned about the DRAW statement.

**Summary**

After completing this chapter, you are an experienced artist. You have learned to use most of the motion commands in a DRAW statement. You are now able to

- append a string to a DRAW statement to define the draw movements that you desire
- move to any point on the screen with a Blank Move, such as
• DRAW from the starting point in one of many directions:
  - U for up
  - D for down
  - L for left
  - R for right
  - E for 45°
  - F for 135°
  - G for 225°
  - H for 315°
• use the N motion command to return to the point of origin of the previous move, such as
  \[
  \text{DRAW"BM128,96;U40;N;R40;N;D40;N;L40"}
  \]
  \[\text{start each of these at 128,56 (up 40 from 128,96)}\]
• choose the color for drawing lines
  \[
  \text{DRAW"C6........."}
  \]
  \[\text{draw the lines defined in cyan}\]
• define string variables for use in a DRAW statement
• use a substring to add to the drawing performed by the main string
• define strings without separating the motion commands by semicolons (one exception, the X substring command)
• concatenate strings within a DRAW statement, such as
  \[
  \text{DRAW A$+B$+C$}
  \]

Try the chapter test to see if you can remember all these capabilities. You will find some additional DRAW features in the next chapter.

Chapter Test

1. The information defining the motion commands of a DRAW statement is contained in a (check one)
   - a. crayon box
   - b. IF-THEN statement
   - c. DEF statement
   - d. string
2. Why should you use a B before the first motion command of a DRAW statement?

3. To draw a line from the center of the screen 10 units upward and then 10 units to the left, the DRAW statement should include

4. Name the four motion commands used in a DRAW statement for drawing a square 10 units on a side.
   a. up
   b. down
   c. left
   d. right

5. Write a DRAW statement that would draw a rectangle with these points as corners (use D, U, L, and R). Start at the first point given and proceed in the order given: 40,20 to 110,20 to 110,80 to 40,80.

6. Write a DRAW statement to draw a rectangle 15 units outside the rectangle of exercise 5.

7. Write a DRAW statement that would draw this diamond shape (each side the same length).

8. Write a DRAW statement to draw a diagonal from the lower left corner to the upper right corner of the figure drawn by line 150.

9. Draw a sketch of the lines that would be drawn by the statement:

   DRAW"BM128,96;U50;NR50;NU50;NL50"
10. To specify a green line, the first command in a DRAW statement would be

```
DRAW
```

11. Give the string variable to be used in a DRAW statement that would draw a cyan square when using PMODE 1,1 and SCREEN 1,1.

```
A$ = "C5;BM10,10;R20;D20;L20;U20"
B$ = "C6;BM20,90;E20;F20;G20;D20"
C$ = "C6;BM30,60U20R20D20L20"
```

The correct variable is

12. If A$ has been defined for use in a substring of a DRAW statement, what is the command used within the DRAW statement that would execute the substring?

13. Write a DRAW statement to catenate these string variables.

```
A$ = "C7"
B$ = "BM120,90R20D30L20U30"
DRAW
```

14. Explain the difference produced on the display by these two statements:

```
150 DRAW"BM40,50R30D30L30U30"
160 DRAW"BM+40,-50R30D30L30U30"
```

15. Complete these statements concerning relative motion commands used in a DRAW statement with a choice of one word: up, down, left, or right.

a. A positive Y-offset moves the "pen" ________________

b. A negative X-offset moves the "pen" ________________
c. A positive X-offset moves the "pen"

d. A negative Y-offset moves the "pen"

16. Write a program using DRAW and CIRCLE statements to create this design on the display:

```
150 DRAW"BM128,96;U25;L25;U71;L103"
```

Answers to Exercises Within the Chapter

Exercise 4-1

150 DRAW"BM10,185;U90;R150;D90;L150"

or

150 DRAW"BM10,185;R150;U90;L150;D90"

GOTO 170

Exercise 4-2

110 PMODE 0,1
120 PCLS
130 SCREEN 1,0
150 DRAW"BM40,140;R40;U40;L40;D40"

B for Blank is not needed; 150 ends at lower left

160 DRAW"E40"

Exercise 4-3

a. 150 DRAW"BM40,140;R40;U40;L40;D40"

b. 150 DRAW"BM40,140;U40;L40;D40;E40"
Exercise 4-4

150  DRAW“BM128,96;U50”
160  DRAW“BM128,96;R50”
170  DRAW“BM128,96;D50”
180  DRAW“BM128,96;L50”
190  DRAW“BM128,96;E50”
200  DRAW“BM128,96;F50”
210  DRAW“BM128,96;G50”
220  DRAW“BM128,96;H50”

Exercise 4-5

a. 170  DRAW“C3;BM128,96;U50;NR50;ND50;NL50”
b. 180  DRAW“Cl ;BM128,96;E50;NF50;NG50;NH50”
c. 190  DRAW“C3;BM128,96;E50;NF50;NG50;NH50”
d. 200  GOTO 160

To have a smoother effect, you might want to add a small time delay after certain lines in the program, say

165  FOR W = 1 TO 50: NEXT W
185  FOR W = 1 TO 50: NEXT W

Exercise 4-6

a. turns buff
b. turns buff
c. turns magenta
d. turns orange

Exercise 4-7

a. A rectangle is drawn near the center of the screen. As the loop is executed, it blinks on and off in colors that change from yellow to blue to red.
b. Line 440 erases the rectangle (turns it off by using the background color, green).

Exercise 4-8

220  DRAW“BM-40,-40R30D30L30U30”
230  DRAW“BM+80,80R30D30L30U30”

You are moving from the position where the second square ended. You must move 40 units right and down to get to the origin and then 40 units more right and down to get to the position needed.
Answers to Odd-Numbered Exercises in Chapter Test

1. d. string

3. "BM128,96;U10;L10
   semicolons are optional

5. "BM40,20;R70;D60;L70;U60"

7. "BM60,90;E30;F30;G30;H30"

9. 

11. C$

13. DRAW A$+B$

15. a. down
    b. left
    c. right
    d. up
You covered most of the DRAW functions in the last chapter, but we saved one that will provide much fun and entertainment for this chapter. You'll also learn how to "turn" graphics pages. This will allow the introduction of animation. All in all, this is one of the most important chapters in the book. You will learn

- to change the size of your drawings
- to reserve memory for necessary graphic pages
- to copy graphics from page to page
- to use all the above to animate drawings on the video display

You will also learn

- how the PMODE statement is used to select a graphic page
- how many pages are used in each graphics mode

**SCALE UP OR DOWN**

The color computer has one more motion command that can be used in a DRAW statement. It allows you to scale up (enlarge) or to scale down (reduce) the size of any display created by a DRAW statement. The command is

\[ SX \]

scale \[ \text{an integer from 1-62} \]

Study Table 5-1 to see what the values used for X in the Scale command do.
TABLE 5-1. SCALE FACTORS FOR DRAW

<table>
<thead>
<tr>
<th>X Value</th>
<th>Scale of Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/4</td>
</tr>
<tr>
<td>2</td>
<td>2/4</td>
</tr>
<tr>
<td>3</td>
<td>3/4</td>
</tr>
<tr>
<td>4</td>
<td>4/4 (full scale)</td>
</tr>
<tr>
<td>5</td>
<td>5/4</td>
</tr>
<tr>
<td>60</td>
<td>60/4</td>
</tr>
<tr>
<td>61</td>
<td>61/4</td>
</tr>
<tr>
<td>62</td>
<td>62/4</td>
</tr>
</tbody>
</table>

The SX command may be inserted in the DRAW string where needed. Within a given program, all absolute and relative motion commands will be scaled according to the last SX command that has been executed. If no SX commands have been executed, the full-scale value (4/4) is used.

If the following statement is executed, the starting coordinates for the drawing are 120 for X and 90 for Y.

```
DRAW"52; BM120, 90; U20; L20; D20; R20"
```

The lines will be drawn 10 units up, 10 units left, 10 units down, and 10 units right even though 20 units are stated in the DRAW statement. The S2 command at the beginning of the statement tells the computer to scale everything (except the original X, Y coordinates) by a factor of two-fourths (or one-half).

Once again, you can use the STR$ function to sneak a changing value into the string of a DRAW statement.

**Example:**

When the following loop is executed

```
FOR X = 1 TO 4
    S$ = "S"+STR$ (X) + ";"
    DRAW S$+"BM128, 80; R40; D40; L40; U40"
NEXT X
```

the string S$ will be evaluated as
 SCALE UP OR DOWN  

\[ S$ = "S" + "1" + ";" \text{ when } X = 1 \]
\[ = "S" + "2" + ";" \text{ when } X = 2 \]
\[ = "S" + "3" + ";" \text{ when } X = 3 \]
\[ = "S" + "4" + ";" \text{ when } X = 4 \]

Four squares will be drawn. The last square drawn will be the largest:
When \( X = 1 \), the length of the square will be 10 units.
When \( X = 2 \), the length of the square will be 20 units.
When \( X = 3 \), the length of the square will be 30 units.
When \( X = 4 \), the length of the square will be 40 units.

Exercise 5-1

Draw a sketch of the display that would be drawn by the execution of this program.

100  ' SET SCREEN
110  PMODE 4, 1
120  PCLS
130  SCREEN 1, 1
200  ' MAKE 4 DRAWINGS TO SCALE
210  FOR X = 1 TO 4
220  S$ = "S"+STR$ (X) + ";"
230  DRAW S$+"BM120, 80; R40D40L40U40"
240  NEXT X
300  ' LOOK AT ALL 4 SQUARES
310  GOTO 310

(Answer is at the end of the chapter.)

The following program displays all the scale factors from 1 through 62. It draws squares with the Blank Motion command and R, D, L, and U motions 12 units long.
100 ' SET SCREEN
110 PMODE 4, 1
120 PCLS
130 SCREEN 1, 0

200 ' 62 SQUARES ON THE SCREEN
210 FOR X = 1 TO 62
220 S$ = "S" + STR$ (X) + ";"
230 DRAW S$ + "BM2, 2; R12D12L12U12"
240 NEXT X

300 ' LOOP TO LOOK
310 GOTO 310

When the program is executed, the upper left corner for each square will be at the point 2, 2, but the size of the squares grow and grow until they fill up the screen. They even overflow near the end of the program and "wrap around" the screen's dimensions, distorting the final squares drawn.

If a DRAW statement without a scale factor is encountered following a previous scale change, the last scale factor given stays in effect until another SX command is executed.

Suppose this program is executed:

100 ' SET SCREEN
110 PMODE 4, 1
120 PCLS
130 SCREEN 1, 0

200 ' STEP DOWN THE SCALE
210 FOR X = 4 TO 1 STEP -1
220 S$ = "S" + STR$ (X) + ";"
230 DRAW S$ + "BM80, 80; R40U40L40D40"
240 FOR W = 1 TO 300: NEXT W
250 NEXT X

300 ' DRAW SQUARE NO SCALE FACTOR
310 FOR W = 1 TO 300: NEXT W
320 DRAW "C0; BM80, 80; R40; U40; L40; D40"

400 ' LOOK MA, ONE SQUARE GONE
410 GOTO 410

Four squares were drawn by lines 210-250. The lengths of their sides were 40, 30, 20, and 10, respectively. The execution of line 320 erased one of the squares. It erased the last square drawn (the smallest one). The last scale factor used in lines 210-250 was one-fourth. That stays in effect until
another SX command is executed. Therefore, line 320 is executed with a scale factor of one-fourth also.

Exercise 5-2

Suppose that you had defined S$ and B$ and had executed the DRAW statement in this loop:

130 FOR X = 1 TO 4
140   S$ = "S" + STR$(X) + ";"
150   B$ = "BM120, 80;"
160   DRAW S$+B$+"R40; D40; L40; U40"
170 NEXT X
180 GOTO 180

a. Would the loop produce the same display as that of exercise 5-1?

b. If line 160 were changed to

160   DRAW B$+"R40; D40; "+S$+ "L40; U40"

would the display be different?

c. Draw a sketch of the display resulting from the change in line 160 given in part b.

(Answers are at the end of the chapter.)

In previous programs, you probably noticed that the squares moved down and to the right as the scale was changed. This movement can be avoided by moving the "center" of the square rather than keeping the upper left corner in the same location.

A typical DRAW statement for the new method, as used in a loop, would be
FOR X = 8 TO 60 STEP 4
A$ = "BM120, 90; BH4"
S$ = "S" + STR$(X)
DRAW A$+S$+"R8D8L8U8"
NEXT X

Previous Method

New Method

Blank Move to left corner, then change scale, then begin square

Move to center of square, then change scale, then blank move 315 degrees to begin square

Square seems to move down and to the right

Square seems to move outward

Figure 5-1. Centering squares.

The following program uses the preceding loop and has some added lines to give a short time delay and to erase the square between changes in scale factors.

100 ' SET SCREEN
110 PMODE 1, 1
120 PCLS
130 SCREEN 1, 0
200 ' DRAW
210 FOR X = 8 TO 60 STEP 4
220 A$ = "BM120, 90; BH4"
230 S$ = "S" + STR$(X)
240 DRAW A$+S$+"R8D8L8U8"
For a variation try 520 DRAW"S4": GOTO 210. This sets the scale factor back to full size and starts all over again.

## TURNING PAGES IN MODE 0

It's time now to "turn" the page and learn how the TRS-80 Color Computer graphics memory is allocated. Remember, the PMODE statement has two parameters:

```plaintext
PMODE n, m
```

- graphics mode (0-4)
- starting page (1-8)

High-resolution graphics modes use more memory pages than low-resolution modes as Table 5-2 shows.

### TABLE 5-2. GRAPHIC PAGES AND COLORS

<table>
<thead>
<tr>
<th>Mode</th>
<th>Memory Pages</th>
<th>Colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMODE 4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>PMODE 3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>PMODE 2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>PMODE 1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>PMODE 0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

More memory is necessary when more colors are available or when higher resolution is available. Each graphics page uses 1.5K of memory (approximately 1,500 locations).
PMODES 3 and 4 use the most memory. More memory is necessary for higher resolution, and more memory is necessary for more colors.

Normally, four pages of memory are reserved for graphics when the color computer is turned on. More than this number of pages can be reserved for graphics by executing the following statement:

```
PCLEAR n
```

\( n \) may be a number from 1 through 8

Here is an exercise to demonstrate how the number of graphics pages reserved changes the memory available for your programs.

**Exercise 5-3**

Clear any program that may be in the computer's memory by typing NEW.

a. Now, type in PRINT MEM (and press the ENTER key)

What value is printed? 

b. Successively type in the following PCLEAR and PRINT MEM statements. Fill in the memory values that you see.

```
PCLEAR 1
PRINT MEM
PCLEAR 2
PRINT MEM
PCLEAR 3
PRINT MEM
PCLEAR 4
PRINT MEM
```
PCLEAR 5
PRINT MEM
PCLEAR 6
PRINT MEM
PCLEAR 7
PRINT MEM
PCLEAR 8
PRINT MEM

You should get the same answers as shown in Table 5-3.

<table>
<thead>
<tr>
<th></th>
<th>Memory Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13095</td>
</tr>
<tr>
<td>2</td>
<td>11559</td>
</tr>
<tr>
<td>3</td>
<td>10023</td>
</tr>
<tr>
<td>4</td>
<td>8487</td>
</tr>
<tr>
<td>5</td>
<td>6951</td>
</tr>
<tr>
<td>6</td>
<td>5415</td>
</tr>
<tr>
<td>7</td>
<td>3879</td>
</tr>
<tr>
<td>8</td>
<td>2343</td>
</tr>
</tbody>
</table>

To learn how to "turn" pages, let's go to the lowest resolution mode and reserve two pages of memory for graphics. We'll then draw a square on page 1 and a circle on page 2. As you look at the demonstration program, notice that we did not use the SCREEN statement while we made the drawings. The SCREEN statement is used to display what you have drawn.

100 ' RESERVE MEMORY AND SET SCREEN
110 PCLEAR 2 ← save 2 graphics pages
120 PMODE 0, 1 ← low res., start at page 1
130 PCLS

200 ' DRAW A SQUARE
210 DRAW"BM120, 90; R20D20L20U20"

300 ' CHANGE PAGES, DRAW CIRCLE
310 PMODE 0, 2
320 PCLS
330 CIRCLE (120, 90), 10
On the screen you will alternately see

A square on page 1

A circle on page 2

Figure 5-3. Two-page display.

Exercise 5-4

Complete this program to display a circle in a different position on each of the eight pages of memory. Don't forget that the graphics screen is not turned on in the DRAW section of the program.

100 ' RESERVE 8 PAGES
110 PCLEAR 8
200 ' DRAW SECTION
210 FOR P = 1 TO 8
220 PMODE 0, P
230 PCLS
240 CIRCLE (20+24*P, 90), 10
250 NEXT P
300 ' YOU DISPLAY IT
310 FOR
320 PMODE
330
340
Not only can you turn graphic pages, but also you can copy the graphics from one page to another. This is done by the statement

```
PCOPY X TO Y
```

where X and Y are integers 1 through 8

The PCOPY statement can be used in producing apparent movement on the video display. Suppose that you want to move a circle from corner to corner around a square like this:

```
counterclockwise
```

1 2 3 4

```
clockwise
```

5 6 7 8

If you work in PMODE 0, you can use one page of graphic memory for each position of the circle. The square appears in the same position on all pages, but the circle changes. You also notice that some of the positions of the circle are the same as others: 1=5, 2=8, 3=7, and 4=6. This presents an ideal opportunity to use the PCOPY statement.

First, draw a square on graphics page 1 and copy it to all the other seven pages.

```
100 ' CLEAR 8 PAGES AND DRAW A SQUARE
110 PCLEAR 8
120 PMODE 0, 1
130 PCLS
140 DRAW"BM120, 88; R16D16L16U16"
200 ' COPY SQUARE ON OTHER PAGES
210 FOR P = 2 TO 8
```
Second, put circles in correct positions and copy to appropriate pages.

300 ' PAGE 1 AND 5
310 PMODE 0, 1
320 CIRCLE (136, 88), 8
330 PCOPY 1 TO 5

400 ' PAGE 2 TO 8
410 PMODE 0, 2
420 CIRCLE (120, 88), 8
430 PCOPY 2 TO 8

500 ' PAGE 3 TO 7
510 PMODE 0, 3
520 CIRCLE (120, 104), 8
530 PCOPY 3 TO 7

600 ' PAGE 4 TO 6
610 PMODE 0, 4
620 CIRCLE (136, 104), 8
630 PCOPY 4 TO 6

Third, turn on the display and turn the pages. Use a time delay to view each page. Do it slowly first, using a fairly long time delay. Then change the delay loop parameters to speed things. We are using 500 as the count for our initial time delay, but we lowered it to 50 after the program was running satisfactorily.

700 ' DISPLAY PAGES IN SUCCESSION
710 FOR P = 1 TO 8
720 PMODE 0, P
730 SCREEN 1, 0
740 FOR W = 1 TO 500: NEXT W
750 NEXT P
760 GOTO 710

Enter and run the complete program. Watch the circle move from corner to corner. First it goes in a counterclockwise direction, then it reverses directions. It keeps going back and forth.
Now, let’s get a little fancier. Have you ever tried to pat the top of your head with one hand while rubbing your stomach in a circular motion with the other? We are going to try to produce a similar action on the screen.

While our initial program is moving the circle back and forth around the square, we’ll move a second circle around a larger rectangle in a counterclockwise direction.

First, you need to draw the larger square. Do this by adding

150 DRAW “BM80, 50; R100D100L100U100”

Second, you need circles in eight different positions. This can be done by adding the following lines to the original program:

650 PMODE 0, 1: CIRCLE (180, 50), 25
655 PMODE 0, 2: CIRCLE (130, 50), 25
660 PMODE 0, 3: CIRCLE (80, 50), 25
665 PMODE 0, 4: CIRCLE (80, 100), 25
670 PMODE 0, 5: CIRCLE (80, 150), 25
675 PMODE 0, 6: CIRCLE (130, 150), 25
680 PMODE 0, 7: CIRCLE (180, 150), 25
685 PMODE 0, 8: CIRCLE (180, 100), 25

Enter these changes and run the program with the modifications. ACT 1 moves the inner circle back and forth around its square. ACT 2 moves the second circle counterclockwise around the outer rectangle.

To reverse the directions, you could change line 710 to

710 FOR P = 8 TO 1 STEP-1
If you had three hands, you could pat your head with one hand, rub your tummy with the second hand, and bounce a ball with the third. It would take some coordination to do all three actions at the same time. You can do something similar by adding a third action to ACT 1 and ACT 2 of our program. Let's put the bouncing ball to the left of the churning circles.

ACT 3 The Bouncing Ball

To get all your acts together, modify lines 650 through 685 of our program as follows:

```plaintext
650 PMODE 0, 1: CIRCLE (180, 50), 25: CIRCLE (40, 50), 10
655 PMODE 0, 2: CIRCLE (130, 50), 25: CIRCLE (40, 75), 10
660 PMODE 0, 3: CIRCLE (80, 50), 25: CIRCLE (40, 100), 10
665 PMODE 0, 4: CIRCLE (80, 100, 25: CIRCLE (40, 125), 10
670 PMODE 0, 5: CIRCLE (80, 150), 25: CIRCLE (40, 150), 10
675 PMODE 0, 6: CIRCLE (130, 150), 25: CIRCLE (40, 125), 10
680 PMODE 0, 7: CIRCLE (180, 150), 25: CIRCLE (40, 100), 10
685 PMODE 0, 8: CIRCLE (180, 100), 25: CIRCLE (40, 75), 10
```

Enter these changes and run the finished program. Can you keep your eyes on all three ACTs? It's just like a three-ring circus. It's hard to see it all, but it's great fun. You may want to add other acts if you can find room on the screen.

**Exercise 5-5**

Complete a program that will make a small circle "roll" around a larger circle, as in Figure 5-4.
Figure 5-4. Roll around circles.

Draw the large circle on graphics page 1. Copy page 1 to pages 2, 3, 4, 5, 6, 7, and 8. Draw each small circle on a separate page in the position shown in Figure 5-5. Use 40 for the radius of the large circle and place its center at 128, 96. Use 10 for the radius of each small circle. Draw the circles in PMODE 0.

100 ' CLEAR 8 PAGES AND DRAW BIG CIRCLE
110
120
130
140
200 ' COPY BIG CIRCLE
210
220
230
300 ' DRAW SMALL CIRCLES ON SEPARATE PAGES
310 PMODE 0, 1: CIRCLE (128, 46), 10
320 PMODE 0, 2: 
330
340
350
360
370
380
400 ' DISPLAY — GOES AROUND AND AROUND
410
420
TURNING PAGES IN OTHER MODES

Each graphics page fills the entire screen in mode 0. Therefore, "turning" each page presents a new video display. When you move to higher resolution modes, more than one page of memory is necessary to fill the screen. Table 5-2 summarizes graphics pages and colors used in each mode.

PMODES 1 and 2 require two pages of memory for a screenful.

PMODES 3 and 4 require four pages to fill the screen.

Let's look first at mode 1. We'll put an orange line at the top of page 1 and a magenta line at the bottom of page 1. The following program will do it and will put nothing on page 2.

100 'SET SCREEN AND DRAW LINES
110 PMODE 1, 1
120 PCLS
130 LINE (0, 0) - (255, 0), PSET
140 COLOR 7, 5
150 LINE (0, 95) - (255, 95), PSET
200 'TURN ON DISPLAY
When line 210 is executed, the computer displays page 1 and page 2 together because two pages are necessary to fill the screen in PMODE 1. You see

![](diagram.png)

To prove the point, add this line to the program and run it again:

160 PCOPY 1 to 2

After the lines are drawn on page 1, they are copied to page 2 by line 160. When line 210 is executed, you see

![](diagram2.png)

It begins to look like all the screen positions from 0, 0 through 255, 95 are displayed by graphics page 1 when in PMODES 1 and 2. Consequently, all the screen positions from 0, 96 through 255, 191 must be displayed on graphics page 2. See if this is true by entering and running this program.

100 ' SET SCREEN – DRAW BOXES
110 PMODE 1, 1
120 PCLS
130 LINE (0, 96) - (127, 145), PSET, BF
140 COLOR 7, 5
150 LINE (128, 145) - (255, 191), PSET, BF
200 ' TURN ON DISPLAY
210 SCREEN 1, 1
220 GOTO 220
When line 210 is executed, both pages 1 and 2 are displayed. Since you drew nothing in the page 1 area, the top half of the screen is blank. The colored rectangles appear on the bottom half of the screen (the area of page 2).

![Diagram of page layout](image)

To copy the rectangles to the top half of the screen, insert line 160 and run the program again.

160 PCOPY 2 TO 1

When line 210 is executed, you see

![Figure 5-5. Copied page in mode 1.](image)

The PCLS statement clears both pages. You can see this by adding

```
220 FOR W = 1 TO 500: NEXT W
230 PCLS
240 SCREEN 1, 1
250 GOTO 250
```

You see that pages 1 and 2 must be considered together when you are working in PMODE 1 or PMODE 2. In addition, graphics pages 3 and 4, 5 and 6, and 7 and 8 work together in these two modes. When you change pages, start with page 1, but turn two pages at a time.

The following program draws a line at the top of page 1 and a circle at
the bottom of page 2. It copies these to pages 3 and 4. It then copies pages 3
and 4 back in the reverse order. Page 4 goes to page 1 and page 3 goes to page
2. When the pages are displayed, the relative positions of the line and circle
are quite different.

100 'SET SCREEN
110 PMODE 1, 1
120 PCLS
200 'DRAW LINE AND CIRCLE, PAGES 1 and 2
210 LINE (0, 0) - (255, 0), PSET
220 CIRCLE (128, 180), 5

300 'COPY TO 3 AND 4
310 PCOPY 1 TO 3: PCOPY 2 TO 4

400 'PCOPY IN REVERSE TO OPPOSITE PAGES
410 PCOPY 4 TO 1: PCOPY 3 TO 2

500 'DISPLAY 1 & 2, THEN 3 & 4
510 FOR P = 1 TO 3 STEP 2
520 PMODE 1,P
530 SCREEN 1,1
540 FOR W = 1 TO 500:NEXT W
550 NEXT P
600 'ALTERNATE PAGES
610 GOTO 510

As you can see, the picture presented on pages 1 and 2 is quite different
due to the order in which pages 3 and 4 were copied back.
Let's go on to the higher resolution modes. In PMODES 3 and 4, it takes four pages of graphics to fill the screen. Therefore, you must work with four pages of memory at a time.

To see how the screen is divided by the four graphics pages, draw a line at the bottom of page 1 and copy the line to pages 2, 3, and 4 with this program.

100 'SET SCREEN
110 PMODE 3, 1
120 PCLS
200 'DRAW LINE AT BOTTOM, PAGE 1
210 LINE (0, 47) - (255, 47), PSET
The following program moves a circle from the top of the screen to the bottom by using all eight pages of graphics memory in PMODE 3. The PCOPY statement is used to place the circles in the correct positions on individual pages.

```
100 'SET GRAPHICS SCREEN
110 PCLEAR 8
120 PMODE 3, 1
130 PCLS
200 'PUT BLANK CIRCLE ON PAGES
210 COLOR 7, 5
220 CIRCLE (10, 106), 5
230 PCOPY 1 TO 6: PCOPY 2 TO 5
240 PCOPY 3 TO 8: PCOPY 4 TO 7
300 'DISPLAY 4 PAGES AT A TIME
310 FOR P = 1 TO 8 STEP 4
320 PMODE 3,P
330 SCREEN 1, 1
340 FOR W = 1 TO 500: NEXT W
```

Figure 5-6. Copied page in mode 3.
Here is a graphic picture of what happens:

from lines 210-240

from lines 410-480

You could change line 510 to

510 GOTO 410

This would repeat the cycle.

Time for another break to review what has been covered and to recall all that you have learned.

Summary

In this chapter you reduced and enlarged drawings and manipulated graphics pages to produce moving images on the video screen. You learned

- that once a scale command has been executed in a program, that scale stays in effect until a new scale command is executed

- that mode 0 needs only one graphics page to fill the video screen

- that modes 1 and 2 need two graphics pages to fill the video screen
• that modes 3 and 4 need four graphics pages to fill the video screen

You also learned
• to scale your drawings with

\[ \text{DRAW"SX \ldots \ldots \ldots . \quad "} \]
\[ x \times \frac{1}{4} \text{ gives the scale factor} \]
• to draw on different pages to set up rapidly changing pictures
• to turn from one page to another in the low-resolution graphics mode
• to reserve necessary graphics pages with

\[ \text{PCLEAR } n \]
\[ \text{where } n \text{ is the number of pages reserved} \]
• to copy graphics from one page to another with

\[ \text{PCOPY } X \text{ TO } Y \]
\[ X \text{ and } Y \text{ are integers from 1 through 8} \]
• to turn two or four pages at a time in high-resolution modes

Proceed to the chapter test and demonstrate your knowledge.

Chapter Test

1. How many units would the side of a square be, on the display, if line 160 were executed?

\[ 160 \text{ DRAW"S3;BM10, 10;R80D80L80U80" } \]
\[ \text{units long} \]

2. Change line 160 in exercise 1 to produce a square whose sides are 160 units long.

\[ 160 \text{ DRAW"S3;BM10, 10;R80D80L80U80" } \]
\[ \text{units long} \]

3. Define S$ in line 210 so that the three squares drawn will have sides of 20, 60, and 100 units, respectively.

\[ 210 \text{ FOR } X = \]
4. Complete these statements:

PMODE 0 requires ___________ graphics memory pages to fill the display screen.

PMODE 2 requires ___________ graphics memory pages to fill the display screen.

PMODE 4 requires ___________ graphics memory pages to fill the display screen.

5. Study the following program. DO NOT run it on the computer. Find a serious error and add a line to correct the error.

```
110 PMODE 3, 1
120 PCLS
130 COLOR 7, 5
140 LINE (0,0) - (255, 191), PSET, BF
150 PCOPY 1 TO 5: PCOPY 2 TO 6
160 PCOPY 3 TO 7: PCOPY 4 TO 8
170 COLOR 8, 5
180 CIRCLE (128, 96), 20
190 FOR P = 1 TO 8 STEP 4
200    PMODE 3, P
210    SCREEN 1, 1
220    FOR W = 1 TO 500: NEXT W
230    NEXT P
240 GOTO 190
```

Add:

6. If a 16K Color Computer with Extended Color BASIC is to be used without high-resolution graphics, you can make the most memory available for your program by executing:

```
```

7. Complete the necessary variable assignments and write a loop to draw the following figures, one on each of eight pages of graphics memory.
100 'SET SCREEN AND PICK VARIABLES
110 PMODE 0, 1
120 PCLS
130 A$(1) = "USO": A$(2) = "E35"
140 A$(3) = _____: A$(4) = _____
150 A$(5) = _____: A$(6) = _____
160 A$(7) = _____: A$(8) = _____
200 'DRAW
210 FOR P = 1 TO 8
220 PMODE 0, P
230 DRAW _______________________
240 NEXT P

8. Add a loop to display the eight pages drawn in exercise 7 so that a rotating clock hand spins around and around.

300 'TURN ON DISPLAY
310 FOR P = _____________________
320 ____________________________
330 ____________________________
340 ____________________________
350 GOTO 350

9. Robbie, the robot, is to be drawn so that only his eyes move as graphics pages are turned, as shown.

Given the initial part of the program, complete the necessary lines to copy the nonmoving parts to the necessary pages.

110 PMODE 0, 1
120 PCLS
130 LINE (63, 32) - (191, 159), PSET, B
140 LINE (89, 51) - (115, 77), PSET, B
150 LINE (139, 51) - (165, 77), PSET, B
160 LINE (102, 118) - (152, 128), PSET, BF
200  ' COPY NONMOVING PARTS
210  FOR P = 
220  PCOPY ___ TO ___
230  NEXT P

10. Complete a loop to draw the moving eyes of exercise 9 onto the necessary graphics pages.

300  ' DRAW MOVING EYES
310  FOR P = 1 TO 3
320  PMODE 0, P
330  LINE (___, ___) - (___, ___), PSET, BF
340  NEXT P

11. Finish the program of exercises 9 and 10 to display Robbie with the rolling eyes.

400  ' DISPLAY
410  FOR P = 
420  PMODE 0, ___
430  SCREEN 1, 1
440  FOR W = 1 TO 50: NEXT W
450  
460  GOTO ___

12. Use PMODE 4 and write a program to simulate a marble rolling down a four-part chute like this:
Answers to Exercises Within Chapter

Exercise 5-1

Exercise 5-2

a. yes
b. yes
c.

Exercise 5-3 was answered in Table 5-6.

Exercise 5-4

310 FOR P = 1 TO 8
320 PMODE 0, P
330 SCREEN 1, 1
340 FOR W = 1 TO 100: NEXT W
350 NEXT P
410 GOTO 310

Exercise 5-5

110 PCLEAR 8
120 PMODE 0, 1
130 PCLS
140 CIRCLE (128, 96), 40
210 FOR P = 2 TO 8
220 PCOPY 1 TO P
230 NEXT P
320 PMODE 0,2: CIRCLE (88, 66), 10
330 PMODE 0,3: CIRCLE (78, 96), 10
340 PMODE 0,4: CIRCLE (88, 126), 10
350 PMODE 0,5: CIRCLE (128, 146), 10

Changing scales in midstring can be disastrous.
360 PMODE 0,6: CIRCLE (168, 126), 10
370 PMODE 0,7: CIRCLE (178, 96), 10
380 PMODE 0,8: CIRCLE (168, 66), 10
410 FOR P = 1 TO 8
420 PMODE 0,P
430 SCREEN 1,0
440 NEXT P

Answers to Odd-Numbered Exercises in Chapter Test

1. 60 units long
3. 210 FOR X = 1 TO 5 STEP 2
   220 S$ = "S" + STR$(2*X) + ";"
   or
   210 FOR X = 2 TO 10 STEP 4
   220 S$ = "S" + STR$(X) + ";"
5. 100 PCLEAR 8
   all eight pages will be used
7. 140 A$(3) = "R50": A$(4) = "F35"
   150 A$(5) = "D50": A$(6) = "G35"
   160 A$(7) = "L50": A$(8) = "H35"
   230 DRAW "BM 128, 96" + A$(P)
9. 210 FOR P = 2 TO 3
   220 PCOPY 1 TO P
11. 410 FOR P = 1 TO 3
    420 PMODE 0,P
    430 NEXT P
    460 GOTO 410
PUT, GET, and JOYsticks

By this time, you are getting pretty fast with the DRAW com and. However, objects can be moved even faster with a combination of two statements to be covered in this chapter. You'll also finally learn the statements necessary to use the Radio Shack joysticks, which can be added to your color computer. By the time you finish this chapter, you will

- be able to move all the graphics in a given area of the video screen to another area by means of GET and PUT statements
- know which two joystick numbers correspond to the left joystick and which correspond to the right joystick
- know which two joystick numbers correspond to the horizontal movement and which two joystick numbers correspond to vertical movement
- be able to paint a picture with one joystick
- know how to use the push buttons on the joysticks
- be able to use the joysticks to call subroutines or alter program execution in other ways

GET AND PUT

The PUT and GET statements provide the fastest method for moving objects about the video display. The GET statement allows you to access a rectangular area of a graphics display. The data are stored in an array. The PUT statement allows you to place the data from the array back on the video display at a later time at the same, or a different, place.

Suppose that you have a figure drawn in the upper left area of the screen such as this:
You need to GET an area of the screen that includes the figure.

\[(X_1, Y_1) \rightarrow (X_2, Y_2)\]

The necessary statement would be

\[\text{GET} \ (X_1, Y_1)-(X_2, Y_2), A, G\]

coordinates of upper left corner of area
coordinates of lower right corner of area
variable name of array where graphic data are to be
tells the computer to store details (optional sometimes; use only in PMODEs 2 and 4)

Since you are going to store the data in an array, you must dimension the array before using PUT or GET with

\[\text{DIM} \ A(X_2-X_1, Y_2-Y_1)\]

Later in your program, you may PUT the array back on the screen at any place with

\[\text{PUT}(X_1, Y_1)-(X_2, Y_2), A, \text{action}\]

PUT upper left corner here
PUT lower right corner here
use this array
tells the computer what action to take: PSET, PRESET, AND, OR, or NOT

Now, let's see how it works in a program. We'll first draw our figure in
the upper left corner of the screen. Then we'll move it to the lower right corner.

```
100  'SET SCREEN AND DIMENSION ARRAY
110   PMODE 4, 1
120   PCLS
130   SCREEN 1, 0
140   DIM A(20,20)

200  'DRAW FIGURE
210   DRAW"BM10,20;R10D10L10U10"
220   DRAW"E10R10G10D10E10U10"  --green on black
230   PAINT(12,22),1,1   --20 wide and 20 high

300  'STORE IT
310   GET(10,10)-(30,30),A,G

400  'PUT IT IN LOWER CORNER
410   PUT(210,160)-(230,180),A,PSET

500  'LOOP
510   GOTO 510
```

Enter and run the program. The figure is copied from the upper left corner to the lower right. Now you have two figures on the screen. If you are going to simulate motion, you have to be able to erase the old figure before drawing a new one. One way to do this is to use another array with no drawing on it and place this area over the first figure. You can add to the previous program the lines

```
150   DIM B(20,20)  a blank array
320   PUT(10,10)-(30,30),B  PUT it over the first
```
When the program is executed now

![Diagram](https://via.placeholder.com/150)

Next, move it in much shorter steps. Using FOR-NEXT loops, we'll move the figure across the screen instead of from corner to corner.

```
100  'SET SCREEN AND DIMENSION ARRAY
110  PMODE 4, 1
120  PCLS
130  SCREEN 1, 0
140  DIM A(20,20)
150  DIM B(20,20)
200  'DRAW FIGURE
210  DRAW"BM10,20; R10D10L10U10"
220  DRAW"E10R10G10D10E10U10"
230  PAINT(12,22), 1, 1
300  'STORE ARRAY A
310  GET (10, 10) - (30, 30), A, G
400  'MOVE IT ACROSS THE SCREEN
410  Y=10
420  FOR X = 10 TO 220 STEP 30
430       PUT (X,Y) - (X+20, Y+20), A, PSET  ←PUT it on
440       FOR W = 1 TO 50: NEXT W
450       PUT (X,Y) - (X+20, Y+20), B  ←erase it
460  NEXT X
500  'DO IT AGAIN
510  GOTO 420
```

Now you see
It moves across the screen from left to right and repeats the action over and over.

Exercise 6-1

Change, or add, lines in the program so that the figure will move across the screen from left to right as before. When it reaches the right side of the screen, have it return to the left side, but 30 Y positions below the previous placement. In other words

Changes and additions:

(Answers are at the end of the chapter.)

The movement in the exercise may be rather jumpy because you were moving the figure in steps of 30 locations. The steps can be shortened by changing the STEP parameter in line 420 to

420 FOR X = 10 TO 220 STEP 8

You should experiment with the time delay at line 440, for the FOR-NEXT STEP in line 420, and line 450, which erases the figure. The variation that follows puts a new figure on the screen before erasing the last one. The effect may be more pleasing to your eyes. The changes are

420 FOR X = 10 TO 220 STEP 4
440 FOR W = 1 TO 10: NEXT W
450 PUT (X-4,Y) - (X+16,Y+20),B

The changes in line 450 match the STEP.

DRAW IT SMALL

When you are using PMODE 4, you can draw small, recognizable shapes. The next program draws a chair at the bottom, center of the screen. A hook is suspended at the top center of the picture.
As the program progresses, the hook is lowered and attached to the chair back.

Then it raises the chair to the ceiling.

Enter and run the program.

A CHAIR-RAISING PROGRAM

100 'SET THE SCREEN AND DIMENSION
110 PMODE 4, 1
120 PCLS
130 SCREEN 1, 0
140 DIM A(8,8): DIM B(20,30): DIM C(12,2)
200 'DRAW FIGURES
210 GOSUB 1000
220 PAINT (112, 182), 1, 1
230 FOR W=1 TO 200: NEXT W
240 LINE (100, 10) - (120, 15), PSET, BF
Exercise 6-2

Add a section to the chair-raising program that would lower the chair to the floor and raise the empty hook again. Fill in the additions in the exercise. Add

600 ' LOWER HOOK AND CHAIR
610 GET (__________________) - (________________), B,G
620 FOR Y = _________ TO _________ STEP ________
630 PUT (__________________) - (________________), B, PSET
640 PUT (__________________) - (________________), C
650 LINE (__________________) - (________________), PSET
660 FOR W = 1 TO 25: NEXT W
670 NEXT Y
PUT, GET, AND JOYSTICKS

700 ’ RAISE EMPTY HOOK
710 FOR Y = ________ TO ________ STEP ________
720 PUT (__________) - (__________), ___, ___
730 PUT (__________) - (__________), ___
740 IF Y = 160 GOSUB 900
750 FOR W=1 TO 10: NEXT W
760 NEXT Y
800 ’ LOOP
810 GOTO 810
900 ’ CHAIR FIX
910 GET (220, 0) - (232, 2), C ← get a blank area
920 FOR Q=160 TO 170 STEP 2
930 PUT (110, Q) - (112,Q+2),C
940 NEXT Q

(Answers are at the end of the chapter.)

No doubt you can think of many changes that you would like to make to this program. You might want to change the hook’s retriever into a horizontally moving crane that would move objects from place to place, such as

![Diagram of crane]

1 2 3 4 5, etc.

THE JOYSTICKS

Joysticks may be used with Radio Shack’s TRS-80 Color Computer. There is only one statement used to read the coordinates of the joysticks, but four possible values can be used to give different readings:

\[
\text{JOYSTK}(x) = \begin{cases} 0, & \text{left} \\ 1, & \text{up} \\ 2, & \text{right} \\ 3, & \text{down} \end{cases}
\]

\(x\) is an integer (0,1,2, or 3)
The convention used by the color computer to interpret the joystick readings is

- **JOYSTK(0)** returns the horizontal position of the left joystick.
- **JOYSTK(1)** returns the vertical position of the left joystick.
- **JOYSTK(2)** returns the horizontal position of the right joystick.
- **JOYSTK(3)** returns the vertical position of the right joystick.

Horizontal and vertical positions are returned as values from 0 through 63 as the joysticks are moved from left to right (horizontal) or up and down (vertical).

You should experiment to see how the joysticks are connected. To do this, insert both joystick connectors at the appropriate places as marked on the rear of the TRS-80 keyboard.

![Figure 6-1. Rear view of TRS-80 Color Computer.](image)

Place the joystick controls within reach with one to the left of the other. Arrange them so that the pushbuttons are to the rear.

The joysticks move freely from left to right and up and down so that they can be rotated through 360°. First move both sticks to the left (horizontal) and center (vertical). If you are clock-oriented, that would be the 9 o'clock position.
After you have done this, turn on the computer and enter this program:

```
100    'SET UP JOYSTICKS
110    CLS
120    PRINT @0, JOYSTK(0)
200    'READ JOYSTICK AGAIN
210    GOTO 120
```

You now are going to experiment in order to discover which joystick the computer is using as the left one. If you have entered the program and placed both joysticks to the left-center position as shown in Figure 6-2, type RUN. You should see

```
0
```

Slowly move the left joystick from left to right. Does the number on the screen change? If it does not, exchange the positions of the two controllers. Then try the controller on the left.

As you move the left joystick slowly to the right, the number in the upper left corner should increase from 0 through 63 (when the joystick reaches the extreme right).
You now know which joystick the computer considers to be the left one. Now add this line to read the right joystick:

```
140 PRINT @16,JOYSTK(2)
```

Run the program again. This time move the right joystick slowly from left to right. We left the left joystick in the right-most position as we moved the right joystick and saw

![Diagram showing joystick positions]

You now have full control of the horizontal positions of both joysticks. Let's try for vertical control. Add these lines to your program:

```
130 PRINT @32,JOYSTK(1)
```

vertical, left joystick
Exercise 6-3

With both joysticks connected, enter the revised program (if you have not already done so):

100 'SET UP STICKS
110 CLS
120 PRINT @0,JOYSTK(0)
130 PRINT @32,JOYSTK(1)
140 PRINT @16,JOYSTK(2)
150 PRINT @48,JOYSTK(3)
200 'READ JOYSTICKS AGAIN
210 GOTO 120

Fill in the screen values for these conditions:

<table>
<thead>
<tr>
<th>Joysticks</th>
<th>Screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td></td>
</tr>
<tr>
<td>Left</td>
<td>Right</td>
</tr>
<tr>
<td>left (H)</td>
<td>center (H)</td>
</tr>
<tr>
<td>½ up (V)</td>
<td>up (V)</td>
</tr>
<tr>
<td>b.</td>
<td></td>
</tr>
<tr>
<td>Left</td>
<td>Right</td>
</tr>
<tr>
<td>¾ right (H)</td>
<td>center (H)</td>
</tr>
<tr>
<td>¾ up (V)</td>
<td>down (V)</td>
</tr>
<tr>
<td>c.</td>
<td></td>
</tr>
<tr>
<td>Left</td>
<td>Right</td>
</tr>
<tr>
<td>¼ right (H)</td>
<td>¾ right (H)</td>
</tr>
<tr>
<td>¼ up (V)</td>
<td>¼ up (V)</td>
</tr>
</tbody>
</table>
Now that you are in full control of the joystick controller, let's move on to the push buttons.

**PUSH BUTTONS**

In addition to the `JOYSTK` statement, memory location 65280 is assigned to the push buttons of each controller. When neither push button is pressed, the value in memory location 65280 is either 127 or 255. If the left button only is depressed, the value in memory location 65280 changes to either 126 or 254. If the right only is pressed, the value changes to 125 or 253. If both buttons are pressed at once, the value changes to 124 or 252. Therefore, by `PEEKing` at memory location 65280, you can tell whether the left, right, both, or neither push buttons are depressed.

Add to the previous joystick program

```
160 PRINT @68,PEEK(65280)
```

With no buttons pushed, you see

<table>
<thead>
<tr>
<th>ww</th>
<th>xx</th>
<th>or</th>
<th>yy</th>
<th>zz</th>
<th>255</th>
</tr>
</thead>
</table>

With the left button only pushed, you see

<table>
<thead>
<tr>
<th>ww</th>
<th>xx</th>
<th>or</th>
<th>yy</th>
<th>zz</th>
<th>254</th>
</tr>
</thead>
</table>

(Answers are at the end of the chapter.)
With the right button only pushed, you see

```
W W
Y Y
253
```

or

```
W W
Y Y
```

With both push buttons pushed, you see

```
W W
Y Y
252
```

or

```
W W
Y Y
```

Let's work first with the left controller only. We'll write a program to utilize all features. Remember that the joysticks give values from 0 through 63, but high-resolution graphics screen positions go from 0 through 255 horizontally and 0 through 191 vertically. By multiplying the horizontal joystick value by 4 and the vertical joystick value by 3, you will get the values needed to scan the complete video display.

COLOR SKETCH

```
100 'SET SCREEN
110 PMODE 1,1
120 PCLS
130 SCREEN 1,1
200 'ASSIGN JOYSTICKS
210 H::: JOYSTK(0)
220 V::: JOYSTK(1)
300 'TEST PUSH BUTTON FOR COLOR
310 C::: 5
320 B::: PEEK(65280)
330 IF B:::254 OR B:::126 THEN C:::8
400 'SET POINT OR ERASE
410 PSET (4*H,3*V,C)  "set background or color from line 330"
500 'DO IT AGAIN
510 GOTO 210
```
HOW TO USE THE JOYSTICKS

There are many possible variations if both joystick controllers are used. You might use the left joystick for the horizontal and vertical screen positions as in the last program. The right joystick could select a color for the point to be set. This is done in the following program. The left push button is used to clear the entire screen for a new picture.

**PLAYFUL SKETCHING**

100  'SET SCREEN
110   PMODE 1,1
120   PCLS
130   SCREEN 1,1
200  'ASSIGN JOYSTICK VALUES
210   H = JOYSTK(0)
220   V = JOYSTK(1)
230   R = JOYSTK(2)
240   S = JOYSTK(3)
300  'TEST FOR COLOR
310   IF R>=31 AND S<31 THEN C=5 ← erase with buff
320   IF R<31 AND S<31 THEN C=6 ← cyan
330   IF R<31 AND S>=31 THEN C=7 ← magenta
340   IF R>=31 AND S>=31 THEN C=8 ← orange
400  'TEST PUSH BUTTON
410   B=PEEK(65280)
420   IF B=254 OR B=126 THEN PCLS
500  'SET POINT AND GO BACK
510   PSET (4*H,3*V,C)
520   GOTO 210

You can select the color you want with the right joystick and the position you want with the left joystick. To erase the screen, use the left push button.

Another possible use of the push buttons is to call a subroutine to perform some action. In the playful sketching program, you might use the 400 section to call one of three subroutines as follows:

410   B=PEEK(65280)
420   IF B=254 OR B=126 THEN GOSUB 1000
430   IF B=253 OR B=125 THEN GOSUB 2000
440   IF B=252 OR B=124 THEN GOSUB 3000
1000 'DRAW A CIRCLE
1010 CIRCLE(4*H,3*V),10,C
1020 PAINT(4*H+2,3*V+2),C,C
1030 RETURN
2000 'DRAW A RECTANGLE
2010 COLOR C,5
2020 LINE(4*H,3*V) – (4*H+10,3*V+10),PSET,BF
2030 RETURN
3000 'DRAW A TRIANGLE
3010 Z$=STR$(4*H):V$=STR$(3*V)
3020 COLOR C,5
3030 DRAW"BM"+Z$+","+V$+"E10F10L 10"
3040 PAINT(4*H+2,3*V-2),1,1
3050 RETURN

Now you can choose the color with the right joystick, choose horizontal and vertical positions with the left joystick, draw a circle with the left push button, draw a rectangle with the right push button, and draw a triangle by pressing both push buttons.

USING JOYSTICKS IN GAMES

The next program creates a test for your motor skills. Three objects (one cyan, one magenta, and one orange) are drawn on the lower part of the screen.

An electromagnet is positioned on the screen according to the setting of the left joystick. Electricity for the magnet is turned on when you press the push button on the left controller. You begin by placing the joystick in the approximate center position (both horizontal and vertical).
The magnet will then appear in approximately the center of the screen.

![Blinking Magnet Diagram]

The object of the game is to rearrange the objects in these two orders:

1. magenta (left), orange (center), and cyan (right)
2. orange (left), cyan (center), and magenta (right)

The magnet may be moved by the joystick left to right and up and down. The objects are rearranged by moving one of them at a time with the magnet. If the magnet is moved to a position just above one of the objects and “turned on” by pressing the push button, the object will be attracted to the magnet. It may then be picked up and moved.

If the magnet is close enough to attract an object, both magnet and object will blink when the push button is pressed.

Caution: If you get too close, you may destroy the object. If you do not get close enough, you may attract only part of the object. Both of these are mistakes. If you latch the object correctly, you may pick it up and move it out of the way temporarily with the magnet.

Press the push button again to detach the object (the object stops blinking). You then may move the magnet to a position above another object.
Pick up the second object and move it to the desired position (to the left in this example).

Press the push button to detach the object, and go get the third object.

Pick up the third object and move it (to the center in this example).

Detach the object and go get the one that you left in midair. Move it to the right position, detach it and you are finished.

We found that the program was easier to write than to use without destroying the objects. With a little practice, however, the objects became controllable.

**ELECTRO-MAGNET**

100 'SET SCREEN;DIMENSION ARRAYS
110 PMODE 1,1
Because of the popularity of shooting games, we can't overlook these applications of joystick operation. However, we'll be nonviolent by restricting our demonstration program to target practice. This program moves a target left or right at the top of the screen. The left joystick will control the position of a moving gun that can be fired by pressing the controller's button. We've restricted the missile's movement to straight up. Also, you can have only one missile in flight at a given time. The target appears at a random loca-
Figure 6-3. Fire stations flowchart.
tion and may move either left or right until it disappears from the screen. It will reappear in a new random position and move again either left or right. A flowchart of the program is shown in Figure 6-3. Here is the program:

```
100 'SET SCREEN
110 PMODE 4,1
120 PCLS
130 SCREEN 1,0

200 'SET VARIABLES
210 MS=0: R=126
220 A=RND(2): B=RND(128)+30
230 IF A=1 THEN C=250: D=-8
240 IF A=2 THEN C=0: D=-8

300 'ACTION
310 FOR Z=B TO C STEP D
320 'TARGET ON
330 LINE(Z,10) - (Z+15,14),PSET,BF
340 'LOOK FOR SHOTS
350 FOR Q = 1 TO 3
360 'ERASE GUN
370 COLOR 0,1
380 LINE(R,180) - (R+5,184),PSET,BF
390 'READ JOYSTICK
400 H=JOYSTK(0): R=4*H
410 'REDRAW GUN-NEW POSITION
420 COLOR 1,0
430 LINE(R,180) - (R+5,184),PSET,BF
440 'START OR MOVE MISSILE
450 IF MS=1 GOTO 710
460 IF PEEK(65280) =254 OR PEEK(65280) =126 THEN
   X=R+2: Y=176: CIRCLE(X,Y),2: MS=1
470 NEXT Q

500 'ERASE TARGET
510 COLOR 0,1: LINE(Z,10) - (Z+15,14),PSET,BF
520 COLOR 1,0
530 NEXT Z

600 'NEW TARGET
610 GOTO 220

700 'MOVE MISSILE UP
710 CIRCLE(X,Y),2,0: Y=Y-12
```
PUT, GET, AND JOYSTICKS

150 'SEE IF IN RANGE OF TARGET
730 IF X>=Z and X<=Z+15 AND Y<=15 GOTO 810
740 'SEE IF MISSILE MISSED
750 IF Y<10 THEN MS=0: GOTO 470
760 'DRAW NEW MISSILE POSITION
770 CIRCLE(X,Y),2,1
780 GOTO 470
800 'BLINK TARGET—IT'S HIT
810 FOR X=1 TO 10
820 LINE(Z,10) - (Z+15,14),PRESET,B
830 FOR W=1 TO 20:NEXT W
840 LINE(Z,10) - (Z+15,14),PSET,B
850 NEXT X
900 'START ANEW
910 GOTO 110

You can make many modifications to this basic target shooting program. You could easily set a given number of missile appearances, keep score (-1 for a miss, +1 for a hit), change the size of the target, change the movement of target or missile, etc.

Have fun playing with the program before going on to the chapter summary and chapter test.

Summary

In this chapter, you learned two ways to move objects about the video screen. You learned to GET an object within a rectangular area of the screen and to PUT it somewhere else on the screen. You learned how to read the horizontal and vertical positions of the joysticks and to use this information to cause some action. You also learned to PEEK at the on-off condition of the joystick's push buttons and to use the result. You learned

- that the GET statement allows you to access the graphic information in a rectangular area of the screen and to store the information in an array previously dimensioned as

  \[
  \text{DIM A (10,10)}
  \]

  \[
  \text{GET(5,25) - (15,35),A,G}
  \]

- that the PUT statement allows you to PUT the information stored in an array back on the screen at a specified location:

  \[
  \text{PUT(125,100) - (135,110),A,PSET}
  \]
You also learned

- to alternate a blank (or empty) array and an array containing graphics information to create apparent movement of an object on the video display:
  
  ```plaintext
  DIM A(10,10), B(10,10)  dimension 2 arrays
  GET(5,25) - (15,35), A, G  store graphics info
  PUT(5,25) - (15,35), B  erase 10x10 area
  PUT(25,25) - (35,35), A, PSET  move graphics
  ```

- the convention used to interpret the joystick readings:
  
  ```plaintext
  JOYSTK(0)  horizontal, left joystick (0-63)
  JOYSTK(1)  vertical, left joystick (0-63)
  JOYSTK(2)  horizontal, right joystick (0-63)
  JOYSTK(3)  vertical, right joystick (0-63)
  ```

- to interpret the on-off condition of the push buttons:
  
  ```plaintext
  IF PEEK(65280)=127 OR 255—they buttons off
  IF PEEK(65280)=126 OR 254—left button on
  IF PEEK(65280)=125 OR 253—right button on
  IF PEEK(65280)=124 OR 252—both buttons on
  ```

- to use the joysticks to maneuver objects about the screen

Chapter Test

1. The graphics details of a circle drawn by the following CIRCLE statement are to be stored in an array named A:

   CIRCLE(120,90), 20

   Complete the DIM statement to provide the minimum dimensions necessary to store the circle in the array:

   DIM

2. The graphics details of a rectangle drawn by the following LINE statement are to be stored in array B:

   LINE(10,20) - (30,25), PSET, B

   Complete the DIM statement that would dimension array B:

   DIM
3. Lines 110-220 of the following program draw two lines on the screen as shown in the sketch on the left. Use GET and PUT statements to complete the program in order to draw the figure in the sketch on the right.

```
100 'SET SCREEN AND DIMENSION ARRAYS
110 PMODE 4,1
120 PCLS
130 SCREEN 1,0
140 DIM A(10,2),B(2,10)
200 'DRAW LINES
210 DRAW"BM10,10R10;BM250,10D10"
300 'GET LINES
310
320
400 'PUT LINES
410
420
430
440
500 GOTO 500 'LOOP
```

4. Add line 150 to dimension a blank array, and add lines 330 and 340 to PUT the blank array so that the original lines in the upper left and upper right corners of the screen are erased.

```
150 DIM 
330 PUT 
340 PUT
```

5. Add line 160 to the program of exercises 3 and 4 to dimension an array that will be large enough to GET the figure in the center of the screen.
Then add line 450 to GET the figure at the center of the screen, and add lines 460, 470, 480 and 490 to PUT it in each corner of the screen.

6. Rewrite the program of exercises 3, 4 and 5 using PMODE and four pages of graphics memory so that the center figure stays “on” and the corner figures “come on” one at a time in succession.

page 1

page 2

page 3

page 4

Your program:

7. Fill in the conventions used to read the joystick statements:
   a. JOYSTK(0)
   b. JOYSTK(2)
   c. JOYSTK(1)
   d. JOYSTK(3)
8. The following JOYSTK statements are used in a program like this:

\[
\begin{align*}
310 \ H &= \ JOYSTK(0) \\
320 \ V &= \ JOYSTK(1) \\
410 \ \text{IF} \ H > = 31 \ \text{AND} \ V > = 31 \ \text{THEN} \ C = 1 \\
420 \ \text{IF} \ H > = 31 \ \text{AND} \ V < 31 \ \text{THEN} \ C = 2 \\
430 \ \text{IF} \ H < 31 \ \text{AND} \ V > = 31 \ \text{THEN} \ C = 3 \\
440 \ \text{IF} \ H < 31 \ \text{AND} \ V < 31 \ \text{THEN} \ C = 4 \\
510 \ \text{CIRCLE}(128,96),10,C
\end{align*}
\]

PMODE 1,1 and SCREEN 1,0 are being used. Give the color that would be used to draw the circle if the joysticks are in the positions shown:

[Diagram showing four positions of joysticks]

color = _____  color = _____  color = _____  color = _____

9. Enter and run the following program using the joysticks. Describe the resulting display.

\[
\begin{align*}
100 \ \text{PMODE} \ 1,1 \\
120 \ \text{PCLS} \\
130 \ \text{SCREEN} \ 1,0 \\
210 \ H &= \ JOYSTK(0) : \ R = 4*H \\
220 \ V &= \ JOYSTK(1) : \ S = 3*V \\
310 \ \text{IF} \ H > = 31 \ \text{AND} \ V > = 31 \ \text{THEN} \ F = 4 \\
320 \ \text{IF} \ H > = 31 \ \text{AND} \ V < 31 \ \text{THEN} \ F = 3 \\
330 \ \text{IF} \ H < 31 \ \text{AND} \ V > = 31 \ \text{THEN} \ F = 2 \\
340 \ \text{IF} \ H < 31 \ \text{AND} \ V < 31 \ \text{THEN} \ F = \text{RND}(3)+1 \\
410 \ \text{IF} \ \text{PEEK}(65280) = 254 \ \text{OR} \ \text{PEEK}(65280) = 126 \ \text{THEN} \\
\text{CIRCLE}(R,S),15,F: \ \text{PAINT}(R,S),F,F \\
420 \ \text{GOTO} \ 210
\end{align*}
\]

DISPLAY: _______________________________________

_______________________________________

10. Modify the shooting gallery program of this chapter to allow 20 appearances of the target. Keep track of the number of hits. Display the score
at the end of the 20 appearances. (A solution to this program will appear in a future issue of *The Dymax Gazette.* Send us your solution; it may be published.

Answers to Exercises Within the Chapter

Exercise 6-1

Change line 410 to 410 FOR Y = 10 TO 170 STEP 30
Add 470 NEXT Y

Exercise 6-2

610 GET(106,16) - (126,46),B,G
620 FOR Y = 18 TO 160 STEP 2
630 PUT(106,Y) - (126,Y+30),B,PSET
640 PUT(106,Y) - (118,Y),C
650 LINE(110,14) - (110,Y+2),PSET
710 FOR Y=160 TO 20 STEP -2
720 PUT(106,Y-2) - (114,Y+6),A,PSET
730 PUT(106,Y+6) - (118,Y+8),C

Exercise 6-3

a.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td>31</td>
<td>0</td>
</tr>
</tbody>
</table>

(answers approximate)

b.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>31</td>
</tr>
<tr>
<td>15</td>
<td>63</td>
</tr>
</tbody>
</table>

c.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>45</td>
</tr>
<tr>
<td>45</td>
<td>45</td>
</tr>
</tbody>
</table>

d.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>15</td>
</tr>
<tr>
<td>31</td>
<td>15</td>
</tr>
</tbody>
</table>
Answers to Odd-Numbered Exercises in Chapter Test

1. DIM A(40,40)  
   If radius is 20, the array must be 40 wide by 40 high.

3. 310 GET(10,10) - (20,12), A,G  
   320 GET(250,10) - (252,20), B,G  
   410 PUT(130,96) - (140,98), A, PSET  
   420 PUT(116,96) - (126,98), A, PSET  
   430 PUT(128,84) - (130,94), B, PSET  
   440 PUT(128,98) - (130,108), B, PSET

5. 160 DIM D(24,24)  
    450 GET(118,84) - (140,108), D, G  
    460 PUT(0,0) - (24,24), D, PSET  
    470 PUT(231,0) - (255,24), D, PSET  
    480 PUT(0,167) - (24,191), D, PSET  
    490 PUT(231,167) - (255,191), D, PSET

7. JOYSTK(0) left joystick, horizontal  
   JOYSTK(2) right joystick, horizontal  
   JOYSTK(1) left joystick, vertical  
   JOYSTK(3) right joystick, vertical

9. With joystick in upper right, blue circle appears when push button is pressed.  
   With joystick in lower right, red circle appears.  
   With joystick in lower left, yellow circle appears.  
   With the joystick in the upper left, a random colored circle appears; if 
   the push button is held, the color of the circle will randomly change.
This chapter will provide some diversion. Two statements are introduced that can produce sounds sent from the computer to your TV set. After learning how to use all the options with these two statements, you will learn to coordinate sound with graphic movements. By the end of the chapter, you will know

- how to produce a musical note with a given duration by the SOUND statement
- how to use the SOUND statement with graphics
- how to use options in PLAY statements to produce a series of sounds of varying notes, octaves, note lengths, and volumes
- how to use PLAY statement options to change tempo, insert rests, and include substrings
- how to add sounds to graphic programs previously used
- how to add sounds to new graphic programs

**SOUND**

You can use two functions with Extended Color BASIC to produce a wide variety of sounds. The first of these, called SOUND, is available from either the 4K Color BASIC or the Extended Color BASIC of the 16K Color Computer.
The SOUND function may be used to produce a specified note for a specified duration. It has the format

\[ \text{SOUND N,D} \]

Note, a value from 1-255
Duration, a value from 1-255

The note values produce low notes for low numbers and high notes for high numbers. The duration values work the same way. Low numbers produce short durations, and high numbers produce long durations. In fact, high numbers produce very long durations.

To hear the range of notes possible with the SOUND function, try this program:

```
100 'SET THE DURATION
110 D = 1 ← the shortest possible duration
200 'PLAY ALL NOTES
210 FOR N = 1 TO 255
220 SOUND N,D
230 NEXT N
```

From this program, you can tell that it is possible to use the SOUND function to create many notes. Thus, it is possible to create simple melodies. The Radio Shack manual *Getting Started with Color BASIC* shows the numerical values for notes corresponding to a piano keyboard in Appendix A.

Here is a short program that plays random "music" for you. I hope that it will motivate you to produce something more original.

```
100 'SET DURATION
110 D = 2
200 'CHOOSE A RANDOM NOTE
210 N = RND(RND(145))+100
300 'PLAY THREE QUICK NOTES
310 SOUND N,D
320 SOUND N+10,D
330 SOUND N-10,D
400 'REPEAT
410 GOTO 210
```

It may not take you long to tire of this random music. When you do, try producing some creations. It might be interesting to print the notes chosen by line 210 of the preceding program.
ADDING SOUND TO GRAPHICS

You can add sound to your graphics programs to enliven them. Here is a program that draws a musical staff and draws the notes on it as they are played by the SOUND function.

```
100 'SET GRAPHICS SCREEN
110 PMODE 1,1
120 PCLS
130 SCREEN 1,0
200 'DRAW STAFF
210 FOR Y = 56 TO 88 STEP 8
220 LINE(12,Y) - (240,Y),PSET
230 NEXT Y
240 LINE(12,96) - (40,96),PSET
300 'DRAW AND PLAY NOTE
310 Y = 96
320 FOR X = 24 TO 220 STEP 28
330 READ N
340 CIRCLE(X,Y),5,2
350 SOUND N,3
360 Y = Y-4
370 NEXT X
400 'NOTE DATA
410 DATA 80, 108, 125, 133, 147, 159, 170, 176
500 'LOOP HERE
510 GOTO 510
```

Enter and play the scale. It's not very exciting, is it? You can enliven it by adding to the program as outlined in the next exercise.

Exercise 7-1

Starting with line 500 of the previous program, modify it by erasing the screen, redrawing the staff, and playing the octave in reverse.

```
500 'CLEAR SCREEN
510
```
THE PLAY FUNCTION

The PLAY function has capabilities for sound that are similar to those of the DRAW function for graphics. Many options can be used to fit desired objectives. The PLAY function is much more versatile than SOUND. Here is a brief description of the PLAY options.

PLAY “sound string”

The options are:
- note, A through G or 1 through 12
- octave, Ox where x = 1 through 5
- note length, Lx where x = 1 through 255
- tempo, Tx where x = 1 to 255
- volume, Vx where x = 1 through 31
- pause length, Px where x = 1 through 255
- substring, XA$ a substring (A$) of PLAY commands

(Answers are at the end of the chapter.)
Let's start by playing the full five octaves of natural notes, sharps, and flats. On a piano, one octave would be

```
2 4 7 9 11
3 5 6 8 10 12
```

The numbers 1 through 12 correspond to the notes in Table 7-1. You can use either letters or the numerical equivalents in the PLAY string.

<table>
<thead>
<tr>
<th>Number</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>C# or D-</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>4</td>
<td>E- or D#</td>
</tr>
<tr>
<td>5</td>
<td>E or F-</td>
</tr>
<tr>
<td>6</td>
<td>F or E#</td>
</tr>
<tr>
<td>7</td>
<td>F# or G-</td>
</tr>
<tr>
<td>8</td>
<td>G</td>
</tr>
<tr>
<td>9</td>
<td>G# or A-</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>A# or B-</td>
</tr>
<tr>
<td>12</td>
<td>B</td>
</tr>
</tbody>
</table>

This program will print the octave and note values as they are played. All five octaves are played with naturals, sharps, and flats.

```
100 'SELECT NOTE LENGTH
110 INPUT "NOTE LENGTH";L
120 CLS
200 'OCTAVE AND NOTE LOOPS
210 FOR O = 1 TO 5
220 PRINT @5, "OCTAVE ";O
230 FOR N = 1 TO 12
240 PRINT @37, "NOTE ";N
250 PLAY "O"+STR$(O)+";L"+STR$(L)+";"+STR$(N)
260 NEXT N
270 NEXT O
300 GOTO 110
```
On the screen, you will see the octave and note values displayed as they are changed.

\[
\begin{align*}
\text{OCTAVE} \ y &\quad \leftarrow \text{octave, 1 through 5} \\
\text{NOTE} \ xx &\quad \leftarrow \text{note, 1 through 12 for each octave}
\end{align*}
\]

When you run the program, you will be asked to input a value for the note length. This value may be any integer from 1 through 255. You should be warned, however, that low values produce long durations. High values produce short durations. Experiment with the note length.

You may want to print the note length on the screen so that you remember the last value used. Using a value of 255 for \( L \) will demonstrate how much faster notes can be played with \texttt{PLAY} as compared to \texttt{SOUND}.

The tempo controls the speed at which the complete, or partial, \texttt{PLAY} statement is executed. If no tempo is specified, a value of 2 is used. Any integer from 1 through 255 may be used. Low numbers give a slow tempo. High numbers give a fast tempo.

The note length controls the duration of individual notes. The range of values are shown in Table 7-2. The specified tempo and note length stay in effect until a new value is specified.

**TABLE 7-2. NOTE LENGTH VALUES**

<table>
<thead>
<tr>
<th>Length Specified</th>
<th>Note Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>whole note</td>
</tr>
<tr>
<td>L2</td>
<td>1/2 note</td>
</tr>
<tr>
<td>L3</td>
<td>1/3 note</td>
</tr>
<tr>
<td>L4</td>
<td>1/4 note</td>
</tr>
<tr>
<td>L8</td>
<td>1/8 note</td>
</tr>
<tr>
<td>L16</td>
<td>1/16 note</td>
</tr>
<tr>
<td>L255</td>
<td>1/255th note</td>
</tr>
</tbody>
</table>
Here are two tunes using tempo and note length. One tune is written with numbers designating the notes. The second uses letters.

The second tune also uses pauses (or rests). These are designated by the command

\[ P_x \]

This is a value from 1 through 255 and corresponds to the note lengths in Table 7-2.

**This Old Man**

100 'T FOR TEMPO; L FOR NOTE LENGTH; O FOR OCTAVE
110 PLAY"T4;O3;L4;8;5;L2;8;L4;8;5;L2;8"
120 PLAY"L4;10;8;6;5;3;5;L2;6"
130 PLAY"O2;L4;8;1;1;1;5;L2;8"
140 PLAY"L4;8;3;3;6;5;3;L2;1"

**A Jig**

200 'P FOR PAUSE; NOTES BY LETTER
210 PLAY"T6;O3;L4;D;D;O2;B;G;A;C;B;G;O3;D;D"
220 PLAY"O2;B;G;A;O3;C;O2;L2;A;O3;L4;D;D"
230 PLAY"O2;B;G;A;O3;C;O2;B;G;A;O3;C"
240 PLAY"O2;B;G;A;O3;D;O2;L2;G"
250 FOR X = 1 TO 2 'REPEAT THIS PART
260 PLAY"P4;L4;G;A;O3;C;O2;P4;A;B;O3;D;P4;O2;B"
270 PLAY"A;O3;C;C;O2;A;B;O3;D;P4;O2;G"
280 PLAY"A;O3;C;P4;O2;A;B;O3;D;P4;O2;B"
290 PLAY"A;B;P4;A;D;P4"
300 NEXT X

You may notice that several symbols repeat in “A Jig.” For instance, the sequence O2;B;G appears in lines 210, 220, 230, and 240. The sequence A;O3;C appears in lines 220, 230, 260, 270, and 280. The sequence O2;A;B;O3;D;P4;O2 appears in lines 270 and 280.

The PLAY function can use sound substrings in the same way that the DRAW function used graphic substrings. The form of such a substring in a PLAY statement is

\[ XA$ \]

A$ is the name of a substring that has been previously defined

X signifies a substring
The program, "A Jig," when modified for the suggested substrings, becomes

```
200  'DEFINE STRINGS
210  A$ = "02;B;G;"
220  B$ = "A;O3;C"
230  C$ = "O2:A;B;O3;D;P4;O2"
300  'PLAY THIS PART ONCE
310  PLAY"T6;O3;L4;D;D;XA$;A;C;B;G;O3;D;D."
320  PLAY"XA$;XB$;O2;L2;A;O3;L4;D;D"
330  PLAY"XA$;XB$;O2;B;G;A;O3;C"
340  PLAY"XA$;A;O3;D;O2;L2;G"
400  'PLAY THIS PART TWICE
410  FOR X=1 TO 2
420  PLAY"P4;G;XB$;O2;P4;A;B;O3;D;P4;O2;B"
430  PLAY"XB$;C;XC$;G"
440  PLAY"XB$;P4;XC$;B"
450  PLAY"A;B;P4;A;D;P4"
460  NEXT X
```

Music by a computer is fine, but it can be produced much better by a musical instrument, a phonograph record, or a tape recorder. However, the computer can combine graphics and sound to produce some interesting effects.

Let's add some simple graphics to "This Old Man." We'll draw an old man and have him indicate the verse being played:

This old man, he played one .........
This old man, he played two .........
This old man, he played three .........

This Old Man with Graphics

```
100  'SET MODE AND CLEAR SCREEN
110  PMODE 0,1
120  PCLS
200  'DRAW HAT
210  CIRCLE(110,20),8,,1.8,,4,,9
220  CIRCLE(120,20),8,,1.8,,7,0
230  CIRCLE(115,25),32,,3,,9,,7
```

Note how much shorter this part is than the original
300 'DRAW MAN AND CANE
310 DRAW"BM100,35;D1SF15R2E16U10"
320 DRAW"BM100,35;F9U4R4L4U4R4D2R8U2R4
   D4L4U4R4D2E9"
330 DRAW"BM116,40;D6R2U6"
340 DRAW"BM109,55;E6F6"
350 LINE(100,50) - (60,60),PSET
360 LINE-(70,80),PSET
370 LINE(100,60) - (75,65),PSET
380 LINE-(80,80),PSET
390 LINE-(70,80),PSET
400 DRAW"BM70,80;F5R10H5"
410 DRAW"BM78,58;D10F5D10G5D10F5D6"
420 DRAW"R3U6H5U10E5U10H5U10"
430 DRAW"BM100,60;D30R30U30"
440 DRAW"BM103,90;D40R10U35R6D35R10U40"
450 DRAW"BM104,130;F6R6H6"
460 DRAW"BM122,130;F6R6H6"
470 DRAW"BM135,50;R20E15H9R2F7R1O2D2G25L26"

500 'PLAY 3 VERSES OF THIS OLD MAN
510 FOR P = 1 TO 3
520   PMODE 0,P
530   SCREEN 1,1
540   A$ = STR$(166+4*P)
550   PLAY"T4;O3;L4;8;5;L2;8;L4;8;5;L2;8"
560   DRAW"BM"+A$+";33;U5" ← raise a finger
570   PLAY"L4;10;8;6;5;3;5;L2;6"
580   PLAY"O2;L4;8;1;1;1;1;5;L2;8"
590   PLAY"L4;8;3;3;6;5;3;L2;1"
600 NEXT P

As you can see when you run this program, producing high-resolution graphics takes a lot of work for all but the simplest of figures. The only moving parts of this program are the moving fingers to signify the verses 1, 2, and 3. If you want to make the old man dance a jig, it will take much more programming.

OTHER NOISES

Perhaps music is not the best application of the PLAY function. Other noises can be produced to simulate sounds heard in the real world. For instance, a metronome can be simulated by the following simple program:
Play with the metronome program, using different values in the delay loop and using different noises. Then enter the following program that displays a clock face and uses a similar sound. The program is a variation of exercises 7 and 8 in the chapter test for Chapter 5.

Be sure to PCLEAR eight pages before running the program. Also type CLEAR 100 to clear some string space.

The Ticking Clock

100 'LET'S HAVE A CLEAN SCREEN
110 FOR P = 1 TO 8
120 PMODE 0,P: PCLS
130 NEXT P
200 'SET VARIABLES FOR CLOCK HAND
210 A$(1) = "U40": A$(2) = "E35"
220 A$(3) = "R50": A$(4) = "F35"
230 A$(5) = "D40": A$(6) = "G35"
240 A$(7) = "L50": A$(8) = "H35"
300 'DRAW CLOCK HAND
310 FOR P = 1 TO 8
320 PMODE 0,P
330 CIRCLE(128,96),60,,.8
340 DRAW"BM127,96"+A$(P)
350 NEXT P
400 'DISPLAY AND PLAY
410 FOR P = 1 TO 8
420 PMODE 0,P
430 SCREEN 1,1
440 PLAY"L255;V25;D"
450 FOR W = 1 TO 80: NEXT W
460 NEXT P
470 GOTO 410

The sound can be changed by playing a different note at line 440. You might want to vary the note played during the revolution of the clock hand by changing line 440 to something like
OTHER NOISES

440  PLAY“L255;V25;”+STR$(P)

That would give a changing numerical value for the note (1 through 8). How could you change the STR$ parameter to get other notes? See if you can give the answers in the following exercise.

Exercise 7-2

Look at the change just suggested for note values in line 440. Then fill in the parentheses to give the following numerical notes:

a. note range 2 through 9: STR$ ( ____________ )
b. note range 3 through 10: STR$ ( ____________ )
c. note range 4 through 11: STR$ ( ____________ )
d. note range 5 through 12: STR$ ( ____________ )

(Answers are at the end of the chapter.)

You may recall Robbie, the robot, from exercises 9 and 10 of the chapter test at the end of Chapter 5. Robbie's eyes moved from left to center to right. You can add some eyeball “clicking” with the following version of the program.

Robbie with the Roving Eyes

100 'CLEAN 3 SCREENS
110 FOR P = 1 TO 3
120 PMODE 0,P; PCLS
130 NEXT P
200 'DRAW ROBBIE
210 LINE(63,32) - (191,159),PSET,B
220 LINE(89,51) - (115,77),PSET,B
230 LINE(139,51) - (165,77),PSET,B
240 LINE(102,118) - (152,128),PSET,BF
250 PCOPY 1 TO 2: PCOPY 1 TO 3 ← put Robbie on all three pages
300 'DRAW EYEBALLS
310 FOR P = 1 TO 3
320 PMODE 0,P
330 LINE(80+9*P,51) - (88+9*P,77),PSET,BF
340 LINE(130+9*P,51) - (138+9*P,77),PSET,BF
350 NEXT P
400 'FLICK PAGES AND CLICK EYEBALLS
410 FOR P = 1 TO 3
420 PMODE 0,P
Now, as Robbie's eyes move left to right, a very short note is played. You can see how the STR$ function enables you to use paging in order to change the numerical value for the note.

![Eyes left STR$(2) note, C#](image1)

![Eyes center STR$(4) note, E](image2)

![Eyes right STR$(6) note, F](image3)

**Figure 7-1.** Robbie's roving eyes.

After you have had enough of Robbie's roving eyes, return to the three-act program of Chapter 5. That one had the oscillating, rotating, and bouncing balls. It consisted of ACT 1, ACT 2, and ACT 3.

You can add sounds to the acts in various ways. Try these variations to the ACT 3 program. The major changes come in the last section of the program, but there are two earlier lines to add.

Add

```
160 LINE(30,40) - (50,40),PSET
170 LINE(30,160) - (50,160),PSET
```

Revise section 700-760 and add new lines as follows:

```
700 'DISPLAY PAGES WITH SOUND
710 FOR P = 1 TO 4
720 PMODE 0,P
730 SCREEN 1,0
735 PLAY"O2;L255;V25;D"
740 FOR W = 1 TO 50: NEXT W
750 NEXT P
```

"add limits to bouncing ball"
On each movement of the rotating and oscillating balls, a quick D note in octave 2 is sounded. As the bouncing ball hits the limits of the top and bottom of its travel, a quick C note in octave 5 is sounded. Be sure that you use the letter O, rather than the numeral zero, in lines 735, 760, 800 and 830. You can vary the sounds to suit yourself. The values used in our version of the program are merely suggestions.

Here is another variation that you might try:

Change these lines:

735  PLAY"02;L255;V25;"+STR$(P)
760  PLAY"05;L255;V25;"+STR$(P)
800  PLAY"02;L255;V25;"+STR$(P)
830  PLAY"05;L255;V25;"+STR$(P)

With a little imagination, you can probably develop more interesting sounds than we have provided. Feel free to experiment, experiment, and experiment.

Return to exercise 5-5 in Chapter 5. You wrote this program to roll a small circle around a larger circle.

All that you need to do is to alter the section starting at line 400. We've added some sound in the following section of the program. As the small circle rotates around the larger one, rising notes are played. After 13 rotations, notes in the next higher octave are played. After five octaves, the program stops.

Here are the necessary changes to the program from exercise 5-5:

405  FOR X = 1 TO 5 ←to change octave
170  SOUND WITH PLAY

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>406</td>
<td>Z = 5</td>
<td>for first note</td>
</tr>
<tr>
<td>407</td>
<td>FOR R = 1 TO 13</td>
<td>for 13 rotations</td>
</tr>
<tr>
<td>410</td>
<td>FOR P = 1 TO 8</td>
<td>from exercise 29</td>
</tr>
<tr>
<td>420</td>
<td>PMODE 0,P</td>
<td></td>
</tr>
<tr>
<td>430</td>
<td>SCREEN 1,0</td>
<td></td>
</tr>
<tr>
<td>435</td>
<td>PLAY&quot;O&quot;+STR$(X)+&quot;;L&quot;+STR$(Z)+&quot;;V25;&quot;+STR$(P+4)</td>
<td>add sound</td>
</tr>
<tr>
<td>440</td>
<td>NEXT P</td>
<td></td>
</tr>
<tr>
<td>450</td>
<td>Z = INT(Z*1.4)</td>
<td>raise note</td>
</tr>
<tr>
<td>460</td>
<td>NEXT R</td>
<td>next rotation</td>
</tr>
<tr>
<td>470</td>
<td>NEXT X</td>
<td></td>
</tr>
</tbody>
</table>

Now it’s your turn to create some sounds to go with graphics. We’ll provide the program that draws the graphics. Then you add the sound.

The program draws a long red bar. It then draws a border around the bar in yellow. In addition, the red bar is sectioned with yellow vertical lines.

In your part of the program, you are to color each section blue from left to right, play a sound, then color the current section red (as it was originally).

Exercise 7-3

Fill in the blanks in the program as previously described:

100  'SET MODE AND SCREEN
110  PMODE 3,1
120  PCLS
130  SCREEN 1,0
200  'DRAW RED COLOR BAR
210  LINE(10,10) - (245,18),PSET,BF

300  'DRAW BORDER AND SECTIONS
310  DRAW“C2;BM10,10;R235,D8L235U8;BM20,10”
320  FOR X=1 TO 14
330  DRAW“C2;D8BR8U8BR8”
340  NEXT X

---
red bar
How would you like to have a musical instrument driven by joysticks rather than keys, strings, or reeds? If you have the Radio Shack joysticks, you can create such an instrument with the aid of the color computer.

You can use the horizontal control of the left joystick to play notes 1 through 12 and use the vertical control to vary the octave from 1 through 5. The horizontal control of the right joystick can vary the note length, and the vertical control can vary the volume. The statements for the controllers could be:

\[
\begin{align*}
LH &= \text{JOYSTK(0)} & \text{for notes 1-12} \\
LV &= \text{JOYSTK(1)} & \text{for notes 1-5} \\
RH &= \text{JOYSTK(2)} & \text{for note lengths 1-255} \\
RV &= \text{JOYSTK(3)} & \text{for volumes 1-31}
\end{align*}
\]

Remembering that the JOYSTK values range from 0 through 63, you will have to make some compensation for the readings and the directions that you move the joysticks. We suggest the following:

400 'PAINT AND PLAY
410 FOR X = _________ TO _________
420   PAINT _____________
430   Y = _________ : Z = _________
440   PLAY _____________
450   PAINT _____________
460 NEXT X
470 GOTO 410

(Answers are at the end of the chapter.)
For notes: low (1) → high (12)

**JOYSTK(0) reads: left (0) → right (63)**

The equation: \( N = \text{INT}(LH / 5.7) + 1 \)
gives values from 1 through 12 for the horizontal control of the left joystick with left for low notes and right for high notes.

For octaves: low (1) → high (5)

**JOYSTK(1) reads: up (0) → down (63)**

The equation: \( O = 5 - \text{INT}(LV / 5) \)
gives values of 1 through 5 for the vertical control of the left joystick with down giving a low octave and up giving a high octave.

For note lengths: short (255) → long (1)

**JOYSTK(2) reads: left (0) → right (63)**

The equation: \( L = 255 - RH \times 4 \)
gives values of 255 through 3 for the horizontal control of the right joystick with short notes on the left and long notes on the right.
For volumes: low (1) ——— high (31)

JOYSTK(3) reads: up (0) ——— down (63)

The equation: \( V = 31 - \text{INT}(RV/2.1) \)
gives values from 1-31 for the vertical control of the right joystick with the up position giving high volume and the down position giving low volume.

After reading the joysticks for octave, note, note length, and volume, you can use the results in a PLAY command such as:

\[
\text{PLAY}"O"+\text{STR}$(O)+";L"+\text{STR}$(L)+"; V"+\text{STR}$(V)+";"+\text{STR}$(N)
\]

In addition, you can add a color graphics bar to show the note, the note length, and the octave. We chose to do it in this way:

Our program looks like this:

JOYSTICK MUSIC
100 'SET MODE AND SCREEN
110 PMODE 1,1
120 PCLS
130 SCREEN 1,0
200 'READ JOYSTICKS
210 LH = JOYSTK(0): LV = JOYSTK(1)
220 RH = JOYSTK(2): RV = JOYSTK(3)
300 'ASSIGN VARIABLES
310 N = INT(LH/5.7)+1: O = 5-INT(LV/15)
320 L = INT((255-RH*4)/3): V = 31-INT(RV/2.1)
330 X = 30*O: Y = (15-N)*12: Z = INT(RH/2.5)
400 'DRAW AND PLAY
410 LINE(X,Y) - (X+Z, Y+10),PSET,BF
420 PLAY"O"+STR$(O)+";L"+STR$(L)+";V"
   +STR$(V)+";"+STR$(N)
430 GOTO 210

Enter the program and play. Adjust the volume and note length with your right hand. Play the notes and vary the octave with your left hand. If you are right-handed, you may wish to switch the joysticks.

May this 'stick music bring you much joy. Have fun before going on to the chapter summary and test.

Summary

In this chapter you learned how to use the SOUND and PLAY statements with all their options to produce a variety of noises and tones. You learned

• that the SOUND statement can be used to produce 255 notes with selectable durations:

```
SOUND N,D
```

<table>
<thead>
<tr>
<th>Note—any value from 1-255; low numbers give low notes, high numbers give high notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration—any value from 1-255; low numbers give short durations; high numbers give long durations</td>
</tr>
</tbody>
</table>

• that a musical scale can be played with the SOUND statement with the following values for notes:

```
N = 89, 108, 125, 133, 147, 159, 170, 176
```

• that the PLAY function can produce a much wider variety of sounds and has more options than the SOUND statement.
PLAY "sound string"

• that PLAY options are
  note, A through G or 1 through 12
  octave, Ox where x = 1 through 5
  note length, Lx where x = 1 to 255
  tempo, Tx where x = 1 through 255
  volume, Vx where x = 1 through 31
  pause length, Px where x = 1 through 255
  substring, XA$ a substring (A$) of PLAY commands

• that the PLAY function can cover five octaves including sharps and flats

You also learned
• to add some graphics to display the notes being played
• to use all the options of the PLAY functions
• to add sound to existing programs with the PLAY function
• to select PLAY function options by positioning the joysticks

Chapter Test

1. A program has just played a sound using the SOUND function:

   SOUND 40,70

Compare the parameters of the following SOUND statements and tell whether the new sound would be higher or lower and shorter or longer than the one in the program:

a. SOUND 20,90  ________ and  ________
b. SOUND 20,60  ________ and  ________
c. SOUND 50,80  ________ and  ________

2. a. The shortest acceptable value for duration in the SOUND statement is ______________________ ; the longest is ______________________

b. The lowest note value acceptable for the SOUND statement is ______________________ ; the highest is ______________________
3. The following program is a variation of the SOUND program that played the scale and displayed the notes on a staff. This program plays random notes and draws the corresponding note on the scale as it is played. Fill in the missing blanks in the program:

100 'SET SCREEN AND DRAW STAFF
110 PMODE 1,1
120 PCLS
130 SCREEN 1,0
140 FOR Y = 56 TO 88 STEP 8
150 LINE(12,Y) - (240,Y),PSET
160 NEXT Y
170 LINE(12,96) - (40,96),PSET
180 _____________________________________________} copy staff to pages 3 and 4
190 ____________________________________________
200 'GET RANDOM NOTE AND DURATION
210 D = RND(3)+2
220 N = RND(8)
230 'PICK X,Y; PLAY AND DRAW ON PAGE 1
240 ON N GOSUB 1001,1002,1003,1004,1005,1006,1007,1008
250 PMODE 1,1: CIRCLE (X,Y),5
260 SOUND N,D
270 'BLANK OUT NOTE WITH PAGES 3 & 4
280 PMODE __________ , ______________
290 PCOPY _____ TO _____ ; PCOPY _____ TO __________
300 GOTO _____________________________
310 'SELECT NOTE AND GRAPH PARAMETERS
320 N=89: X=26: Y=96: RETURN
330 _______________ ______________
340 _______________ ______________
350 _______________ ______________
360 _______________ ______________
370 _______________ ______________
380 _______________ ______________
390 _______________ ______________
400 _______________ ______________
410 _______________ ______________
420 _______________ ______________
430 _______________ ______________

Hint: Make notes move right and up as the note gets higher.
4. Write a program similar to that of exercise 3, but display piano keys being played instead of notes on a staff. Send your answer to

_Dymax Gazette_
P.O. Box 310
Menlo Park, CA 94025

5. Write a program to play the following notes using the PLAY function.

6. Name that tune (in exercise 5):

7. Write a program to draw a red circle and divide it into eight sectors by red lines.

Paint one section yellow, sound a note, then repaint the sector green. Do this for each sector going clockwise. Raise the note for each sector.
8. Combine portions of the programs in exercises 5 and 7 in a program that will provide a color wheel with music.

9. Write a program using the left joystick to select the octave with its up and down motion and the note with its left to right motion and PLAY the note if the controller button is pressed.

10. Play some fancy music with your program of exercise 9.

Answers to Exercises Within the Chapter

Exercise 7-1

510  PCLS
610  FOR Y = 56 TO 88 STEP 8
620   LINE(12,Y) - (240,Y),PSET
630   NEXT Y
640   LINE(12,96) - (40,96),PSET
710   Y=68
720   FOR X = 220 TO 24 STEP-28
ANSWERS

730 READ T
740 CIRCLE(X,Y),5,2
750 SOUND T,3
760 Y = Y+4
770 NEXT X
810 DATA 176,170,159,147,133,125,108,89

Exercise 7-2
a. STR$(P+1)
b. STR$(P+2)
c. STR$(P+3)
d. STR$(P+4)

Exercise 7-3
410 FOR X = 14 TO 240 STEP 8
420 PAINT(X,12),3,2
430 Y = INT(X/60): Z = Y*3
440 PLAY"O"+STR$(Y+1)+";L55;"+STR$(Z+1)
450 PAINT(X,12),4,2

Answers to Odd-Numbered Exercises in Chapter Test

1. a. lower and longer
   b. lower and shorter
   c. higher and longer
3. 180 PCOPY 1 TO 3
   190 PCOPY 2 TO 4
410 PMODE 1,3
420 PCOPY 3 TO 1: PCOPY 4 TO 2
430 GOTO 210
1002 N=108: X=56: Y=92: RETURN
1003 N=125: X=86: Y=88: RETURN
1004 N=133: X=116: Y=84: RETURN
1005 N=147: X=146: Y=80: RETURN
1006 N=159: X=176: Y=76: RETURN
1007 N=170: X=206: Y=72: RETURN
1008 N=176: X=236: Y=68: RETURN
5. Here's how we did it. Yours may be different.
110 PLAY"T3L2GL4GGA-B-L2GL4GGA-B-L2GL4GEFGL2A-L4A-"
7. 110 PMODE 3, 1
120 PCLS
130 SCREEN 1, 0
140 CIRCLE (128, 96), 60, , .8
150 LINE (128, 50) – (128, 142), PSET
160 LINE (84, 62) – (174, 130), PSET
170 LINE (68, 96) – (188, 96), PSET
180 LINE (84, 130) – (174, 62), PSET
190 PAINT (130, 55), 2, 4
200 PLAY"T6;02,C": PAINT (130, 55), 1, 4
210 PAINT (160, 75), 2, 4
220 PLAY "D":PAINT (160, 75), 1, 4
230 PAINT (160, 100), 2, 4
240 PLAY"E":PAINT (160, 100), 1, 4
250 PAINT (130, 120), 2, 4
260 PLAY "F":PAINT (130, 120), 1, 4
270 PAINT (125, 120), 2, 4
280 PLAY"G":PAINT (125, 120), 1, 4
290 PAINT (90, 100), 2, 4
300 PLAY"A":PAINT (90, 100), 1, 4
310 PAINT (95, 75), 2, 4
320 PLAY"B":PAINT (95, 75), 1, 4
330 PAINT (100, 60), 2, 4
340 PLAY"O3;C":PAINT (100, 60), 1, 4
350 GOTO 190

9. 100 'DEFINE JOYSTICKS
110 N = JOYSTICK(0)
120 O = JOYSTICK(1)
130 NN = INT(N/6)+1
140 OO = INT(O/13)+1
200 ' PEEK TO SEE IF PLAY
210 P = PEEK (65280)
220 IF P=254 OR P=126 THEN PLAY"O"+STR$(OO)+"N"+STR$(NN)
230 GOTO 110
There will be times when you will want to display some text on the screen along with graphics. The TRS-80 Color Computer operates in either the text mode or in one of the graphics modes but not both at the same time. To overcome this fact, we will investigate drawing alphabetic characters in this chapter.

You will also encounter situations where you will want to determine how long it takes to complete some event. TRS-80 Extended Color BASIC has a timer that may be used for this purpose. Part of this chapter is devoted to learning to use the timer.

In this chapter you will learn
- to change back and forth from graphics and text modes
- to draw alphabetic characters in the graphics mode
- to place alphabetic characters on the screen where desired
- to create and use alphabetic characters that are stored in an array
- to time a portion of a program as it is executed
- to use a timer in a program to time your reactions
- to use the timer to calculate a score in game programs

---

**SWITCHING FROM GRAPHICS TO TEXT MODE**

As you know, the text mode is used as you enter a program from the keyboard. The graphics are not displayed until a SCREEN statement is executed by the computer. Enter the following program.

```
100 PMODE 4, 1
110 PCLS
```
RUN the program; nothing seems to happen. The screen shows

```
OK
100 PMODE 4, 1
110 PCLS
120 CIRCLE (128, 96), 30
200 GOTO 200
RUN
```

The text remains on the screen. The circle is not displayed. When you are convinced of this, press the break key.

```
200 GOTO 200
RUN
BREAK IN 200
OK
```

You can see by the message that the program did reach line 200. Therefore, the circle must have been drawn. You need the SCREEN statement to display the circle. Add it at line 130.

```
BREAK in 200
OK
130 SCREEN 1, 0
```

Now RUN the program again.

The circle is displayed.
Now suppose that you want to label your drawing CIRCLE. You can return to the text mode and display the label. Add lines 140 to 160 to your program.

```
100 PMODE 4, 1
110 PCLS
120 CIRCLE (128, 96), 30
130 SCREEN 1, 0
140 FOR W=1 TO 500:NEXT W ← delay to view
150 CLS ← clear text screen
160 PRINT @365, "CIRCLE" ← print label
200 GOTO 200
```

RUN this version of the program and you see

```
first
then
```

The next step would be to alternate the text and graphics displays. Do this by adding

```
170 FOR W=1 TO 500:NEXT W
180 SCREEN 1, 0 ← graphics
190 FOR W=1 TO 500:NEXT W
200 SCREEN 0, 0 ← text
210 GOTO 170
```

When this version is run, the label and the circle are alternately displayed. You might want to change the values in the time delays in lines 170 and 190.

This method does allow mixing text and graphics, but not at the same time. In some cases this may be satisfactory, but in others you may want a different alternative. Why not create your own alphabetic characters with graphics?
The graphics DRAW statement can be used to create readable text in the graphics mode. In the previous program, we wanted to label the circle that we had drawn. To do this in the graphics mode, we might draw the letters as follows:

```
CIRCLE
```

Think of each letter in a 12-by-12 grid. The letter will actually fill an 8-by-8 grid leaving the extra rows and columns for spacing.

To be consistent, we will start and end the drawing of each letter in the lower left corner. The complete word will be drawn as in Figure 8-1.

```
CIRCLE
```

Since the circle has its center at 128, 96 and has a radius of 30, we would like to center the label below the point 128, 96. We might choose 95, 148 as a starting point of the first letter.
Replacing some of the lines of the previous program, we might write

```
100 PMODE 4, 1
110 PCLS
120 CIRCLE (128, 96), 30
130 SCREEN 1, 0
140 ' DRAW LETTER C
150 DRAW"BM95, 148;U8R8BD8L8"
160 ' SKIP 12 TO RIGHT AND DRAW I
170 DRAW"BM+12,0;BU8R8BL4D8BR4L8"
180 ' DRAW R
190 DRAW"BM+12,0;U8R8D4L8BR4F4BL8"
200 ' DRAW C
210 DRAW"BM+12,0;U8R8BD8L8"
220 ' DRAW L
230 DRAW"BM+12,0;U8BR8BD8L8"
240 ' DRAW E
250 DRAW"BM+12,0;U8R8BD4L8BR8BD4L8"
260 GOTO 260
```

By starting and ending each letter in the lower left corner, we can keep track of where we are at all times. Then a relative move 12 spaces to the right ("BM+12,0") will always position the DRAW statement correctly for the next letter.

**Exercise 8-1**

Using the previous program, replace the section that draws the word CIRCLE with a section that will draw the word ROUND. Draw the letters first on the grid.
CREATING AN ALPHABET ARRAY

As you can see from exercise 8-1, drawing characters is time-consuming. It would be convenient to have a subroutine, which could be saved on tape, to draw any letter of the alphabet. You could use an array in the subroutine to assign DRAW commands. Table 8-1 shows such a subroutine.

Using the subroutine in the previous program to draw and label a circle would now become

80 CLEAR 500 ← clear some string space
90 DIM L$(28) ← dimension the array
100 PMODE 4, 1
110 PCLS
120 CIRCLE (128, 96), 30
130 SCREEN 1, 0
CREATING AN ALPHABET ARRAY

TABLE 8-1. TEXT CHARACTERS FROM GRAPHICS

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>' SUBROUTINE FOR DRAWING LETTERS</td>
<td></td>
</tr>
<tr>
<td>10010</td>
<td>L$(1) = &quot;U8R8D4L8BR8D4BL8&quot;</td>
<td>← A</td>
</tr>
<tr>
<td>10020</td>
<td>L$(2) = &quot;U8R6F2D2L8BR8D2G2L6&quot;</td>
<td>← B</td>
</tr>
<tr>
<td>10030</td>
<td>L$(3) = &quot;U8R8BD8L8&quot;</td>
<td>← C</td>
</tr>
<tr>
<td>10040</td>
<td>L$(4) = &quot;U8R6F2D4G2L6&quot;</td>
<td></td>
</tr>
<tr>
<td>10050</td>
<td>L$(5) = &quot;U8R8BD4L8BR8BD4L8&quot;</td>
<td></td>
</tr>
<tr>
<td>10060</td>
<td>L$(6) = &quot;U8R8BD4BL4L4BD4&quot; etc.</td>
<td></td>
</tr>
<tr>
<td>10070</td>
<td>L$(7) = &quot;U8R8BD4L4BR4D4L8&quot;</td>
<td></td>
</tr>
<tr>
<td>10080</td>
<td>L$(8) = &quot;U8R8BD8BU4L8BD4&quot;</td>
<td></td>
</tr>
<tr>
<td>10090</td>
<td>L$(9) = &quot;BU8R8BL4D8BR4L8&quot;</td>
<td></td>
</tr>
<tr>
<td>10100</td>
<td>L$(10) = &quot;U4BU4BR8D8L8&quot;</td>
<td></td>
</tr>
<tr>
<td>10110</td>
<td>L$(11) = &quot;U8BR8G4L4BR4F4BL8&quot;</td>
<td></td>
</tr>
<tr>
<td>10120</td>
<td>L$(12) = &quot;U8BR8BD8L8&quot;</td>
<td></td>
</tr>
<tr>
<td>10130</td>
<td>L$(13) = &quot;U8F4ED8BL8&quot;</td>
<td></td>
</tr>
<tr>
<td>10140</td>
<td>L$(14) = &quot;U8F8U8BG8&quot;</td>
<td></td>
</tr>
<tr>
<td>10150</td>
<td>L$(15) = &quot;U8R8D8L8&quot;</td>
<td></td>
</tr>
<tr>
<td>10160</td>
<td>L$(16) = &quot;U8R8D4L8BD4&quot;</td>
<td></td>
</tr>
<tr>
<td>10170</td>
<td>L$(17) = &quot;U8R8D8H4BF4L8&quot;</td>
<td></td>
</tr>
<tr>
<td>10180</td>
<td>L$(18) = &quot;U8R8D4L8BR4F4BL8&quot;</td>
<td></td>
</tr>
<tr>
<td>10190</td>
<td>L$(19) = &quot;BU4U4R8BD4L8BR8D4L8&quot;</td>
<td></td>
</tr>
<tr>
<td>10200</td>
<td>L$(20) = &quot;BU8R8BL4D8B4L&quot;</td>
<td></td>
</tr>
<tr>
<td>10210</td>
<td>L$(21) = &quot;BU8R8D8L8&quot;</td>
<td></td>
</tr>
<tr>
<td>10220</td>
<td>L$(22) = &quot;BU8D4F4E4U4BG8&quot;</td>
<td></td>
</tr>
<tr>
<td>10230</td>
<td>L$(23) = &quot;U8BR8D8H4G4&quot;</td>
<td></td>
</tr>
<tr>
<td>10240</td>
<td>L$(24) = &quot;BU8F8BU8BG8&quot; ← X</td>
<td></td>
</tr>
<tr>
<td>10250</td>
<td>L$(25) = &quot;BU8F4E4BE4D4BL4&quot; ← Y</td>
<td></td>
</tr>
<tr>
<td>10260</td>
<td>L$(26) = &quot;BU8R8G8R8BL8&quot; ← Z</td>
<td></td>
</tr>
<tr>
<td>10270</td>
<td>RETURN</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>140</td>
<td>GOSUB 10000</td>
<td>← set draw variables</td>
</tr>
<tr>
<td>150</td>
<td>DRAW&quot;BM95, 148&quot;+L$(3)</td>
<td>← draw the C</td>
</tr>
<tr>
<td>160</td>
<td>DRAW&quot;BM+12,0&quot;+L$(9)</td>
<td>← move-draw the I</td>
</tr>
<tr>
<td>170</td>
<td>DRAW&quot;BM+12,0&quot;+L$(18)</td>
<td>← move-draw the R</td>
</tr>
<tr>
<td>180</td>
<td>DRAW&quot;BM+12,0&quot;+L$(3)</td>
<td>← move-draw the C</td>
</tr>
<tr>
<td>190</td>
<td>DRAW&quot;BM+12,0&quot;+L$(12)</td>
<td>← move-draw the L</td>
</tr>
<tr>
<td>200</td>
<td>DRAW&quot;BM+12,0&quot;+L$(5)</td>
<td>← move-draw the E</td>
</tr>
<tr>
<td>210</td>
<td>GOTO 210</td>
<td></td>
</tr>
</tbody>
</table>

Lines 150–200 could be further condensed to one line as

140  DRAW"BM95, 148"+L$(3)+"BR12"+L$(9)+"BR12"+L$(18)+"BR12"+L$(3)+"BR12"+L$(12)+"BR12"+L$(5)
From the last condensation, you can see that we should have included the movement to the right after drawing each character. For instance

for A

L$(1)="U8R8D4L8BR8D4BR4"

for B

L$(2)="U8R6F2D2L8BR8D2G2L6BR12"

Making similar additions to the other letters, you will get a more useful character set as shown in Table 8-2.

**TABLE 8-2. REVISED TEXT CHARACTERS FROM GRAPHICS**

<table>
<thead>
<tr>
<th>Line</th>
<th>Character Set</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>REVISED CHARACTER SET FOR GRAPHICS</td>
<td>A</td>
</tr>
<tr>
<td>10010</td>
<td>L$(1)=&quot;U8R8D4L8BR8D4BR4&quot;</td>
<td>A</td>
</tr>
<tr>
<td>10020</td>
<td>L$(2)=&quot;U8R6F2D2L8BR8D2G2L6BR12&quot;</td>
<td>B</td>
</tr>
<tr>
<td>10030</td>
<td>L$(3)=&quot;U8R8BD8L8BR12&quot;</td>
<td>C</td>
</tr>
<tr>
<td>10040</td>
<td>L$(4)=&quot;U8R6F2D4G2L6BR12&quot;</td>
<td>D</td>
</tr>
<tr>
<td>10050</td>
<td>L$(5)=&quot;U8R8BD4L8BD4R8BR4&quot;</td>
<td>E</td>
</tr>
<tr>
<td>10060</td>
<td>L$(6)=&quot;U8R8BD4L8BD4BR12&quot;</td>
<td>F</td>
</tr>
<tr>
<td>10070</td>
<td>L$(7)=&quot;U8R8BD4L4BR4D4L8BR12&quot;</td>
<td>G</td>
</tr>
<tr>
<td>10080</td>
<td>L$(8)=&quot;U8R8D8BU4L8BD4BR12&quot;</td>
<td>H</td>
</tr>
<tr>
<td>10090</td>
<td>L$(9)=&quot;BU8R8BL4D8BL4R8BR4&quot;</td>
<td>I</td>
</tr>
<tr>
<td>10100</td>
<td>L$(10)=&quot;U4BU4BR8D8L8BR12&quot;</td>
<td>J</td>
</tr>
<tr>
<td>10110</td>
<td>L$(11)=&quot;U8BR8G4L4BR4F4BR4&quot;</td>
<td>K</td>
</tr>
<tr>
<td>10120</td>
<td>L$(12)=&quot;U8BD8R8BR4&quot;</td>
<td>L</td>
</tr>
<tr>
<td>10130</td>
<td>L$(13)=&quot;U8F4E4D8BR4&quot;</td>
<td>M</td>
</tr>
<tr>
<td>10140</td>
<td>L$(14)=&quot;U8F8U8BD8BR4&quot;</td>
<td>N</td>
</tr>
<tr>
<td>10150</td>
<td>L$(15)=&quot;U8R8D8L8BR12&quot;</td>
<td>O</td>
</tr>
<tr>
<td>10160</td>
<td>L$(16)=&quot;U8R8D4L8BD4BR12&quot;</td>
<td>P</td>
</tr>
<tr>
<td>10170</td>
<td>L$(17)=&quot;U8R8SH4G4R8BR4&quot;</td>
<td>Q</td>
</tr>
<tr>
<td>10180</td>
<td>L$(18)=&quot;U8R8D4L8BR4F4BR4&quot;</td>
<td>R</td>
</tr>
<tr>
<td>10190</td>
<td>L$(19)=&quot;BU4U4R8BD4L8BR8D4L8BR12&quot;</td>
<td>S</td>
</tr>
<tr>
<td>10200</td>
<td>L$(20)=&quot;BU8R8BL4D8BR8&quot;</td>
<td>T</td>
</tr>
<tr>
<td>10210</td>
<td>L$(21)=&quot;U8BR8D8L8BR12&quot;</td>
<td>U</td>
</tr>
<tr>
<td>10220</td>
<td>L$(22)=&quot;BU8D4F4E4U4BD4BR4&quot;</td>
<td>V</td>
</tr>
<tr>
<td>10230</td>
<td>L$(23)=&quot;U8BR8D8H4G4BR12&quot;</td>
<td>W</td>
</tr>
<tr>
<td>10240</td>
<td>L$(24)=&quot;E8BL8F8BR4&quot;</td>
<td>X</td>
</tr>
<tr>
<td>10250</td>
<td>L$(25)=&quot;BU8F4E4BG4D4BR8&quot;</td>
<td>Y</td>
</tr>
<tr>
<td>10260</td>
<td>L$(26)=&quot;BU8R8G8R8BR4&quot;</td>
<td>Z</td>
</tr>
<tr>
<td>10270</td>
<td>L$(27)=&quot;BR12&quot;</td>
<td>to give a space</td>
</tr>
<tr>
<td>10280</td>
<td>RETURN</td>
<td></td>
</tr>
</tbody>
</table>
The revised subroutine is much easier to use in a program. Our circle program now looks like this:

80 CLEAR 500
90 DIM L$(28)
100 PMODE 4, 1
110 PCLS
120 CIRCLE (128, 96), 30
130 SCREEN 1, 0
140 GOSUB 10000
150 DRAW"BM95, 148"+L$(3)+L$(9)+L$(18)+L$(3)+L$(12)+L$(5)
160 GOTO 160

Of course, the subroutine must be used with the program each time. You may enter it from tape (if you saved it), or you may type it in each time you want to use it.

Exercise 8-2

Write a program using PMODE 4, 1 to display the figures and letters that follow. Use the subroutine in Table 8-2 to draw the letters.

Your program:
**USING THE TEXT CHARACTER SUBROUTINE**

When you are using the text character subroutine, you must plan the placement of the text on the screen just as you would a graphics display. The text characters will not automatically wrap around to a new line when the end of a line is reached.

Since each character (with spaces) takes 12 horizontal graphics positions, 21 characters will fit on a line if you start at the extreme left.

- **21 characters per line**
- **256 horizontal positions**

Since each character takes 12 positions, 21 characters will fit on a line if you start at the extreme left.

- **12 positions**
- **24 characters**
- **16 positions**
- **256 horizontal positions**
- **12 characters**
- **4 positions**
- **a part of a character**
- **left over**

Load the subroutine from tape if you saved it. Then enter lines 80–300 of this program. If you did not save the subroutine on tape, enter it now as well as lines 80–300. Run this program to see the complete alphabet displayed:

```basic
10 ' COMPLETE ALPHABET PROGRAM
80 CLEAR 500
90 DIM L$(28)
100 PMODE 4, 1
110 PCLS
120 SCREEN 1, 0
130 GOSUB 10000
140 DRAW"BM2, 12"
150 FOR X=1 TO 21
160 DRAW L$(X)
170 NEXT X
200 DRAW"BM2, 24"
210 FOR X=22 TO 27
220 DRAW L$(X)
230 NEXT X
300 GOTO 300
```

1st line of characters

2nd line of characters
USING THE TEXT CHARACTER SUBROUTINE

10000 ' REVISED CHARACTER SET FOR GRAPHICS
10010 .
10280 RETURN

When the program is RUN, you should see this displayed:

```
ABCDEFGH IJKLMNOPQRSTUVWXYZ
```

If you want to put a space between each letter, try one of these changes to lines 140-300:

```
140 DRAW"BM2,12"
150 FOR X=1 TO 9
160  DRAW L$(X)+L$(27)
170  NEXT X
180 DRAW"BM2,24"
190 FOR X=10 TO 18
200  DRAW L$(X)+L$(27)
210  NEXT X
220 DRAW"BM2,36"
230 FOR X=19 TO 26
240  DRAW L$(X)+L$(27)
250  NEXT X
300 GOTO 300
```

When you enter and RUN either of these programs, the display shows

```
ABCDEFGHI
JKL MNOPQR
STUVWXYZ
```

Does the color computer DRAW as fast as it interprets and executes other BASIC commands? Extended Color BASIC has a TIMER statement, which can be used to determine how long it takes to complete such things.
The moment that you power up your TRS-80 Color Computer, a clock is turned on. It starts at 0 and counts up to 65535. It then recycles to 0 and repeats the process over and over. The counter is stopped during cassette and printer operations but otherwise keeps ticking away.

The TIMER function uses the clock to measure time in units that are approximately 1/60 of a second.

- If you want to see the time, you use the statement

  PRINT TIMER

- If you want to set the timer to 0, you use the statement

  TIMER = 0

- The timer may be set to any number that you wish (from 0 through 65535) by

  TIMER = n

  n may be any integer 0-65535

To demonstrate the TIMER function, let's time how long it takes to draw the complete alphabet. We'll modify the complete alphabet program that you used earlier. Don't forget to load in the subroutine before entering the program.

10 'TIMED ALPHABET PROGRAM #1
80 CLEAR 500
90 DIM L$(28)
100 PMODE 4, 1
110 PCLS
120 SCREEN 1, 0
130 GOSUB 10000
135 TIMER = 0
140 DRAW"BM2,12"
150 FOR X=1 TO 21
160 DRAW L$(X)
170 NEXT X
200 DRAW"BM2,24"
210 FOR X=22 TO 27
This is what we saw when we ran the program the first time:

30

that's 30/60 seconds or approximately ½-second

We made several repeat runs and found that the timer reading varied from 28 to 30.

You can also save the time that passed for the occurrence of portions of an event. You could then display it later when all the timing was finished. For instance, you could find out how long it takes to draw the first line of the alphabetic characters as well as the complete alphabet by adding these lines:

change 10 'TIMED ALPHABET PROGRAM #2
add 180 L1 = TIMER
change 240 LS = TIMER
add 250 CLS;PRINT"FIRST LINE =";L1
add 260 PRINT"SECOND LINE =";LS-L1
add 270 PRINT"BOTH LINES =";LS

The display that we first saw using these revisions was

FIRST LINE = 23
SECOND LINE = 7
BOTH LINES = 30
We ran this revision several times and the timer varied quite a bit:

<table>
<thead>
<tr>
<th></th>
<th>23</th>
<th>23</th>
<th>23</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>first line</td>
<td>23</td>
<td>23</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>second line</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>both lines</td>
<td>29</td>
<td>30</td>
<td>28</td>
<td>30</td>
</tr>
</tbody>
</table>

LEAVE THIS PROGRAM IN THE COMPUTER. We'll modify it once more shortly.

It doesn't make much sense to read time in 1/60 seconds. Let's make the computer give us a more meaningful number by dividing the result of the timer by 60. To do this, we can change these lines:

```
250  CLS: PRINT"FIRST LINE =";L1/60;"SECONDS"
260  PRINT"SECOND LINE =";(LS-L1)/60;"SECONDS"
270  PRINT"BOTH LINES =";LS/60;"SECONDS"
```

A run of these revisions gave us

```
FIRST LINE = .366666667 SECONDS
SECOND LINE = .116666667 SECONDS
BOTH LINES = .483333333 SECONDS
```

Again, some variation was seen when the program was repeated several times.

**TIMING MORE INTERESTING EVENTS**

You have been using the letter-drawing subroutine for a short time, but you have been using the alphabet for quite a long time. You know there are 26 letters in the alphabet, and we used 27 subscripted variables (L$(1)-L$(27)) in our subroutine. L$(27) was used for a space. In the game that follows, we'll ignore L$(27).

Let's create a program that will place a letter of the alphabet in the middle of the screen. Then you guess its corresponding number (A=1, B=2, C=3, ..., Z=26). Remember, L$(1) draws an A, L$(2) draws a B, etc.

We'll use the timer to determine how long it takes you to respond. Of course, we'll add on a penalty whose magnitude will depend on how far you miss the correct number of the letter. We'll give you 10 letters before displaying your score.

```
10 ' ALPHABET GAME
100 CLEAR 500
110 DIM L$(28),Q(10)
120 CLS
```
TIMING MORE INTERESTING EVENTS

130 PMODE 4, 1
140 PCLS
150 SCREEN 1, 0
160 GOSUB 10000

200 Q = 0
210 FOR X = 1 TO 10
220 R = RND(26)    \random letter picked
230 DRAW "BM120,100"+L$(R)    \draw the letter
240 TIMER = 0    \start timer
250 DRAW "BM2,12;U8"+L$(27)    \ready for first digit
260 A$(1)=INKEY$: IF A$(1)=" " THEN 260
270 T1=TIMER: TIMER=0    \save time to get 1st digit guess, reset timer
280 DRAW "BM2,12;U8BR12D8"    \ready for 2nd digit
290 A$(2)=INKEY$: IF A$(2)=" " THEN 290
300 T2=TIMER    \save time to get second digit guess
310 N=10*VAL(A$(1))+VAL(A$(2))    \calculate guess
320 E=ABS(R-N)    \calculate error
330 T=INT((T1+T2)/6)    \response in 10ths of seconds
340 Q(X)=T+20*E    \add time and error penalty
350 PCLS    \get ready for new try
360 NEXT X    \new try

400 CLS: PRINT "YOUR SCORES WERE:"
410 FOR X = 1 TO 10
420 Q=Q+Q(X)
430 PRINT "#";X;Q(X)
440 NEXT X
450 PRINT: PRINT "TOTAL SCORE =";Q
500 PRINT: PRINT
510 PRINT "PRESS ANY KEY TO PLAY AGAIN"
520 A$=INKEY$: IF A$=" " THEN 520
530 GOTO 140

Don’t forget to add the alphabet subroutine. Have fun learning your alphabet again. Feel free to modify the game for higher or lower rewards and penalties. The next exercise explains the use of the game a little more and has a scorecard for recording your results.
Exercise 8-3

Run the alphabet game and record your scores. Remember that speed and accuracy are both important. The program uses INKEY$. Two digits must be entered for each letter:

- If A, enter 01
- If B, enter 02
- If C, enter 03
- ...
- If Z, enter 26

A prompt is given in the upper left corner of the screen so that you can tell when each digit is accepted.

Example 1:  \( P = L$(16) \)

\( P \) is 16th letter of the alphabet

ENTRY WOULD BE: 16

Example 2:  \( C = L$(3) \)

\( C \) is the 3rd letter of the alphabet

ENTRY WOULD BE: 03
TIMING MORE INTERESTING EVENTS

ready for 1st digit

ENTER: 0

ready for 2nd digit

ENTER: 3

YOUR RECORD SHEET

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The shooting gallery program in Chapter 6 could be timed to provide a way to assign a score to your results. The flowchart for that program is reproduced in Figure 8-2. The flowchart shows places in the program to provide timing, rewards, and penalties. The timer is only turned on when no missile is in flight. A penalty is given if the missile goes out of range before being hit, and a reward is given when the missile is hit. The score with rewards added and penalties received is purely arbitrary and may be changed if desired.

The timer is checked to stop the game when it exceeds 10000 (approximately 167 seconds). Therefore, you have about three minutes of target practice before your score is displayed.

The following changes were made to the original program.

Additions to line 210: TT=0 ← total time set to zero
                   N=0 ← reward set to zero
                   P=0 ← penalty set to zero
                   S=0 ← total score set to zero
New: 215 TIMER = 0 ← start timing

Addition to line 450: IF MS=0 THEN T=TIMER ← save time if missile in flight

Changed line: 610 TT=TT+T ← add new time to time
New: 620 IF TT<10000 THEN 220 ELSE 920 ← calculate score if time up

Additions to line 750: P=P+50 ← add penalty for miss
                   TIMER=0 ← reset timer
New: 860 N=N+200:PCLS ← add reward for hit, clear screen

New lines 900-950 to calculate the score:
TT=TT+T ← total time
IF TT<10000 THEN 215 ← return if more time
S=N-P-INT(TT/40) ← subtract miss and time penalties from reward

Also clear screen print score, check if player wants to replay, and start over.
Figure 8-2. Flowchart for timed fire stations.
The revised program follows:

```
100 'SET SCREEN
110 PMODE 4, 1
120 PCLS
130 SCREEN 1, 1
200 'SET VARIABLES
210 MS;;:;O: R;;:;126:TI;;:;0: N;;:;O: P;;:;O:S;;:;O
215 TIMER;;:;O
220 A=RND(2):B=RND(128)+30
230 IF A=1 THEN C=250:D=8
240 IF A=2 THEN C=0:D=-8
300 'ACTION
310 FOR Z;;:;B TO C STEP D
320 'TARGET ON
330 LINE(Z,10)-(Z+15,14),PSET,BF
340 'LOOK FOR SHOTS
350 FOR Q;;:;l TO 3
360 'ERASE GUN
370 COLOR 0, 1
380 LINE(R,180) - (R+5,184),PSET,BF
390 'READ JOYSTICK
400 H=JOYSTK(O): R=4*H
410 'REDRAW GUN
420 COLOR 1, 0
430 LINE(R,180) - (R+5,184),PSET,BF
440 'START OR MOVE MISSILE
450 IF MS=1 THEN T=TIMER:GOTO 710
460 IF PEEK(65280)=254 OR PEEK(65280)=126
    THEN X=R+2:Y=176:CIRCLE(X,Y),2:MS=1
470 NEXT Q
500 'ERASE TARGET
510 COLOR 0,1:LINE(Z,10) - (Z+15,14),PSET,BF
520 COLOR 1, 0
530 NEXT Z
600 'NEW TARGET
610 TT=TT+T
620 IF TT<10000 THEN 220 ELSE 920
700 'MOVE MISSILE
710 CIRCLE (X,Y),2,0:Y=Y-12
```
SUMMARY

In this chapter, you learned to create and place text characters on the graphics screen and to use the TIMER statement to time events. You learned

- to control the type of display with

  SCREEN 1,n  to turn on the graphics screen
  (n may be 0 or 1 for color set)

  SCREEN 0,0  to turn on the text screen

- to use the DRAW statement to create alphabetic characters in the graphics mode such as

  DRAW"BM95,148;U8R8BD8L8"  for the letter C

- to string several letters together on the graphics screen such as

  DRAW"BM95,148;U8R8BD8L8"  for a C
  DRAW"BM+12,0;BU8R8BL4D8BR4L8"  move 12 right, draw I
• to write a subroutine containing all the letters of the alphabet using an array, L$

• to access letters in the alphabet subroutine and place them on the screen, such as

```
DRAW"BM95,148"+L$(3)+L$(9)+L$(18)+L$(3)+L$(12)+L$(5)
```

C I R C L E

• all about the TIMER function:

- TIMER=0 sets the timer to zero
- PRINT TIMER prints the value of timer (0-65535)
- TIMER=n set timer to value of n (0-65535)
- T=TIMER save a given time value in variable T

• to use the TIMER function to record your reaction time in pairing letters with corresponding numbers

• to use the TIMER function in a shooting gallery game to aid in calculating a score

Chapter Test

1. The purpose of the statement SCREEN 1,0 is ________________________________

2. How does the effect of the statement SCREEN 0,0 differ from that of SCREEN 1,0?

   ________________________________

3. In the subroutine that you used to create alphabetic characters in this chapter, the array L$ was used. Give the letters that correspond to the following elements of L$:

   a. L$(8) = the letter ______________________

   b. L$(22) = the letter ______________________

   c. L$(15) = the letter ______________________
4. Write a subroutine similar to L$ for lower case letters: a, b, c, d, etc.

5. Write a program that will put the word SCROLL at the bottom center of the screen, such as

   SCROLL

   SCROLL

6. Add lines to your program of test exercise 5 to make the word SCROLL move smoothly upward and eventually off the screen like this. (*Hint:* use GET, PUT, and paging)
7. a. What line would you change in the program that follows Table 8-2 to scale the letters to half-size? 

b. Show what the revised line would be.

8. Modify the complete alphabet program to change the scale of the letters to one-half and then to three-fourths. RUN the program with each scale and see if we could have used smaller letters.

| Changes: |

9. Since the TIMER function counts at approximately 1/60th of a second, make a clock tell the time. Use S for seconds, M for minutes, and H for hours. We'll give you a start:

```
100 S=0:M=0:H=0
110 CLS
120 TIMER=0
130 IF TIMER>60 THEN TIMER=0:S=S+1
140 IF S=60 THEN S=0:M=M+1
150 IF M=60 THEN M=0:H=H+1
160 IF H=13 THEN H=1
180 GOTO 130
```

Add line 170 to print the time in the upper left corner of the screen as

```
170
```

10. Change line 100 (add more if needed) to the program in test exercise 9 to allow you to set the hours, minutes, and seconds to give the correct time of day.
Answers to Exercises Within the Chapter
(Your answers may vary from ours—try them on the computer.)

Exercise 8-1

150 DRAW“BM95,148;U8R8D4L8BR4F4BL8”
170 DRAW“BM+12,0;U8R8D8L8”
190 DRAW“BM+12,0;U8BR8D8L8”
210 DRAW“BM+12,0;U8F8U8BG8”
230 DRAW“BM+12,0;U8R6F2D4G2L6”

Exercise 8-2

100 CLEAR 500
110 DIM L$(28)
120 PMODE 4, 1
130 PCLS
140 SCREEN 1, 0
150 GOSUB 10000
160 CIRCLE(50, 60), 15
170 DRAW“BM100,65”+L$(3)+L$(9)+L$(18)+L$(3)+L$(12)+L$(5)
180 DRAW“BM35, 105;R30H15G15”
190 DRAW“BM100, 100”+L$(20)+L$(18)+L$(9)+L$(1)+L$(14)
+L$(7)+L$(12)+L$(5)
200 LINE(35, 140) – (65, 125), PSET,B
210 DRAW“BM100,135”+L$(18)+L$(5)+L$(3)+L$(20)+L$(1)
+L$(14)+L$(7)+L$(12)+L$(5)
220 GOTO 220
10000 ' REVISED CHARACTER SET FOR GRAPHICS ← add
10280 RETURN

Exercise 8-3
Record sheets will vary. I hope that yours was excellent.
Answers to Odd-Numbered Exercises in Chapter Test

1. to turn on the graphics screen
   
   SCREEN 1,0 selects the first color set
   SCREEN 1,1 selects the second color set

3. a. \texttt{L$(8) = the letter H}
   
   b. \texttt{L$(22) = the letter V}
   
   c. \texttt{L$(15) = the letter O}

5. Sample (yours may be different)
   
   \begin{verbatim}
   80  CLEAR 500  
   90  DIM L$(28)  
   100 PMODE 4,1:PCLS  
   110 SCREEN 1,0  
   120 GOSUB 10000  
   130 DRAW"BM92,190"+L$(19)+L$(3)+L$(18)+L$(15)+L$(12)  
   +L$(12)  
   140 GOTO 140  
   10000  ' REVISED CHARACTER SET FOR GRAPHICS  
   10280  RETURN  
   \end{verbatim}

7. a. Line 150
   
   b. 150 DRAW"S2;BM95,148"+L$(3)+L$(9)+L$(18)+L$(12)+L$(5)
   
   \[ \text{lscale} = 2/4 \]

9. 150 \texttt{PRINT @0, STR$(H)+" ::": STR$(M)+" ::": STR$(S)
This chapter will introduce you to machine language. It is the "native" language of the computer. All BASIC instructions must be interpreted by the BASIC language ROM interpreter in order for the computer to understand what your BASIC statements mean. If you can write programs in machine language, you can save much execution time and memory space. However, programming in machine language is much more detailed and time-consuming.

A blend of BASIC and machine language programming is possible through the BASIC USR function. It provides a link between a BASIC program and one or more machine language subroutines. Therefore, you can program a large portion of a program in BASIC and do the other parts in machine language. The USR function then acts like a GOSUB statement, except that you go to a machine language subroutine.

In this chapter, you will learn

- to reserve memory for a machine language subroutine
- to define the USR function to be used
- to POKE machine language instructions and data into memory
- to call a machine language program using the USR function
- to return to your BASIC program after the machine language program has been executed
- a little bit about binary and hexadecimal numbers
- to use the video display in order to see the results of machine language subroutines
- to use machine language in order to create graphic characters on the screen
MEMORY USE FOR MACHINE LANGUAGE SUBROUTINES

Care must be used to keep your machine language subroutines in a section of memory that is not used by the computer for other purposes. The 16K RAM Memory Map (p. 204 of the TRS-80 Color Computer manual Going Ahead with Extended Color BASIC) shows how the computer uses its memory. Notice the area:

<table>
<thead>
<tr>
<th>Decimal Address</th>
<th>Contents</th>
<th>Hex Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>13824-16383</td>
<td>Program and Variable Storage</td>
<td>3600-3FFF</td>
</tr>
</tbody>
</table>

Your BASIC program, with its variable storage begins at memory address 13824 and works upwards. For our purposes, the area from 16001-16383 will provide ample storage for the small machine language programs used in this book.

To save that area for your machine language program, use this statement at the beginning of your program when using the USR function:

```
CLEAR 383, 16000
```

This reserves 383 memory locations for your machine language program. A smaller number may be used for short programs.

<table>
<thead>
<tr>
<th>Decimal Address</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>13824-16000</td>
<td>BASIC program and variables</td>
</tr>
<tr>
<td>16001-16383</td>
<td>Reserved for machine language</td>
</tr>
</tbody>
</table>
Other memory locations may be used, but we will stay with this area for this book.

A more thorough discussion of memory use and machine and assembly language programming on the TRS-80 Color Computer will be found in a forthcoming book from Reston Publishing Co., Inc., titled *Programming the TRS-80 Color Computer in 6809 Machine/Assembly Language*.

---

**THE USR FUNCTION**

Machine language programs, to be accessed by the USR function, may be entered from BASIC in three ways.

1. If the program already exists on tape, it may be loaded into memory from a cassette recorder by the CLOADM command.
2. It may also be transferred (downloaded) from another computer by the DLOAD command.
3. It may be POKEd into memory directly by the BASIC program that is going to use it.

We will be using the third method in this chapter.

Your BASIC program will first clear an area of memory for the machine language program with a statement such as

```
CLEAR 25,16000
```

In addition to clearing an area of memory for the machine language subroutine, the USR function must be defined. The DEF USR statement is used to define the entry address of a USR function. This tells the computer where to find the machine language subroutine:

```
DEF USRn = address
```

- *n* is a number 0-9; up to 10 USR functions may be used; 0 is used if *n* is omitted
- the entry address of the machine language subroutine (16001 in our example)

In general, the approach to using a machine language subroutine in this chapter will be to

1. clear memory for machine language subroutine,

```
110 CLEAR 30,16000
```
2. define the USR function:

```basic
120 DEF USR0 = 16001
```

3. POKE in machine language subroutine instructions from BASIC and execute other BASIC statements

4. call the machine language subroutine with the USR function

5. execute the machine language subroutine

6. return from the subroutine to BASIC

7. continue the BASIC program

We have discussed steps 1 and 2 of the preceding list. The values POKEd in by step 3 will vary from program to program. We will return to that soon. Let's take a look at step 4.

In its simplest form, the instruction used to enter (call) a machine language subroutine using the previously defined USR function is

```basic
X = USR0 (0)
```

When this statement is executed, control is passed to the machine language subroutine at the memory address in the USR function definition.

The machine language program is then executed as stated in step 5. When it has been completed, the computer needs some way to return to BASIC. The last machine language subroutine executed must return you to your BASIC program (step 6). A machine language RETURN statement can be used to do this. It will be shown in an example.

Upon return to BASIC, further BASIC statements can be executed (step 7). Frequently you will use the PEEK statement to look for results of your machine language subroutine. You may also call the same subroutine again or call other subroutines (USR0-USR9).

That may seem like a lot to remember, but it will be easy after a few short examples.

Before jumping into how to write a machine language program, let's see how the USR function works. Don't worry about how we arrived at the machine language instructions of the subroutine; just use them. We've listed the steps of the general approach to using a machine language subroutine to the right of the program.

### PROGRAM TO POKE SUBROUTINE, THEN PEEK AT IT

```basic
100 ' SET UP USR
```
THE USR FUNCTION

110 CLEAR 23, 16000
120 DEF USR0 = 16001
200 ' POKE IN SUBROUTINE
210 FOR X = 16001 TO 16023
220 READ A
230 POKE X, A
240 NEXT X
300 ' SUBROUTINE DATA POKEO IN
310 DATA 142, 62, 142, 166, 128, 39, 5, 189
320 DATA 163, 10, 32, 247, 57, 83, 85, 67, 67
330 DATA 69, 83, 83, 33, 13, 0
340 CLS
500 CONTINUE HERE
510 FOR X = 16001 TO 16023
520 PRINT PEEK(X); 
530 NEXT X

Notice that we skipped step 4. We want you to see that the subroutine is actually poked into memory correctly before you run the program. Lines 210-240 POKE the instructions into memory. Lines 510-530 let you check the entries. Study them carefully. Any entry errors may hang up the computer, and you may have to reset and start all over.

Run the program as it is. The screen will clear and then the computer will display the decimal values of the instructions and the data used in the machine language subroutine.

| 142 | 62 | 142 | 166 | 128 | 39 | 5 |
| 189 | 103 | 10 | 32 | 247 | 57 | 83 | 8 |
| 69 | 83 | 83 | 33 | 13 | 0 |

OK

Now that you are satisfied that the machine language subroutine actually exists in memory, add line 410 to the program. This line will execute the subroutine:

410 Y = USR0(0)

Exercise 9-1

Run the program, now that line 410 has been added. Show what you see on the screen.
Memory locations in the TRS-80 Color Computer hold eight pieces of information in numerical form. Unfortunately for us humans, these pieces of information are not decimal digits. Due to the structure of the computer, these pieces are binary digits or bits (binary digits). Therefore, it is essential to learn how to interpret binary numbers in order to communicate directly with the computer. To us, the content of a memory location might look like this:

![Memory Location Diagram](image)

If you are familiar with binary and hexadecimal number systems, feel free to skip the following discussion.

The binary number system has a base of two. That means that each bit (binary digit) has a place value of an increasing power of two as you move from right to left.

**Example:**

<table>
<thead>
<tr>
<th>2^7</th>
<th>2^6</th>
<th>2^5</th>
<th>2^4</th>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

To convert this number to an understandable decimal form, you would evaluate each bit from right to left:

```
No 2^0 = 0*1 = 0
```
MACHINE LANGUAGE FORMAT

No $2^1 = 0*2 = 0$
No $2^2 = 0*2*2 = 0$
One $2^3 = 1*2*2*2 = 8$
One $2^4 = 1*2*2*2*2 = 16$
No $2^5 = 0*2*2*2*2*2 = 0$
One $2^6 = 1*2*2*2*2*2*2 = 64$
No $2^7 = 0*2*2*2*2*2*2*2 = 0$

01010000 binary = 64+16+8 = 88 decimal

You can see that you might convert every machine language instruction and all data back and forth from binary to decimal and decimal to binary.

You probably can't imagine that anyone would want to talk to the computer in this cumbersome format; neither can we.

Long ago, someone noticed that binary numbers could be expressed in the hexadecimal number system in a much shorter way. The hexadecimal system uses a base of 16 ($2^4$). Its symbols are converted to decimal as in Table 9-2.

**TABLE 9-2. HEX DIGITS TO DECIMAL EQUIVALENT**

<table>
<thead>
<tr>
<th>Hexadecimal Symbol</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
</tr>
<tr>
<td>D</td>
<td>13</td>
</tr>
<tr>
<td>E</td>
<td>14</td>
</tr>
<tr>
<td>F</td>
<td>15</td>
</tr>
</tbody>
</table>

- these are the same
- these are different
Now look at the binary-hexadecimal relationships in Table 9-3.

**TABLE 9-3. FOUR-BIT BINARY TO HEXADECIMAL EQUIVALENT**

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
</tr>
</tbody>
</table>

You can see that each hexadecimal digit may be represented as a combination of four binary digits. Therefore, an eight-bit memory number (as used by the computer) can be represented by two hexadecimal digits. In our previous example,

\[
\begin{align*}
0 & 1 & 0 & 1 & \quad & 1 & 0 & 0 & 0 & \quad & \text{binary} \\
5 & \quad & 8 & \quad & \text{hexadecimal}
\end{align*}
\]

In decimal, this is \(5 \times 16 = 80\)

In decimal, this is \(8 \times 1 = 8\)

Therefore \(58\) hexadecimal = \(80 + 8 = 88\) decimal.

We are building up to the fact that most machine language reference books list machine language instructions in hexadecimal format. Quite often it is possible to enter the codes for the machine language instructions in their hexadecimal form. Keep in mind that the computer actually works with binary numbers and BASIC language uses decimal numbers. You must learn to “think” in all three formats. We will ignore the binary format for now. We will use one more shortcut by referring to the word hexadecimal as hex.
HEX-DECIMAL CONVERSIONS

Even though the computer can make the necessary conversions for you, you should learn to convert between the hex and decimal systems yourself.

1. To convert from four-place hex numbers to decimal numbers, convert each hex place to its decimal equivalent:

<table>
<thead>
<tr>
<th>Hex</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>12 (decimal)</td>
</tr>
<tr>
<td>A</td>
<td>10 (decimal)</td>
</tr>
<tr>
<td>2</td>
<td>2 (decimal)</td>
</tr>
<tr>
<td>C</td>
<td>12 (decimal)</td>
</tr>
</tbody>
</table>

\[
3 \cdot 16^3 + 10 \cdot 16^2 + 2 \cdot 16^1 + 12 \cdot 16^0 = 14892 \text{ decimal}
\]

2. To convert decimal values to four-place hex values, divide first by 4096, then 256, and last 16. For example, 14892 decimal changed to hex:

- a. divide by 4096

\[
\begin{array}{c|c}
& 3 \\
4096 & 14892 \\
& 12288 \\
& 2604
\end{array}
\]

- b. divide remainder by 256

\[
\begin{array}{c|c}
10 & \text{2nd hex digit = A} \\
256 & 2604 \\
256 & 256 \\
44 & \text{next use this remainder}
\end{array}
\]

- c. divide remainder by 16

\[
\begin{array}{c|c}
2 & \text{3rd hex digit = 2} \\
16 & 44 \\
16 & 32 \\
12 & \text{4th hex digit = C} \\
& (12 \text{ dec. = C hex})
\end{array}
\]

14892 decimal = 3A2C hex
Exercise 9-2

With these conversions in mind, you can return to the program to POKE subroutine, then PEEK at it and interpret the instructions and data used in the subroutine. Fill in the table of hex values. Either calculate the hex values by hand or let the computer do it for you with this program:

```
100    CLS
110    INPUT D  'input the decimal value
120    PRINT D;HEX$(D)  'converts D to hex
130    GOTO 110
```

<table>
<thead>
<tr>
<th>Memory Location</th>
<th>Decimal Value</th>
<th>Hex Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>16001</td>
<td>142</td>
<td></td>
</tr>
<tr>
<td>16002</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>16003</td>
<td>142</td>
<td></td>
</tr>
<tr>
<td>16004</td>
<td>166</td>
<td></td>
</tr>
<tr>
<td>16005</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>16006</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>16007</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>16008</td>
<td>189</td>
<td></td>
</tr>
<tr>
<td>16009</td>
<td>163</td>
<td></td>
</tr>
<tr>
<td>16010</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>16011</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>16012</td>
<td>247</td>
<td></td>
</tr>
<tr>
<td>16013</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>16014</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>16015</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>16016</td>
<td>67</td>
<td></td>
</tr>
<tr>
<td>16017</td>
<td>67</td>
<td></td>
</tr>
<tr>
<td>16018</td>
<td>69</td>
<td></td>
</tr>
</tbody>
</table>
HEX-DECIMAL CONVERSIONS

The computer can also convert hexadecimal values to decimal equivalents by using the characters &H before the hex value. However, this technique cannot be used directly in an INPUT or DATA statement. By using the VAL statement (line 130 in the following program), a string &Hxx is formed. The xx is the two-digit hex input and is catenated with the &H. The variable N is assigned to the decimal equivalent of the hex value INPUT as the string N$.

```
100 CLS
110 FOR X=1 TO 23
120 INPUT N$
130  N = VAL("&H"+N$) ←decimal equivalent
140 PRINT 16000+X;N$;N
150 NEXT X
```

You can check the results of exercise 9-2 by running this program. The INPUT prompt (?) appears at the left of the screen. Input your answers in order. The memory location, your hex input, and its decimal equivalent appear from right to left.

```
?8E ←input hex value 8E
16001 8E 142 ←next input
```

The machine language subroutine consists of instructions (called operation codes or op codes), data used with the instructions (called operands), and data used by the subroutine. If we divide the subroutine into functional blocks, it becomes more understandable.

We have also included a column for labels and a column for assembly...
language mnemonics. As you can see in Table 9-4, the assembly language mnemonics give a clearer indication of the function of the instructions than do the machine language op codes.

<table>
<thead>
<tr>
<th>Instruct. Number</th>
<th>Label</th>
<th>Machine Op Code</th>
<th>Assembler Mnemonic</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>8E 3E 8E</td>
<td>LDX #DATA</td>
<td>Load X register with 1st address of data</td>
</tr>
<tr>
<td>2</td>
<td>LOOP</td>
<td>A6 80</td>
<td>LDA ,X+</td>
<td>Load A register from memory held in X register; add one to value in X register</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>27 05</td>
<td>BEQ 05</td>
<td>Branch if A register holds a zero</td>
</tr>
<tr>
<td>4</td>
<td>BD A3</td>
<td>0A</td>
<td>JSR $A30A</td>
<td>Jump to a subroutine at location A30A to display character in A register</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>20 F7</td>
<td>BRA LOOP</td>
<td>Branch back to line labeled LOOP</td>
</tr>
<tr>
<td>6</td>
<td>RETURN</td>
<td>39</td>
<td>RTS</td>
<td>Return to BASIC</td>
</tr>
<tr>
<td>7</td>
<td>DATA</td>
<td>53 55 43 43 45 53 53 21 0D 00</td>
<td></td>
<td>ASCII codes: S U C C E S S ! carriage return end of data</td>
</tr>
</tbody>
</table>
Let's look at the blocks one-by-one. Although we will not be using the assembly language mnemonics, they have been included to clarify the machine language op codes.

1. The first instruction consists of three hex values. The first value is the op code 8E, which tells the computer to load the X register with the two-byte value that follows 8E. That value is the first memory location where the data for the program are stored. The X register is a special 16-bit storage location.

The assembler mnemonic for this instruction is

LDX #DATA

Load the X reg.
the label of the first address of the data

Thus, the X register is loaded with the memory location 3E8E:

<table>
<thead>
<tr>
<th>0011</th>
<th>1110</th>
<th>1000</th>
<th>1110</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>E</td>
<td>8</td>
<td>E</td>
</tr>
</tbody>
</table>

2. The second instruction forms the beginning of a loop (that ends at step 5). This instruction consists of two bytes. The first byte (A6) tells the computer to load the A register. The second byte (80) tells the computer to load A from the memory location held in the X register (set in step one). It also says to increment the value in the X register by one. Therefore, on the next pass through the loop, the value loaded into the A register will come from the next higher memory location. Thus the data from step 7 are loaded into the A register, one value at a time, in the order given in step 7.

The assembler mnemonic is

LDA, X+

load the A register
from the address in the X register; increase X by 1

The first time through the loop:

<table>
<thead>
<tr>
<th>0101</th>
<th>0011</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
3. The third instruction consists of two bytes. The first byte (27) tells the computer to branch (if the A register holds a zero) the number of steps given in the second byte (5). Looking at step 2, you can see that the A register holds the value 53 on the first pass through the loop. Therefore, the branch will not be taken. The computer will ignore that instruction and move on to the next one. The assembler mnemonic for this instruction is

```
BEQ 05
```

branch if zero__/ Ls memory locations forward

4. The fourth instruction consists of three bytes. The first byte (BO) tells the computer to jump to a subroutine located at the address that follows in bytes two and three (A30A). This subroutine displays the character whose ASCII code is presently in the A register (53 hex). The character is an S. On the first pass through the loop, the computer will place the letter S on the screen:

```
S
```

The assembler mnemonic for the jump to the subroutine (built into the computer's ROM) that displays the character is

```
JSR $A30A
```

Jump to_J
le x l subroutine desired is located
subroutine va lue at memory lo cati on A30A

5. This instruction consists of two bytes. The first byte (20) tells the computer to branch. The second byte tells where to branch (back 11 steps to the step labeled LOOP). The assembler mnemonic is

```
BRA LOOP
```

branch~
Lto instruction labeled LOOP

6. This is a one-byte instruction. It is the return from subroutine. Its Op Code is 39. It will be executed only when reached from instruction 3 at the time the A register contains a zero. The assembler mnemonic is
HELP WITH THOSE MYSTERIOUS MACHINE CODES

RTS

7. This block is the data loaded by step 2. One byte is loaded at a time as the value in the X register changes.

<table>
<thead>
<tr>
<th>Value in X register</th>
<th>Address of Data</th>
<th>Value Loaded into A register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3E8E</td>
<td>53</td>
</tr>
<tr>
<td>1</td>
<td>3E8F</td>
<td>55</td>
</tr>
<tr>
<td>2</td>
<td>3E90</td>
<td>43</td>
</tr>
<tr>
<td>3</td>
<td>3E91</td>
<td>43</td>
</tr>
<tr>
<td>4</td>
<td>3E92</td>
<td>45</td>
</tr>
<tr>
<td>5</td>
<td>3E93</td>
<td>53</td>
</tr>
<tr>
<td>6</td>
<td>3E94</td>
<td>53</td>
</tr>
<tr>
<td>7</td>
<td>3E95</td>
<td>21</td>
</tr>
<tr>
<td>8</td>
<td>3E96</td>
<td>0D</td>
</tr>
<tr>
<td>9</td>
<td>3E97</td>
<td>00</td>
</tr>
</tbody>
</table>

A loop is executed from instruction 2 through instruction 5 until the A register contains the value zero (from memory address 3E97). At that time, it branches to step 6, where the computer is instructed to return to your BASIC program.

When the program ends, you will see this on the screen.

```
SUCCESS!
142 62 142 166 128 39 5
189 163 10 32 247 57 83 8
5 67 67 69 83 83 33 13 0
```

Figure 9-2. Display from program to POKE subroutine.

HELP WITH THOSE MYSTERIOUS MACHINE CODES

The machine language codes may seem complicated and mysterious, but the computer understands them completely. However, the computer very rigidly follows the instruction codes that are entered. It will try to obey what
you have requested even if the codes are incorrect. You must check the codes very carefully for errors before running the machine language program.

If you are seriously interested in learning the machine and assembly language used by the TRS-80 Color Computer, there are some useful tools available. Every programmer has personal preferences for tools. The following information should be considered as personal opinions of the author.

The best and most complete reference book for the 6809 instruction set and general 6809 assembly language programming is 6809 Assembly Language Programming by Lance A. Leventhal, Osborne/McGraw-Hill. It contains a wealth of examples and problems as well as completely documented reference material.

A more specific assembly language book aimed at assembly language programming the TRS-80 Color Computer is in preparation and will be published by Reston Publishing Company. It is coauthored by Don and Kurt Inman and will emphasize graphics techniques used in machine and assembly languages.

Beyond the stage of poking in machine language subroutines from BASIC, you will need other tools to enter your machine language programs directly. For machine language programs, I would recommend the CBUG Monitor from the MICRO WORKS, P.O. Box 1110, Del Mar, CA 92014. It is designed to work on a TRS-80 Color Computer. The monitor comes in two versions: a cassette tape and a 2K ROM. The ROM version eliminates the need to load the CBUG Monitor each time you want to use it. The ROM may be installed in an empty ROM socket on the main computer board, or it may be mounted in an external ROM PACK that plugs into the ROM PACK slot of the computer. Directions for installation, as well as monitor use, are provided in the CBUG Monitor owner's manual.

The MICRO WORKS also sells a Software Development System (SDS 80C). It is contained in a ROM PACK, which can be plugged into the ROM slot of the TRS-80 Color Computer. It has an Editor that aids in writing assembly language programs, an Assembler that accepts standard 6809 assembly language (source) statements and produces machine language (object) programs, and a machine language Monitor that allows direct access to the computer's registers and memory for debugging programs. A manual, describing its use in developing programs, is supplied.

Other systems are available, but the author has used those mentioned in developing this last chapter and the assembly language book that will follow.

LOW-RESOLUTION GRAPHICS

Shapes and creatures may be created in low-resolution graphics in much the same way as you created the text message "SUCCESS!" Low-resolution
graphic characters in various colors may be displayed by using graphics codes in place of the ASCII codes used for text. The graphic characters are blocks divided into four areas.

Each code presents a different color combination of the areas. The codes for green areas within the blocks are shown in Figure 9-3.

![Figure 9-3. Low-resolution graphics codes.](image)

The green areas may be replaced with other colors by adding a constant value to the above codes. These constants are

<table>
<thead>
<tr>
<th>Add</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>yellow</td>
</tr>
<tr>
<td>32</td>
<td>blue</td>
</tr>
<tr>
<td>48</td>
<td>red</td>
</tr>
<tr>
<td>64</td>
<td>buff</td>
</tr>
<tr>
<td>80</td>
<td>cyan</td>
</tr>
<tr>
<td>96</td>
<td>magenta</td>
</tr>
<tr>
<td>112</td>
<td>orange</td>
</tr>
</tbody>
</table>

A figure such as:

![A figure as:](image)

could be designed for
the screen on paper first. We'll use a 4 x 4 arrangement of blocks.

The graphics codes would be (left to right for each row)

1st row: 128, 141, 136, 128
2nd row: 142, 143, 142, 138
3rd row: 136, 143, 138, 136
4th row: 128, 138, 138, 128

Exercise 9-3

Fill in the graphics codes that would color the previous figure:

Head and neck - yellow
Body - blue
Legs - orange

1st row:  ,  ,  ,  
2nd row:  ,  ,  ,  
3rd row:  ,  ,  ,  
4th row:  ,  ,  ,  

(Answers are at the end of the chapter.)

You must use 16 data bytes to create the figure. We used 8 data bytes to create the text message "SUCCESS!" The machine language subroutine will be the same as that used for the text message except for 11 additional bytes. Therefore, we must reserve 34 memory locations for the subroutine. The BASIC program is changed slightly to display the figure in changing colors.

100  ' SET UP USR
110  CLEAR 34,16000
120  DEF USR0=16001
SUMMARY

Enter and RUN the program. Keep your eyes on the upper left corner of the screen. If you want to alter the speed of the color changes, modify line 610 or eliminate it completely.

The variable C is used to change the graphics codes from one color to another. The subroutine is read in at line 330 but is POKEd into memory only on the first pass through the loop (when C=0). Line 340 causes the computer to skip the POKE statement at line 350 when C>0.

The lines numbered 400-450 POKE in the graphics codes used for the figure. The data are modified at line 440 to change colors. Line 430 avoids modifying the carriage returns and the zero flag. The data values are restored at line 620.

Summary

The USR function provides a link between a BASIC program and a machine language subroutine used by the BASIC program. After a USR
function has been defined, it may be called from any point within the BASIC program. In this chapter, you learned

• to reserve space in memory for a machine language subroutine:

100 CLEAR 25,16000
save 25 memory locations

• to define a USR function:

120 DEF USR0=16001
USR function starting address of machine language subroutine
number (0-9)

• to POKE in machine language instructions and data:

210 FOR X=16001 TO 16025
220 READ A
230 POKE X,A
240 NEXT X

• to execute the machine language subroutine by the USR function call statement:

310 Y = USR0(0)
subroutine dummy argument number

• to use ASCII codes and low-resolution graphics codes in a machine language program to display text and graphics on the video screen.

You also learned

• that memory locations are used to store binary numbers consisting of one byte, or eight binary digits called bits:

one memory location
8 bits = 1 byte
one bit

• that binary digits may be one of two values: a zero or a one

• that the place values of binary digits increase from right to left by powers of two:
• that hexadecimal symbols have the following decimal values:

<table>
<thead>
<tr>
<th>hex</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>dec</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

• that one hexadecimal digit can represent four binary digits, and two
hex digits can represent 8 bits or one byte:

1101 binary = D hex
01011100 binary = 5C hex

• that hex to decimal and decimal to hex conversions may be calculated
by hand or by the computer

• how the following machine language instructions are used in a pro-
gram:

<table>
<thead>
<tr>
<th>Op Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>8E</td>
<td>Load the X register with the two-byte value that follows</td>
</tr>
<tr>
<td>A6 80</td>
<td>Load the A register from the memory location held in the X register; also increment the value in the X register</td>
</tr>
<tr>
<td>27</td>
<td>Branch if zero (branch the number of steps specified in the value that follows)</td>
</tr>
<tr>
<td>BD</td>
<td>Jump to the subroutine whose address follows</td>
</tr>
<tr>
<td>20</td>
<td>Branch the number of steps specified in the value that follows</td>
</tr>
<tr>
<td>39</td>
<td>Return from subroutine to BASIC program</td>
</tr>
</tbody>
</table>

Chapter Test

1. Write instructions to reserve 29 memory locations for a machine language subroutine, to define the USR function as number 3, and to call the ma-
chine language subroutine from a BASIC program.
THE USR FUNCTION

a. 110 __________________________ → reserve memory
b. 120 __________________________ → define USR
c. 410 __________________________ → call USR

2. If five more bytes are needed in the machine language subroutine that uses the instructions of exercise 1, which of the three instructions of exercise 1 would have to be changed and what would it be changed to?

a. change line _______________________

b. __________________

3. Write a loop to POKE in a program described in exercise 1. The data are to be READ from DATA statements.

210 FOR X = _______ TO _______
220 ____________________________
230 ____________________________
240 ____________________________

4. Write a FOR-NEXT loop to PEEK at the machine language subroutine of exercise 3.

510 ____________________________
520 ____________________________
530 ____________________________

5. Use exercises 1, 3, and 4 to write a program to display the words COLOR COMPUTER. The ASCII codes used are C=67, O=79, L=76, R=82, M=77, P=80, U=85, T=84, E=69, space=20.

6. Use the table of ASCII codes in Appendix E to write your own message for your program in exercise 5.
7. Fill in the following table of conversions between hex and decimal numbers.

<table>
<thead>
<tr>
<th>hex</th>
<th>decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>3F</td>
<td>310</td>
</tr>
<tr>
<td>2CD</td>
<td>16052</td>
</tr>
<tr>
<td>3EA5</td>
<td>16052</td>
</tr>
</tbody>
</table>

8. Give the graphics codes of the following characters and colors.

- black
  - yellow
  - magenta
  - red
  - buff

9. Write a BASIC program and a machine language subroutine to display a figure similar to the one in this chapter with arms raised as

10. Combine the previous program with the one in this chapter to display alternately

and
Answers to Exercises Within the Chapter

Exercise 9-1

SUCCESS!

<table>
<thead>
<tr>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>62</td>
<td>142</td>
<td>166</td>
<td>128</td>
<td>39</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>189</td>
<td>163</td>
<td>10</td>
<td>32</td>
<td>247</td>
<td>57</td>
<td>83</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>67</td>
<td>67</td>
<td>69</td>
<td>83</td>
<td>83</td>
<td>33</td>
<td>13</td>
<td>0</td>
</tr>
</tbody>
</table>
OK

Exercise 9-2

<table>
<thead>
<tr>
<th>Memory Location</th>
<th>Decimal Value</th>
<th>Hex Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>16001</td>
<td>142</td>
<td>8E</td>
</tr>
<tr>
<td>16002</td>
<td>62</td>
<td>3E</td>
</tr>
<tr>
<td>16003</td>
<td>142</td>
<td>8E</td>
</tr>
<tr>
<td>16004</td>
<td>166</td>
<td>A6</td>
</tr>
<tr>
<td>16005</td>
<td>128</td>
<td>80</td>
</tr>
<tr>
<td>16006</td>
<td>39</td>
<td>27</td>
</tr>
<tr>
<td>16007</td>
<td>05</td>
<td>05</td>
</tr>
<tr>
<td>16008</td>
<td>189</td>
<td>BD</td>
</tr>
<tr>
<td>16009</td>
<td>163</td>
<td>A3</td>
</tr>
<tr>
<td>16010</td>
<td>10</td>
<td>0A</td>
</tr>
<tr>
<td>16011</td>
<td>32</td>
<td>20</td>
</tr>
<tr>
<td>16012</td>
<td>247</td>
<td>F7</td>
</tr>
<tr>
<td>16013</td>
<td>57</td>
<td>39</td>
</tr>
<tr>
<td>16014</td>
<td>83</td>
<td>53</td>
</tr>
<tr>
<td>16015</td>
<td>85</td>
<td>55</td>
</tr>
<tr>
<td>16016</td>
<td>67</td>
<td>43</td>
</tr>
<tr>
<td>16017</td>
<td>67</td>
<td>43</td>
</tr>
<tr>
<td>16018</td>
<td>69</td>
<td>45</td>
</tr>
<tr>
<td>16019</td>
<td>83</td>
<td>53</td>
</tr>
<tr>
<td>16020</td>
<td>83</td>
<td>53</td>
</tr>
<tr>
<td>16021</td>
<td>33</td>
<td>21</td>
</tr>
<tr>
<td>16022</td>
<td>13</td>
<td>0D</td>
</tr>
<tr>
<td>16023</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Exercise 9-3

1st line: 128, 157, 152, 128
2nd line: 174, 175, 174, 170
3rd line: 168, 175, 170, 168
4th line: 128, 250, 250, 128

Answers to Odd-Numbered Exercises of Chapter Test

1. a. 110 CLEAR 29,16000
b. 120 DEF USR3=16001
c. 410 Y=USR3(0)

3. 210 FOR X=16001 TO 16029
   220 READ A
   230 POKE X,A
   240 NEXT X

5. 110 CLEAR 29,16000
   120 DEF USR3=16001
   210 FOR X=16001 TO 16029
   220 READ A
   230 POKE X,A
   240 NEXT X
   310 DATA 142,62,142,166,128,39,5,189
   320 DATA 163,10,32,247,57,67,79,76,79
   330 DATA 82,20,67,79,77,80,85,84,69,82
   340 DATA 13,0
   410 Y=USR3(0)

7. | Hex | Decimal |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1C</td>
<td>28</td>
</tr>
<tr>
<td>3F</td>
<td>63</td>
</tr>
<tr>
<td>136</td>
<td>310</td>
</tr>
<tr>
<td>2CD</td>
<td>717</td>
</tr>
<tr>
<td>3EB4</td>
<td>16052</td>
</tr>
<tr>
<td>3EA5</td>
<td>16037</td>
</tr>
</tbody>
</table>

9. 110 CLEAR 34,16000
   120 DEF USR0=16001
210 DATA 142,62,142,166,128,39,5,189
220 DATA 163,10,32,247,57,138,141,136,138,13
230 DATA 140,143,142,136,13,128,143,138,128,13
240 DATA 128,138,138,128,13,0
310 FOR X=16001 TO 16034
320 READ A
330 POKE X,A
340 NEXT X
410 Y=USR0(0)
Additional Programs

This Section contains programs demonstrating additional uses of the knowledge that you gained in the first part of the book. The programs are grouped to correspond to the chapters in Part I and are numbered accordingly, 1A, 2A, 3A, etc. A detailed explanation is provided for each program so that you can understand the uses of the graphics statements more thoroughly. The explanations also make it easier for you to modify our examples to fit your own needs.

You can use the programs as a basis for designing programs of your own. Modify these programs, expand them, or just RUN them as they are. We believe that you can learn more efficiently by seeing new statements in the context of their use in a program than by examples of the statements in an isolated setting.

As more new statements are encountered, the programs will grow in length; it may be to your advantage to save the ones that you intend to use more than once on cassette or disk. We have found the cassette system of the color computer to be quite reliable and reasonably fast. At the time the book was being prepared, the Radio Shack disk system was not available. Therefore, we have relied on the cassette system exclusively for a permanent record of the programs.
Fundamental Coloring

The programs in this section are restricted to the graphics statements introduced in Chapter 1 of Part 1. These statements are

- PCLS
- PSET
- PMODE
- PRESET
- SCREEN
- COLOR

PROGRAM 1-1—HORIZONTAL COLOR BARS

As the name suggests, this program produces a series of color bars on the display. The graphics commands, PSET with SCREEN 1,0 and SCREEN 1,1, are used to draw horizontal bars of colors green, yellow, blue, and red in that order. The bars are repeated from top to bottom of the screen. Both color sets are then used to alternate the pattern of the colors:

- first, green    then, buff
- yellow         cyan
- blue           magenta
- red            orange

The program continues to alternate the color sets until the BREAK key is pressed.

Order of bars:

from SCREEN 1,0  from SCREEN 1,1
green        buff
yellow       cyan
blue         magenta
red          orange

... etc. ...
Program 1-1

1000 PMODE 1,1 '**LOW RESOLUTION, FIRST PAGE
1010 SCREEN 1,0 '**GRAPHICS MODE, FIRST COLOR SET
1020 PCLS ' **CLEAR THE SCREEN
1030 FOR Y = 0 TO 191 STEP 2 ------this loop fills the
1040 Z = (Y/8-INT(Y/8))*4+1 screen with green,
1050 FOR X = 0 TO 255 STEP 2 yellow, blue, and red
1060 PSET(X,Y,Z) horizontal lines, colors
1070 NEXT X calculated in line 1040
1080 NEXT Y
1090 SCREEN 1,1 '**FLIP COLORS change the lines to
1100 FOR W = 1 TO 1500: NEXT W buff, cyan, magenta,
1110 SCREEN 1,0 and orange, the second color set
1120 FOR W = 1 TO 1500: NEXT W then flip the colors
1130 GOTO 1090 back and forth by alternating the first and

Program Description

Lines 1000-1020 set the screen for graphics in PMODE 1 with the first color set (SCREEN 1,0).

As the outer FOR-NEXT loop increases Y in steps of two, line 1040 calculates the color to be used (Z). If you try a few calculations, you will see that Z goes through the sequence: 1,2,3,4,1,2,3,4,1, etc., bringing the colors green, yellow, blue, and red in that order for use in a given bar. The inner FOR-NEXT loop (lines 1050-1070) plots the horizontal bar with the PSET statement (line 1060) using the calculated color.

After the screen is filled (the FOR-NEXT loops are completed), line 1090 switches to the second color set (SCREEN 1,1). The lines then appear as buff, cyan, magenta, and orange. After a brief delay at line 1100, the color set is changed back at line 1110 (SCREEN 1,0) to the first color set. This process continues until you tire of the program and press the BREAK key.

Program 1-2—VERTICAL COLOR BARS

Vertical color bars are displayed on the screen by this program, using PSET with SCREEN 1,0 and SCREEN 1,1. It is similar to program 1-1, but the FOR-NEXT loops using the variables X and Y are reversed. This time X

---

ADDITIONAL PROGRAMS
is used in the outer loop, and \( Y \) is used in the inner loop. This reversal causes the bars to be drawn vertically rather than horizontally.

Program 1-2

```plaintext
1000  PMODE 1,1 'LOW RESOLUTION
1010  SCREEN 1,0 'GRAPHICS, COLOR SET)
1020  PCLS ' CLEAR THE SCREEN
1030  FOR X = 0 TO 255 STEP 4
1040  \( Z = (X/16-\text{INT}(X/16))*4+1 \)
1050  FOR Y = 0 TO 191 STEP 2
1060  PSET(X,Y,Z): PSET(X+2,Y,Z)
1070  NEXT Y
1080  NEXT X
1090  SCREEN 1,1
1100  FOR W = 1 TO 1500: NEXT W
1110  SCREEN 1,0
1120  FOR W = 1 TO 1500: NEXT W
1130  GOTO 1090
```

Once again, lines 1000-1020 prepare the screen for graphics with the first color set being used (green, yellow, blue, and red).

The variables in the FOR-NEXT loops are reversed in lines 1030-1080 from the order used in program 1-1. Notice that the step in the outer loop (change in \( X \)) has been increased to 4. This was done because it is harder to distinguish the colors when the lines are drawn vertically. Consequently the
calculation of Z in line 1040 was also changed. Z still gives the color values 1, 2, 3, and 4. Line 1060 was changed to set two points at once so that the lines are twice as wide as normal. This helps a bit in distinguishing the individual colors.

Other than lines 1030-1080, the program is the same as the horizontal color bar program.

PROGRAM 1-3—BLINKING PARALLEL BARS

PSET is used in this program to draw parallel lines of various colors. Short lines begin at the center of the screen. As the computer draws, two lines at once, the lines become longer as the top and bottom of the screen are approached. When the screen is filled, the color sets are alternated to give a surprising visual effect.

```
1000 PMODE 1,1 'LOW RESOLUTION
1010 SCREEN 1,0 'COLOR SET 0
1020 PCLS 'CLEAR SCREEN FOR GRAPHICS
1030 A = 126: B = 94: C = 2 'set positions
1040 Z = ((C+2)/16-INT((C+2)/16))*4+1 'choose color
1050 FOR X = A TO A+C 'draw parallel lines
1060 PSET(X,B,Z): PSET(X,B+C,Z)
1070 NEXT X
1080 A = A-2: B = B-2: C = C+4 'change positions and
1090 IF B> 32 THEN 1040 'decide if should go back
1100 SCREEN 1,1 'change color sets
1110 FOR W = 1 TO 400: NEXT W
1120 SCREEN 1,0 'change color sets again
```
This program starts drawing horizontal color bars near the center of the screen. The lines start out small. Their length is controlled by the parameter, C. C begins with a value of 2, and increases by 4 in line 1080 each time a new line is drawn. The starting point of each line is controlled by the parameter, A. A starts at a value of 126 and decreases by 2 each time a new line is drawn. The row in which the line is drawn is controlled by the parameter B. B starts at row 94 and decreases by 2 for each new line. The PSET statements in line 1060 sets two points so that two lines are drawn in the FOR-NEXT loop of lines 1050 to 1070.

Lines 1100-1140 change the colors set used in the display so that the colors blink back and forth from one color set to the other.

Try using this program in PMODE 3, which also has four colors but draws finer lines.

**PROGRAM 1-4—BLINKING RECTANGLES**

Horizontal and vertical bars are combined in this program to produce rectangles in color. PSET is once again used. The size of the rectangles grow from small to large as the display is drawn. When all the rectangles have been drawn, the color sets are alternated to produce an unusual color effect.
In Chapter 1, you learned to draw both vertical and horizontal lines. This program puts the vertical and horizontal lines together to form a colored rectangle.

The beginning and ending parameters for the FOR-NEXT loops are changed after both are completed so that a larger rectangle will be drawn each time. The small rectangles start near the center of the screen. The starting point and the length of the sides are changed to create larger and larger rectangles of different colors.

Both top and bottom of the rectangle are drawn first by the FOR-NEXT loop at lines 1050–1070. Then both sides are drawn by the FOR-NEXT loop at lines 1080–1100.

Line 1110 calculates the new starting points for the rectangles (A and B). C determines how long the line will be. It is increased by 4 each time a new rectangle is drawn. Line 1120 checks to see if you are getting too close to the edge of the screen. If you are not too close, line 1120 sends the program back to draw another rectangle. When B is less than or equal to 10, the program proceeds to line 1130, and the usual switching from color set 0 to color set 1 begins. We shortened the time delay this time so that the colors would switch back and forth quickly.
REVISION TO PROGRAM 1-4

You can let the computer randomly choose the colors to be used with a few minor changes to program 1-4. The rectangles will then produce random patterns. Try these changes and additions.

1020 PCLS 1
1040 Z = RND(4)+4 ←random values 5 through 8
1045 IF Z = 5 THEN 1040 ←optional to exclude the background color
1120 IF B > 32 THEN 1040 ←fewer lines
1125 FOR R = 1 TO 5 ←blink 5 times
1140 FOR W = 1 TO 500: NEXT W
1160 FOR W = 1 TO 500: NEXT W
1165 NEXT R
1170 GOTO 1020 ←repeat

Each time line 1170 is executed, a new pattern will be created. Feel free to change the value in the delay loops (lines 1140 and 1160) to produce the desired viewing time.
The programs in this section may use any of the graphics statements introduced in Chapter 1. In addition, the following statements that you learned about in Chapter 2 are used:

LINE
CIRCLE

PROGRAM 2-1—RANDOMLY PLACED CIRCLES

This program uses the CIRCLE statement in its basic form. The position of the center and the radius are the only parameters used in the statement. The color of the circle is determined randomly and used in the COLOR statement as the foreground color.

The center position and radius are also selected randomly. This results in circles of various size and color being randomly placed on the screen.
Program 2-1

1000 PMODE 3,1 ← medium resolution with four colors
1010 PCLS 5 ← buff background
1020 SCREEN 1,1 ← color set 1
1030 C = RND(4)+4 ← color numbers 5-8
1040 R = RND(80); A=RND(175); B=RND(150) ← random radius and center
1050 COLOR C,5 ← use color C
1060 CIRCLE (A+50,B+50),R ← draw circle
1070 GOTO 1030 ← get data for new circle and draw another

This program draws consecutive circles of varying colors and places them randomly on the screen. The center of the circles is determined by the random values of A and B (line 1040). The colors are chosen randomly from the specified color set by the random value C (line 1030). The radius is selected from the random value R (line 1040) to create circles of different size (radius 1-80).

In some cases the circle may hit the edge of the screen, flattening one or two sides. You may want to alter the random values chosen in line 1040 to prevent this from happening.

To stop the program, press the BREAK key. You could limit the number of circles by wrapping a FOR-NEXT loop around lines 1030-1060. For example,

1025 FOR X = 1 TO 15
1070 NEXT X
1080 PCLS: GOTO 1025

You might also want to add a time delay following the FOR-NEXT loop to allow you to see the final pattern produced such as

1075 FOR W = 1 TO 1500: NEXT W

PROGRAM 2-2—CONCENTRIC CIRCLES

Twenty-seven concentric circles of random color and random size are produced in this program. Some circles are drawn over the top of others. The LINE statement is used to draw a colored "curtain" over the display before 27 more circles are drawn on the colored background provided by the LINE statement.
Program 2-2

1000       PMODE 3,1  ← set up resolution for graphics;  
            PCLS 5  choose background color and  
            SCREEN 1,1  color set  
1030       FOR X = 1 TO 27  ← draw 27 concentric circles with  
            C = RND(4)+4  random color and  
3050       R = RND(60)  random radius  
1060       COLOR C,5  
1070       CIRCLE(128,96),R  
1080       NEXT X  
1090       FOR W = 1 TO 200: NEXT W  ← wait to see final pattern  
1100       LINE(5,5)-(250,186),PSET,BF  ← pull down a colored  
            curtain determined by  
1110       FOR W = 1 TO 500: NEXT W  last used C  
            ←wait  
1120       GOTO 1030  ← use new background and  
            do it again  

Although this program is short, it creates some fascinating color patterns. It starts with a buff background. The FOR-NEXT loop draws 27 concentric circles of different colors. Sometimes the background color is selected at line 1040. This causes an unseen circle or the erasure of a previous circle. Since the radius for each circle is chosen randomly in line 1050, some circles may coincide. Therefore, you may not see all the circles when the loop is completed.

Line 1100 draws a rectangle that fills a large portion of the screen, overlaying the previously drawn circles as the rectangle is filled with the last color selected in line 1040.
The rectangle is used for a background as line 1120 sends the computer back to draw a new series of 27 circles. An ever-changing color pattern providing you with increasing enjoyment is created. Watch this program; you may not be able to tear yourself away. The next pattern may be more interesting than the present one.

PROGRAM 2-3—SPLIT SCREEN

Randomly placed circles and rectangles are drawn on alternate sides of the screen. The background color changes to present a new field for the next set of circles and rectangles. The display alternates between left and right halves of the screen.

Program 2-3

```
1000 PMODE 3,1 ← set up the mode, color set and background color
1010 PCLS 1
1020 SCREEN 1,0
1030 FOR A = 1 TO 4 ← go through the process four times
1040 B = 5*((A*4/5)-INT(A*4/5)) ← color B for drawing
1050 COLOR A,1 ← color A for left half background
1060 LINE(0,0)-(127,191),PSET,BF ← color left half background
1070 COLOR B,1 ← use color B for foreground
1080 FOR X = 1 TO 5 ← draw on left half
1090 F=RND(60)+5: G=RND(90)+5
1100 CIRCLE(F+20,G),10
1110 LINE(F+15,G+65)-(F+45,G+105),PSET,BF
1120 NEXT X
1130 LINE(128,0)-(255,191),PSET,BF ← color right half background
```
1140 COLOR A,1 ←use color A for foreground
1150 FOR X = 1 TO 5 ←draw on right half
1160 G=RND(90)+5: H=RND(90)+128
1170 CIRCLE(H+20,G),15
1180 LINE(H-5,G+65)-(H+5,G+105),PSET,BF
1190 NEXT X
1200 FOR W=1 TO 500: NEXT W ←wait awhile
1210 NEXT A ←then go on
1220 GOTO 1030 ←repeat

Two colors, A and B, are used to color each half of the screen alternately. As one half is colored, the second color is used to draw circles and rectangles on the split screen. A is determined by the parameters of the FOR-NEXT loop. B is determined by the equation in line 1040. The color pairs will be

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Lines 1050 and 1060 use color A to fill in a background for the left side of the screen. Line 1070 selects color B to draw the figures.

Line 1130 then uses color B to provide a background for the right half of the screen and line 1140 selects color A to draw the figures.

Lines 1090 and 1160 randomly choose values used to place the circles and rectangles within the correct half of the screen. A delay is used at line 1200 to let you view the finished pattern before going on to new colors. Line 1220 sends the computer back for a repeat performance.
More Circles, PAINT, and POINT

The programs in this section are restricted to the graphics statements introduced in Chapters 1 and 2. In addition, the following statements that you learned in Chapter 3 are used:

PAINT
PPOINT

Since more statements have been introduced in each chapter, the programs are becoming a little longer. You are also able to accomplish more complex graphics.

PROGRAM 3.1–VERTICAL COLOR BARS WITH LINE AND PAINT

This program draws 52 narrow vertical lines using random colors: cyan, magenta, and orange. The PAINT statement is then used to paint vertically between two bars of the same color starting at a random point. The painted area may be narrow or broad, depending upon the distance between the vertical bars being colored. The painting color is chosen randomly.
Program 3-1

1000 PMODE 1,1 — set mode and screen
1010 PCLS 5 — second color set
1020 SCREEN 1,1

1030 FOR X = 0 TO 255 STEP 5 — draw vertical lines five spaces apart
1040 Z = RND(4)+4: IF Z=5 THEN 1040 — color select
1050 COLOR Z,5
1060 LINE(X,0)-(X,191),PSET — line from top to bottom
1070 NEXT X

1080 FOR M = 1 TO 10 — 10 paintings
1090 P=RND(255): R=RND(191) — random starting point
1100 S=RND(4)+4: T=RND(4)+4 — random colors
1110 PAINT (P,R),S,T
1120 NEXT M

1130 FOR W = 1 TO 1000:NEXT W — look at the final pattern
1140 GOTO 1010 — do it all again

The loop of lines 1030 through 1070 draws vertical lines using cyan, magenta, and orange on a buff background. The lines are drawn five locations apart by the STEP 5 in line 1030. One of the colors is chosen randomly at lines 1040 and 1050. After the 52 lines are drawn, the PAINT statement of line 1110 uses a random point (P,R) and paints from one vertical bar of color T to the closest line of the same color. The random color S is used as paint. Ten of these wide color bars are drawn by the loop of lines 1080–1120. These wide bars may be cyan, magenta, or orange.

A delay allows you to look at the final result before a new display is drawn.

PROGRAM 3-2—RANDOMLY COLORED CIRCLES WITH PAINT AND PPOINT

Forty circles of random color are randomly drawn on the screen and filled with color by this program. You probably won’t see all 40 circles. The PPOINT statement is used to test the color of the point where a new circle is to be drawn. If that point is already the same color that has been selected for the new circle, the computer erases the old circle and does not draw a new one.
Program 3-2

1000 PMODE 3,1
1010 PCLS
1020 SCREEN 1,0: C=0
1030 FOR N = 1 TO 40
1040 X = RND(215)+20
1050 Y = RND(151)+20
1060 IF PPOINT(X,Y) = C THEN PAINT(X,Y), 1, 1: GOTO 1110
1070 C = RND(4)
1080 IF C = 1 THEN 1070
1090 CIRCLE(X,Y), 10,C
1100 PAINT(X,Y), C, C
1110 NEXT N
1120 GOTO 1010

This program places a circle with center at a random location (21-235 for X and 21-171 for Y) chosen by lines 1040 and 1050. The point chosen for the center is then tested for color at line 1060. If the color C to be used already exists at point X,Y, line 1060 paints the area green (the background color) so that is erased. A new circle is not drawn in this case. The GOTO statement following the IF statement in line 1060 bypasses the circle drawing procedure. If a new circle is to be drawn (the IF statement is not true), lines 1070 and 1080 choose the color for the new circle from yellow, blue, or red. Green is rejected by line 1080.

The circles are drawn by line 1090 and painted with the same color by
line 1100. The procedure is repeated 40 times by the FOR-NEXT loop (lines 1030-1110). The screen will probably not show all 40 circles due to the erasures made by line 1060. Line 1120 then returns to the beginning to clear the screen and start over.

You can vary the size of the circles by changing the radius in line 1090. Other modes and color sets may be selected by lines 1000 and 1020. Also change line 1070 and 1080 if you change color sets.

PROGRAM 3-3—ROTATING PIE

A circle is divided into eight sections, like pieces of a pie, in this program. Alternate sections are painted a different color (cyan or magenta). When all sections are painted, they are repainted (one at a time) with the other of the two colors. Cyan sections are painted magenta, and magenta sections are painted cyan. This gives an illusion of a rotating color wheel.

Program 3-3

```
1000 PMODE 3,1
1010 PCLS
1020 SCREEN 1,1
1030 ' DRAW THE PIE
1040 CIRCLE(128,92),46,8
1050 ' CUT IT IN SECTIONS
1060 LINE(128,92)-(131,95),PSET,BF
1070 LINE(128,92)-(174,92),PSET
1080 LINE(128,92)-(160,122),PSET
1090 LINE(128,92)-(128,138),PSET
1100 LINE(128,92)-(96,122),PSET
1110 LINE(128,92)-(82,92),PSET
```
This program draws a circle (colored orange) with a radius of 46. It then sections it into eight pieces (lines 1070-1140). The sections are alternately filled with magenta or cyan colors in lines 1160-1310. This is accomplished by the subroutine at lines 1400 to 1420.

The first section is colored cyan (C=6) at line 1170. Line 1180 calls the subroutine which changes C to 7 (magenta).

The second section is colored magenta at line 1190. Line 1200 calls the subroutine which changes C to 8, but line 1410 changes C back to 6 (cyan).

The third section is colored cyan at line 1210. Line 1220 calls the subroutine again to change C to 7. This process repeats.
until the last section is painted magenta (C=7) at line 1310. Line 1320 calls the subroutine, changes C to 8, then back to 6. It returns to line 1330 where C is changed to 7. Line 1340 then sends the computer back to line 1170 where the first section is painted magenta. As the process continues, each cyan section is painted magenta, and each magenta section is painted cyan. Thus, it gives the appearance of revolving sections.

PROGRAM 3-4—CIRCLE MATCHING GAME

This game tests your ability to remember the placement of geometric shapes. Three circles are drawn near the top of the screen. They are not really circles but ellipses of different shape in a random order. They are numbered 1, 2, and 3.

A fourth ellipse is drawn near the bottom of the screen. It is the same shape as one of the ellipses at the top. The object of the game is to match the fourth figure with one of the first three. You type in the number of the ellipse that you think the fourth one matches.

There is one catch to this easy sounding game. The three shapes at the top are erased before the fourth one appears on the screen. Only the numbers of the first three are visible. You must remember the positions of the first three shapes.

Program 3-4

1000 PMODE 3,1
1010 PCLS
1020 SCREEN 1,1
1030 'DRAW FIRST "CIRCLE"
1040 E(1) = .4*RND(3) + .4: C = RND(3)+5
1050 CIRCLE(88,12),10,C,E(1)
1060 'DRAW 2ND "CIRCLE"
1070 E(2) = E(1) + .4: IF E(2) > 1.6 THEN E(2) = .8
1080 CIRCLE(128,12),10,C,E(2)
1090 'DRAW 3RD "CIRCLE"
1100  \[E(3) = E(2) + 0.4: \text{IF } E(3) > 1.6 \text{ THEN } E(3) = 0.8\]
1110  \text{CIRCLE(168,12),10,C,E(3)}
1120  \text{' NUMBER THE CIRCLES}
1130  \text{LINE(88,30)-(88,37),PSET}
1131  \text{PSET(128,30) :PSET(130,29) :PSET(132,29)}
1132  \text{PSET(134,30) :PSET(134,31) :PSET(132,32)}
1133  \text{PSET(130,34) :PSET(128,35) :LINE(128,36)-(134,36),PSET}
1134  \text{PSET(168,30) :PSET(170,29) :PSET(172,29)}
1135  \text{LINE(174,31)-(175,35),PSET}
1136  \text{PSET(172,32) :PSET(168,35)}
1137  \text{PSET(170,36) :PSET(172,36)}
1140  \text{' PAINT THE INSIDE OF THE CIRCLES}
1150  \text{PAINT(88,12),C,C}
1160  \text{PAINT(128,12),C,C}
1170  \text{PAINT(168,12),C,C}
1171  \text{' WAIT THEN ERASE}
1172  \text{FOR W = 1 TO 500: NEXT W}
1173  \text{PAINT(88,12),5,5}
1174  \text{PAINT(128,12),5,5}
1175  \text{PAINT(168,12),5,5}
1180  \text{' CHOOSE A RANDOM "CIRCLE"}
1190  \text{E(4) = 0.4*RND(3)+0.4}
1200  \text{CIRCLE(128,152),10,C,E(4)}
1210  \text{PAINT(128,152),C,C}
1220  \text{' LABEL CIRCLE TO BE MATCHED}
1221  \text{LINE(148,154)-(154,154),PSET}
1222  \text{LINE(148,150)-(154,150),PSET}
1223  \text{PSET(171,144) :PSET(173,142) :LINE(175,140)-(179,140),PSET}
1224  \text{PSET(181,142) :PSET(182,144) :PSET(182,146)}
1225  \text{PSET(181,148) :LINE(179,150)-(179,154),PSET}
1226  \text{PSET(179,160)}
1230  \text{' TEST FOR MATCHING CIRCLES}
1240  \text{A$=INKEY$: IF A$="" THEN 1240}
1250  \text{B = VAL(A$)}
1260  \text{IF E(4) = E(B) GOTO 1300}
1270  \text{CLS: PRINT "YOU MISSED. TRY A NEW ONE."}
1280  \text{FOR W = 1 TO 500: NEXT W}
1290  \text{GOTO 1010}
1300  \text{CLS: PRINT "RIGHT ON."}
1310  \text{PRINT "PRESS ANY KEY TO CONTINUE."}
1320  A$=INKEY$: IF A$ = "" THEN 1320
1330  GOTO 1010

The program draws three randomly colored ellipses at the top of the screen. They are labeled from left to right 1, 2, and 3. After a short time delay the three ellipses will be erased, but their numbers will remain. One of the three figures will then be duplicated at the bottom of the screen.

Alongside the bottom ellipse, a question mark will be displayed to indicate that the player is to type in the number of the ellipse (1, 2, or 3) that the lower ellipse matches.

After the player types in one of the three numbers, the computer returns to the text mode to give the appropriate evaluation of the choice. You have only one chance to get the correct answer. The computer will draw a new series of three ellipses after its evaluation of your answer.
The programs in this section are restricted to the graphics statements introduced in Chapters 1, 2, and 3. In addition, one new statement that you learned about in Chapter 4 is used:

DRAW

The DRAW statement has many options and is probably the most powerful of all the graphics statements. Therefore, a complete chapter was devoted to it in this book. Once again, we'll list the format and options of the DRAW statement.

format: \texttt{DRAW "command string"}

options:
- M for move
- U for up
- D for down
- L for left
- R for right
- A for angle
- N for no update

E for 45°
- F for 135°
- G for 225°
- H for 315°
- B for blank
- S for scale

Substrings are allowed

\hline

PROGRAM 4-1—ROTATING ARM

A rotating arm is created by successively drawing and erasing the arm. Each time a new arm is drawn, it appears 45° clockwise from the preceding position. To accomplish this, the DRAW statement is catenated from several segments. The angle, color, blank move, up and 45° direction commands are used.
Program 4-1

1000 PMODE 3,1
1010 PCLS
1020 SCREEN 1,1
1030 ' SET BASIC MOVEMENTS
1040 F$ = "BM120,90;U40"
1050 G$ = "BM120,90;E30"
1060 ' ASSIGN ANGLE VARIABLES
1070 FOR Z = 0 TO 3
1080 A$(Z) = "A"+STR$(Z)
1090 NEXT Z
1100 ' DRAW CIRCLE
1110 CIRCLE(120,90),8,7
1120 ' DRAW AND ERASE ARMS
1130 FOR B = 0 TO 3
1140 DRAW A$(B)+"C8"+F$
1150 DRAW A$(B)+"C5"+F$
1160 DRAW A$(B)+"C8"+G$
1170 DRAW A$(B)+"C5"+G$
1180 NEXT B
1190 GOTO 1130

After the screen is set in lines 1000 through 1020, the strings F$ and G$ define the basic movements in lines 1040 and 1050. 

The FOR-NEXT loop (lines 1070-1090) assigns the angle E 30 commands to be used: A0=0°, A1=90°, A2=180°, and A3=270°. A pivot around which the arm will revolve is drawn at line 1110.
PROGRAM 4-2—RECTANGLES ON AND OFF

This program draws a square near the center of the screen. Sometimes the square is filled with color. Sometimes it is not. If it is filled, another square appears in the upper left corner of the screen. It will remain there as long as the center square is blinking on and off. When the center square is not filled with color, the upper square is turned off.

Program 4-2

1000 PMODE 0,1 ← set screen
1010 PCLS ← buff border and foreground
1020 SCREEN 1,1 black background
1030 'DEFINE VARIABLES FOR DRAW
1040 A$="C5;BM10,10;R20D20L20U20" ← buff color
1050 B$="CO;BM10,10;R20D20L20U20" ← black to erase
1060 C$="C5;BM120,90;R20D20L20U20" ← buff color
1070 ' PICK A NUMBER
1080 A = RND(2)
1090 ON A GOSUB 1210,1260
1100 'TEST CENTER
1110 IF PPOINT(128,96)=5 THEN DRAW A$ ← if center filled,
1120 FOR W = 1 TO 400: NEXT W draw upper square
1130 ' ERASE AND REDRAW
1140 PAINT(128,96),0,0: DRAW C$ ← erase fill and redraw
1150 FOR W = 1 TO 10: NEXT W center square
1160 ' DO IT AGAIN
1170 GOTO 1080
1200 ' SOLID SQUARE
ADDITIONAL PROGRAMS

1210 DRAW C$ ← draw center square
1220 PAINT(128,96),5,5 ← fill with buff
1230 FOR W = 1 TO 30: NEXT W
1240 RETURN
1250 ' ERASE UPPER SQUARE
1260 DRAW B$
1270 RETURN

Three variables are used to define three DRAW commands (lines 1040–1060) used in the program:

A$ defines a square in the upper left corner of the screen.
B$ defines the same square as A$ but uses the background color to erase.
C$ defines a square near the center of the screen. Sometimes it is painted, sometimes not.

A is used as a random number, 1 or 2 (line 1080), to determine which of two subroutines (1210 or 1260) is used. One subroutine (line 1210) draws C$ and fills it with color. The other subroutine (line 1260) draws B$ to erase A$ (if it is on the display).

Line 1140 erases the center square (and its contents) and then redraws the square, unfilled. Thus, when a colored square is present, it will blink as long as the first subroutine is selected. While the center square is blinking, the square in the upper left corner will stay on the display. If subroutine 1260 is selected (A = 2), the upper square is removed by B$. The center square will not be filled in this case.

Press the BREAK key to stop the program.

PROGRAM 4-3—SHOOTING BLANKS AND LIVE SHOTS

A circle is drawn and painted in this program. The circle will appear in cyan, then magenta, and then orange. DRAW statements are used to create colored dashes emitting from the circle. Blank moves provide the dashes, which are created from lines drawn from the center of the circle.

The circle is painted one color. The dashes appear in orange, then magenta, then cyan. Buff is used to erase the dashes. A new color is used for the circle, and the process repeats, creating an ever-changing color pattern.
The FOR-NEXT loop of lines 1030–1150 provides the colors for the circle: first cyan, then magenta, then orange. The circle is drawn at line 1040 and painted at line 1050.

The FOR-NEXT loop at lines 1060–1130 provides orange, magenta, cyan, and buff (for erasing) dashes. Lines 1080, 1100, and 1120 provide short time delays for viewing.

Line 1140 clears the screen for each new circle to be produced when line 1150 sends the computer back for a new color.
The program is returned to the beginning by line 1160. Press the BREAK key to stop.

PROGRAM 4-4—SUBROUTINE TO BUILD A DRAW STATEMENT

In many cases you may want to experiment with drawing before you formalize a DRAW statement. This program might be used for experimentation, or it might be used as a subroutine inside a main program.

The program begins with the “pen” in the home position near the center of the screen. You then are asked whether you want the pen up or down. If the pen up choice is made, the program lets you move the pen with relative motion to any position on the screen without drawing. If the pen down choice is made, your drawing will begin from the center of the screen.

You can build a drawing with individual commands. The computer makes the addition to the drawing and shows the temporary result. To add more lines to the drawing, press any key. The computer once again will ask you for the next move.

Program 4-4

1000 Z = 0
1010 CLS: CLEAR — clear screen and variables
1020 INPUT “PEN DOWN -YES OR NO”: Y$ — choose draw or blank move
1030 CLS: S$ = “”: D$ = “” — blank old strings
1040 IF LEFT$(Y$, 1) = “Y” THEN GOSUB 4000 ELSE GOSUB 5000
1050 A$ = A$ + S$ + D$
1060 INPUT “HOW FAR WILL YOU MOVE”: M
1070 CLS
1080 A$ = A$ + STR$(M)
1100 Z = Z + 1
The program begins in the text mode. Line 1020 allows you to decide whether you want a blank move (pen up) or a drawing move (pen down). Subroutine 4000 is selected for pen down; subroutine 5000 for pen up at line 1040 after the screen has been cleared at line 1030.

Subroutine 4000 allows a choice of direction and then returns to the main program at line 1050.

Subroutine 5000 lets you make a blank move. Be sure to use a + or - sign in front of the X direction move. If the Y direction move is negative, the - sign must be given for it also. The computer then goes to the subroutine at line 4000 for a choice of direction for a pen down move. Then a return is made to 5050, which returns to the main program at line 1050.

Line 1050 concatenates the new direction to the previous portion of the string. Line 1060 then asks the magnitude of the move. This is concatenated to the string at line 1080.

Line 1100 determines whether this is the first drawing. Lines 1110-1140 then prepare the graphics display and make the drawing. Line 1120 determines whether the graphics screen is to be cleared (only on the first move) or whether a previous set of moves is to be left on the graphics screen.

Remember that the computer starts at the center of the screen as the first move is made. Include a pen-up blank move if you do not want to start the drawing from the center of the screen.
This section contains programs that may use any of the graphics statements introduced in the first four chapters. In addition the programs may contain these new graphics statements and functions:

PCLEAR       PCOPY
Scaling a DRAW Turning Pages

PROGRAM 5-1—MIXED UP SQUARES

Since numeric variables cannot be used directly in a DRAW statement, this program makes use of the string function, STR$, to define numeric portions of a DRAW statement in an acceptable form. The scale command is used to change the size of the displayed square. The relative move command is used to move the square toward the right and downward.

In the first part of the program, this is done in a regular manner, small to large. Then the scale is randomly selected, and squares of different size appear in random places on the screen. The program may even return to the earlier part of the program and present the squares in an orderly way again.
Program 5-1

1000 PMODE 4,1
1010 PCLS
1020 SCREEN 1,1
1030 DRAW "BM40,40"
1040 A$ = "R8D8L8U8"
1050 ' REGULAR INCREASE IN SIZE
1060 FOR X = 1 TO 10
1070 S$ = "S" + STR$(X)
1080 DRAW S$ + "BM" + STR$(X) + "," + STR$(X) + A$
1090 FOR W = 1 TO 100: NEXT W
1100 PCLS
1110 NEXT X
1120 ' STIR IT UP
1130 Y = RND(12): X = Y - 6
1140 S$ = "S" + STR$(Y)
1150 IF X >= 0 THEN X$ = "+" ELSE X$ = ""
1160 DRAW S$ + "BM" + X$ + STR$(X) + "," + X$ + STR$(X) + A$
1170 FOR W = 1 TO 100: NEXT W
1180 PCLS
1190 IF X > 4 THEN GOTO 1030
1200 GOTO 1130

After the screen is set, the DRAW statement with no movement at line 1030 fixes the original position in the upper left part of the screen. Line 1040 defines the basic square to be used.

The FOR-NEXT loop (lines 1060-1110) draws squares of ever-increasing size, moving the square down and to the right each time. When X = 1, the scale factor is one-fourth, and a small square 2X2 is drawn. The relative move in line 1080 moves the square over and down each time through the loop. As the scale factor increases, both the size and relative movement change. Line 1100 erases the square, and line 1110 sends the computer back for the next larger square.

Lines 1120-1190 really mix things up. The scale factor and the relative moves are randomly selected. If the random number gets too big, line 1190 sends the computer back to some sense of order (the first part of the program).

Lines 1080 and 1160 show how string variables and string portions can be concatenated to form the DRAW statement.

Line 1150 assures that a relative move in line 1160 gets a + sign if the move is positive.
PROGRAM 5-2—SCALING FOR PAGES

This program displays squares of ever-increasing size by "flipping" the graphics pages. The graphics are drawn first. A square is drawn with a painted circle inside. The circles are painted a different color on each pair of pages. Mode 1 is used. Therefore, two pages are needed to fill the screen.

The page number is used to determine the relative movement of the upper corner and the size of the square, as well as the radius and paint used for the circle.

![Diagram of a square and circle]

Program 5-2

1000 PCLEAR 8 ← eight pages may be cleared in the program or in the command mode before the program
1010 FOR P = 1 TO 8 STEP 2
1020 PMODE 1,P:PCLS ← clear all pages
1030 NEXT P
1040 PMODE 1,1 ← fix starting point for square
1050 DRAW“BM90,60”
1060 FOR P = 1 TO 8 STEP 2 ← draw squares and circles
1070 PMODE 1,P
1080 A$ = STR$(2+P): R = 6*P: C = (P+3)/2
1090 DRAW “S”+A$+“BM-”+A$+“,”+A$+“R52D52L52U52”
1100 CIRCLE(109,80),R
1110 PAINT(109,80),C,4
1120 NEXT P
1130 FOR P = 1 TO 8 STEP 2 ← display two pages at a time
1140 PMODE 1,P
1150 SCREEN 1,0
1160 FOR W = 1 TO 200: NEXT W
Eight pages are cleared for graphics at line 1000. The pages are cleared at lines 1010-1030. Remember, that mode 1 uses two pages of memory to fill a screen.

Lines 1040 and 1050 put the initial starting position for the square at 90,60 on the screen. This is to the left and above the center.

Lines 1060-1120 draw a square and a painted circle on each of two pages. The square is scaled according to the selected page. The circle is approximately centered within the square and painted a color that is dependent on the selected page. Its radius is also determined by the page.

<table>
<thead>
<tr>
<th>Page</th>
<th>Radius</th>
<th>Color</th>
<th>Scale</th>
<th>Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>2</td>
<td>1/4</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>3</td>
<td>3/4</td>
<td>39</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>4</td>
<td>5/4</td>
<td>65</td>
</tr>
<tr>
<td>7</td>
<td>42</td>
<td>5 or 1</td>
<td>7/4</td>
<td>91</td>
</tr>
</tbody>
</table>

All the drawing is done before the screen is turned on. The loop of lines 1130-1170 display the screens in order. The time delay at line 1160 may be changed.

Line 1180 repeats the display.

**PROGRAM 5-3—PUTTING GREEN**

This program draws a putting green with a blue sky in the background for you golfing fans. The golf hole is placed at the lower left of the putting
green. A small, yellow golf ball rolls in from the right, slows as it approaches the hole, and then drops into the hole.

Program 5-3

1000  PMODE 3,1
1010  PCLS
1020  COLOR 3,1
1030  ' PAINT HOLE
1040  CIRCLE(10,160),6,3
1050  PAINT(10,160),3,3
1060  ' PAINT BLUE SKY
1070  CIRCLE(128,191),100,3,1,.7,.8
1080  CIRCLE(64,191),100,3,1,.7,.8
1090  CIRCLE(192,191),100,3,1,.7,.8
1100  CIRCLE(32,191),100,3,1,.7,.8
1110  CIRCLE(234,191),100,3,1,.7,.8
1120  PAINT(0,0),3,3
1130  ' TURN ON SCREEN AND ACTION
1140  SCREEN 1,0
1150  FOR W = 1 TO 1000:NEXT W
1160  FOR X = 250 TO 150 STEP -10 → fast rolling ball
1170  GOSUB 2000
1180  NEXT X
1190  FOR X = 140 TO 35 STEP -5 → slower rolling ball
1200  GOSUB 2000
1210  NEXT X
1220  FOR X = 30 TO 15 STEP -3 → slowest rolling ball
1230  GOSUB 2000
1240  NEXT X
1250  ' REPAINT HOLE AND DO AGAIN
The sky, putting green, and golf hole are drawn before the screen is displayed. If you want to see the drawing take place, move the SCREEN statement (at line 1140) up to line 1015.

A blue foreground color is selected at line 1020. The hole is drawn and painted at lines 1040 and 1050. Arcs of circles are used at lines 1070-1110 to separate the putting green from the sky, which is painted at line 1120.

Line 1140 turns on the screen, and line 1150 provides a time delay for viewing the scene.

A FOR-NEXT loop (lines 1160-1180) uses a step of -10 to provide highspeed movement as the golf ball enters from the right. The movement is provided by subroutine 2000, which draws a yellow golf ball and a trailing green block to erase the ball's previous position.

The FOR-NEXT loop (lines 1190-1210) slows the golf ball in mid-course, using a step of -5. The speed is slowed still more by the FOR-NEXT loop at lines 1220-1240.

As the ball drops into the hole, lines 1260 and 1270 recolor the hole. Line 1280 sends the computer back for a new ball.
The programs in this section are restricted to the graphics statements introduced in the first five chapters. In addition, programs may use any of the following statements and functions:

- GET
- PUT
- PEEK(65280)
- JOYSTK(0)
- JOYSTK(1)
- JOYSTK(2)
- JOYSTK(3)

**PROGRAM 6-1—BUILDING FLYING OBJECTS**

This program puts several objects on the screen and moves them from left to right. Keep your eyes open for the birds, trucks, and planes. The birds appear in the center portion of the screen, the planes in the upper part, and the trucks near the bottom.

The plane moves faster than the bird, which moves faster than the truck. Which one will reach the right side of the screen first? Look for the one that crashes.
Program 6-1

1000  DIM A(16,12),B(10,10),C(20,16)  ← arrays for truck, bird, and car
1010  FOR P = 1 TO 8 STEP 4  ← four pages used at a time for PMODE 3
1020  PMODE 3,P: PCLS
1030  NEXT P
1040  PMODE 3,1
1050  ' TRUCK
1060  DRAW"BM128,181;U6R12D4R4D2L16"
1070  PAINT(130,178),3,4
1080  PSET(130,184) : PSET(130,183) : PSET(142,184) : PSET(142,183)
1090  ' PLANE
1100  DRAW"BM6,16;U2R2U2R2D2R1OU6R3D6R2D2L2D6L3U6L10D2L2U2L2"
1110  PAINT(8,15),2,4
1120  ' BIRD
1130  CIRCLE(64,90),4,,1,0,.5
1140  PCOPY 1 TO 5: PCOPY 2 TO 6: PCOPY 3 TO 7: PCOPY 4 TO 8
1150  CIRCLE(64,90),4,1,1,0,.5: CIRCLE(64,98),4,4,1,.5,1
1160  G=6: I=6: H=128  ← invert bird on first pages
1170  GET(H,175)-(H+16,187),A,G  ← truck
1180  GET(G,90)-(G+10,100),B,G  ← bird
1190  GET(I,8)-(I+20,24),C,G  ← plane
1200  PCLS
1210  PUT(H+2,175)-(H+18,187),A,PSET
1220  PUT(G+3,90)-(G+13,100),B,PSET
1230  PUT(I+6,8)-(I+26,24),C,PSET
1240  FOR H = 128 TO 228 STEP 4  ← show first pages
1250  SCREEN 1,0  ← PUT and GET last pages
1260  PMODE 3,5
1270  GOSUB 2000
1280  SCREEN 1,0  ← show last pages
1290  PMODE 3,1  ← PUT and GET first pages
1300  J=H+2: K=G+3: L=I+6
1310  GOSUB 3000
This program uses arrays to save portions of the graphics screen. The arrays are dimensioned at line 1000. Array A will contain a truck, array B will contain a bird, and array C will contain a plane.

The graphics screens are cleared at lines 1010-1030. The truck is drawn at lines 1060-1080, the plane at 1100-1110, and the bird at 1130. These figures are copied to the other pages at line 1140. The bird's wings are inverted from upward curl (last pages) to downward curl (first pages) at line 1150. The figures on the first pages are then offset by lines 1170-1230.

The flight of the bird, plane, and truck takes place in the FOR-NEXT loop at lines 1240-1330. One screen is displayed while the GET/PUT subroutine (lines 2000-2070 and 3000-3070) moves the objects on the other screen.

When the objects have moved across the screen, the plane arrives first and crashes and the action stops.

---

PROGRAM 6-2—OLD CARS AND JOYSTICKS

Cars to be abandoned drive in from the right side of the screen. They stop in front of a wall near the left side of the screen. The joysticks control a magnet that will move back and forth across the top of the screen. The magnet can also be lowered and raised by the joysticks. The push buttons on
the joysticks control the activation and the deactivation of the magnet. The object is to lift the cars over the wall and stack up as many as will hide behind the wall.

Joystick left moves the magnet left.

Joystick right moves the magnet right.

Joystick down moves the magnet down.

Joystick up moves the magnet up.

Depressing the push button activates or deactivates the magnet.

Program 6-2

1000 PMODE 3,1
1010 PCLS
1020 DIM C(36, 41), E(36, 41), M(8, 5), N(8, 5)
1030 Z = 152
1040 ' DRAW FIGURES
1050 SCREEN 1,0
1060 LINE(64,55)-(75,181),PSET,BF ← wall
1070 LINE(222,5)-(234,10),PSET,BF ← magnet
1080 LINE(228,10)-(228,25),PSET
1090 LINE(224,20)-(232,25),PSET,BF
ADDITIONAL PROGRAMS

1100 DRAW"BM223,170;R8E4R8F4R8G4L8NE4;"  
     H4L8G4L8NE4D4R4BR4NU4R8NU8;  
     RBR4R4NU4E4U4G4L24"  

1110 CIRCLE(225,178),5: CIRCLE(245,178),5  

1120 ' PAINT CAR  

1130 R = RND(3)+1  

1140 DATA 225,172,237,168,249,172,253,174,249,176,239,       
     176,231,176,221,176,225,178,245,178  

1150 FOR X = 1 TO 10  

1160 READ A,B  

1170 PAINT(A,B),R,4  

1180 NEXT X  

1190 GET(219,166)-(255,182),C  

1200 ' MOVE CAR  

1210 FOR X = 219 TO 99 STEP -10  

1220 PUT(X,166)-(X+36,182),E  
     ← erase old car  

1230 PUT(X-15,166)-(X+21,182),C  
     ← PUT new car  

1240 FOR W = 1 TO 10: NEXT W  

1250 NEXT X  

1260 H=JOYSTK(0) : V=JOYSTK(1)  
     ← joystick left  

1270 IF H<16 THEN GOSUB 3000 ELSE GOTO 1260  

1280 H=JOYSTK(0) : V=JOYSTK(1)  
     ← joystick down  

1290 IF V>47 THEN GOSUB 4000 ELSE GOTO 1280  

1300 M = PEEK(65280)  
     ← push button pressed and joystick up  

1310 IF M = 127 OR M = 255 THEN 1300  

1320 H=JOYSTK(0) : V=JOYSTK(1)  

1330 IF V<16 THEN GOSUB 5000 ELSE 1320  

1340 H=JOYSTK(0) : V=JOYSTK(1)  
     ← joystick left  

1350 IF H<16 THEN GOSUB 6000 ELSE GOTO 1340  

1360 H=JOYSTK(0) : V=JOYSTK(1)  
     ← joystick down  

1370 IF V>47 THEN GOSUB 7000 ELSE 1360  

1380 M = PEEK(65280)  
     ← push button pressed  

1390 IF M = 127 OR M = 255 THEN 1380  
     and joystick up  

1400 H=JOYSTK(0) : V=JOYSTK(1)  

1410 IF V<16 THEN GOSUB 8000 ELSE GOTO 1400  

1420 Z = Z-16  
     ← go again  

1430 RESTORE  

1440 GOTO 1100  

3000 ' MOVE MAGNET LEFT
3010  FOR X = 222 TO 98 STEP-8
3020  GET(X,5)-(X+36,46),C
3030  PUT(X,5)-(X+36,46),E
3040  PUT(X-8,5)-(X+28,46),C
3050  FOR W = 1 TO 10: NEXT W
3060  NEXT X
3070  RETURN

4000  ' LOWER MAGNET
4010  FOR Y = 20 TO 150 STEP 10
4020  GET(98,20)-(106,25),M
4030  PUT(98,Y)-(106,Y+5),N
4040  LINE(101,20)-(101,Y+10),PSET
4050  PUT(98,Y+10)-(106,Y+15),M
4060  FOR W = 1 TO 10: NEXT W
4070  NEXT Y
4080  RETURN

5000  ' LIFT CAR
5010  FOR Y = 154 TO 22 STEP-8
5020  GET(85,Y)-(121,Y+36),C
5030  PUT(85,Y)-(121,Y+36),E
5040  PUT(85,Y-8)-(121,Y+28),C
5050  FOR W = 1 TO 10: NEXT W
5060  NEXT Y
5070  RETURN

6000  ' MOVE LEFT
6010  FOR X = 85 TO 21 STEP-8
6020  GET(X,5)-(X+36,46),C
6030  PUT(X,5)-(X+36,46),E
6040  PUT(X-8,5)-(X+28,46),C
6050  FOR W = 1 TO 10: NEXT W
6060  NEXT X
6070  RETURN

7000  ' LOWER CAR
7010  FOR Y = 20 TO Z STEP 8
7020  GET(18,Y)-(54,Y+26),C
7030  PUT(18,Y)-(54,Y+26),E
7040  LINE(28,20)-(28,Y+10),PSET
7050  PUT(18,Y+8)-(54,Y+34),C
7060  FOR W = 1 TO 10: NEXT W
7070  NEXT Y
7080  RETURN

8000  ' LIFT MAGNET AND MOVE TO BEGINNING
After the mode is selected and the screen cleared at lines 1000 and 1010, the arrays used for animation are dimensioned at line 1020. The variable Z, in line 1030, is used to determine the abandoned car's place on the discard stack.

The screen is turned on at line 1050. The wall, magnet, and cars are drawn by lines 1060-1110. The car is painted a random color by the FOR-NEXT loop at lines 1150-1180. Array C is used at line 1190 to save the graphics of the car in memory.

Lines 1210-1250 move the car from right to left, stopping in front of the wall. The computer waits for a movement of the joystick (full left) before moving the magnet into position above the car (accomplished by subroutine 3000).

Then the computer waits for the joystick to be moved downward before lowering the magnet to the car (lines 1280-1290 and subroutine 4000). It then waits for the push button to be pressed and the joystick to be moved upward before raising the magnet and the car (lines 1300-1330 and subroutine 5000).

Another wait is made (lines 1340-1350) for the joystick to be moved left before moving the car and magnet beyond the wall at subroutine 6000. Then a wait at lines 1360-1370 lets the joystick be moved downward to lower the car to the top of any previously stacked cars (subroutine 7000).

Lines 1380-1410 wait for the push button to release the car and the joystick to be moved up to raise the magnet (subroutine 8000). Line 1420 adjusts Z for the top of the car stack. Line 1430 restores the data list for painting another car. Line 1440 returns the computer to get a new car.

The rest of the program consists of subroutines used to simplify the main program. You do not have complete control (with the joysticks) over the movements. The joystick control is used only to initiate automatic movements. You might want to try altering the program to provide complete control by the joysticks.
The programs in this section may use any of the statements and functions introduced in the first six chapters. In addition, programs are enhanced by sound through the use of the following:

- **SOUND n,d**
  - duration (1-255)
  - note (1-255)

- **PLAY "sound string"**
  - a string of option commands explained in Chapter 7

### PROGRAM 7-1—TWO VERSIONS OF A JIG

This program compares the use of the SOUND and the PLAY command. In the first part of the program, the jig is played using SOUND statements. In the last part of the program, the jig is played using the PLAY command.

Program 7-1

```
1000 ' USING SOUND
1010 FOR X = 1 TO 30
1020 READ N,D
1030 SOUND N,D
1040 NEXT X
1050 DATA 108,3,108,3,78,3,32,3,58,3,1,3,
1060 DATA 58,6,108,3,108,3,78,3,32,3,58,3,89,3,
      78,3,32,3,58,3,89,3,78,3,32,3
1070 DATA 58,3,108,3,32,6
```
ADDITIONAL PROGRAMS

1080 FOR Z = 1 TO 2
1090 DATA 32,3,58,3,89,3,58,3,78,3,108,3,
78,3,58,3,89,3,89,3,58,3,78,3,108,3
1100 DATA 32,3,58,3,89,3,58,3,78,3,108,3,
78,3,58,3,78,3,58,3,1,3
1110 DATA 32,3,58,3,89,3,58,3,78,3,108,3,
78,3,58,3,89,3,89,3,58,3,78,3,108,3
1120 DATA 32,3,58,3,89,3,58,3,78,3,108,3,
78,3,58,3,78,3,58,3,1,3
1130 FOR R = 1 TO 24
1140 READ N,D
1150 SOUND N,D
1160 NEXT R
1170 NEXT Z
1300 ' USING PLAY
1310 A$ = "O2BG": B$ = "AO3C": C$ = "O2ABO3DP4O2"
1320 PLAY"T603L4DDXAS;ACBG03DD"
1330 PLAY"XAS;XB$;O2L2AO3L4DD"
1340 PLAY"XAS;XB$;O2BAO3C"
1350 PLAY"XAS;AO3D02L2G"
1360 FOR X = 1 TO 2
1370 PLAY"P4GXBS;O2P4ABO3DP4O2B"
1380 PLAY"XB$;CXC$:G"
1390 PLAY"XB$;P4XC$:B"
1400 PLAY"ABP4ADP4"
1410 NEXT X
1500 END

You may notice a few notes that the SOUND portion of the program could not reach. The note values for SOUND do not go all the way down to a full octave 2. The PLAY section is much shorter, as the option commands are quite versatile.

--- PROGRAM 7-2—OTHER SOUNDS ---

This program is for pure experimentation. It uses the PLAY statement and lets you build strings of sounds. You can build as many PLAY statements as you wish. Each PLAY statement will hold the volume, tempo, octave, note length, and 10 notes.
Program 7-2

1000 CLS: CLEAR 500
1010 INPUT "NUMBER OF PLAY COMMANDS"; C
1020 DIM T$(C); DIM N(10)
1030 FOR Z = 1 TO C
1040 INPUT "NUMBER OF NOTES"; N(Z)
1050 NEXT Z
1060 FOR Z = 1 TO C
1070 INPUT "VOLUME (1-31)"; V$
1080 INPUT "TEMPO (1-255)"; T$
1090 INPUT "OCTAVE (1-5)"; O$
1100 INPUT "NOTE LENGTH (1-255)"; L$
1110 P$(Z) = " "
1120 FOR X = 1 TO N(Z)
1130 INPUT "NOTE (A-G OR PX)"; N$(X)
1140 PR(Z) = P$(Z)+N$(X)
1150 NEXT X
1160 T$(Z) = "V"+V$+"T"+T$+"O"+O$+"L"+L$+P$(Z)
1170 NEXT Z
1200 ' PLAY THE COMMANDS
1210 FOR Z = 1 TO C
1220 PLAY T$(Z)
1230 NEXT Z

Line 1000 clears the screen and reserves some string space. If you need more string space, change the value following CLEAR in line 1000. The number of PLAY statements that you want to build is input at line 1010. A string array to hold the commands is dimensioned at line 1020.

The FOR-NEXT loop (lines 1060-1170) lets you build the strings. The number of notes in a given PLAY statement is input at line 1040. The array that holds the notes is dimensioned at line 1020. Controlling parameters are input at lines 1070-1100.

The FOR-NEXT loop at lines 1120-1150 builds up the string of notes. Line 1160 puts the options together with the notes for a complete PLAY statement.

Lines 1200-1230 play the complete set of finished PLAY commands.
The programs in this section may include any of the statements introduced in the first seven chapters. In addition, the following statements and functions that were discussed in Chapter 8 are used:

- \text{TIMER} = n \quad n = 0-65535; \text{sets the timer}
- \text{T} = \text{TIMER} \quad \text{assigns TIMER's value to the variable T}

\textbf{PROGRAM 8-1—DISPLAY 23 LINES OF TEXT}

This program uses the revised text character set for graphics subroutine developed in Chapter 8. It uses a scale factor of three-fourths (S=3) to reduce the size of the letters so that more will fit on the screen. You should be able to put up to 28 characters on a line.

The program lets you type in a screen full of characters (23 lines), then wipes the screen clear in preparation for another screenfull. We added a comma and period to the character set.

Program 8-1

```
1000 CLEAR 500: DIM L$(29): GOSUB 10000
1010 PMODE 4,1
1020 PCLS
1030 SCREEN 1,0
1040 X$ = 2
1050 FOR L = 1 TO 23
1060 Y = (L-1)*8+8
```
The character array is dimensioned, string space cleared, and variables assigned at line 1000. Lines 1010-1030 set up the mode and screen.

Since we want to start in the upper left corner, the X position for the first letter is set to two at line 1040. Twenty-three lines are provided by the FOR-NEXT loop at lines 1050-1180. The Y position for each line is set by line 1060 (starting at eight and going down eight for each new line). This value is put in string form by line 1070. Line 1080 sets full scale and makes a blank move to the correct starting position for each line.

The FOR-NEXT loop at lines 1090-1180 provides for up to 28 characters per line. Line 1100 waits for a key to be pressed. Line 1110 checks to see if the ENTER key was pressed. If it is, an exit is made from the loop (to line 1180), and a new line is set up. Line 1120 calculates the correct subscript for the letters. Line 1130 rejects keystrokes that are not in the subroutine. Lines 1140-1160 make the necessary selections for a comma, period, or space. Line 1170 DRAWs the character selected at three-fourths scale. Line 1180 sends the computer back for a new key. If the line is full (C=28), a new line is called for by line 1190.

Don't forget to load the character subroutine at 10000 and add:

```
10280  L$(28) = "D2G1BE1BU2BR12"
10290  L$(29) = "D1BU1R1BR11"
10300  RETURN
```

---

**PROGRAM 8-2—TIMED TYPING PRACTICE**

A good part of a programmer's time is spent modifying programs. Let's modify program 8-1 to calculate how fast the computer can draw text.
characters (or how fast you can type). Can you outpace the computer’s ability to DRAW the characters?
To modify the program you need to
1. set the timer to zero for each new line
2. check the time to type each line
3. add the times for each line
4. calculate the words per minute for the total time
   (we’ll use five characters per word)

These additions and changes should be sufficient:

Change 1000 CLEAR 500: DIM L$(29),T(23) ← time array added
Add 1035 CT=0; T=0
Add 1055 TIMER = 0
Change 1110 IF A$ = CHR$(13) THEN 1185
Add 1185 CT= CT+C; T(L)=TIMER; TIMER=0
Add 1187 T = T+T(L)
Change 1200 TT = T/360 ← total time in minutes
Add 1210 W = CT/5 ← words
Add 1220 S = W/TT ← speed in wpm
Add 1230 CLS: PRINT"SPEED =";S;"WPM"
Add 1240 PRINT"PRESS ANY KEY TO TYPE AGAIN"
Add 1250 A$=INKEY$: IF A$ = " " THEN 1250
Add 1260 CLS: GOTO 1010

Program 8-2

1000 CLEAR 500: DIM L$(29),T(23): GOSUB 10000
1010 PMODE 4,1
1020 PCLS
1030 SCREEN 1,0 ← character count and time
1035 CT=0; T=0 set to zero
1040 X$="2"
1050 FOR L=1 TO 23
1055 TIMER=0 ← start timing
1060 Y=(L-1)*8+8 ← line control (Y)
1070 Y$=STR$(Y) ← start line here
1080 DRAW"S4BM"+X$+","+Y$
1090 FOR C=1 TO 28
1100 A$=INKEY$: IF A$=" " THEN 1100
1110 IF A$=CHR$(13) THEN 1185 ← ENTER key
ADDITIONAL PROGRAMS

1120 N=ASC(A$)-64
← subscript (N)
1130 IF (N<1 and N>-20 and N>-18 AND N>-32) OR N>90 THEN 1100
1140 IF N=-20 THEN N=28
1150 IF N=-18 THEN N=29
1160 IF N=-32 THEN N=27
1170 DRAW"$3"+L$(N)
← "type" character
1180 NEXT C
1185 CT=CT+C:T(L)=TIMER:TIMER=0
← add word count
1187 T=T+T(L)
← for line, save time, restart timer
1190 NEXT L
1200 TT=T/3600
← total time (≈ minutes)
1210 W=CT/5
← total words
1220 S=W/TT
← speed
1230 CLS:PRINT"SPEED =";S;" WPM"
1240 PRINT"PRESS ANY KEY TO TYPE AGAIN"
1250 A$=INKEY$: IF A$=" " THEN 1250
1260 CLS:GOTO 1010

10000 ' SUBROUTINE FOR DRAWING LETTERS
10010 L$(1)="U8R8D4L8BR8D4BR4"
10020 L$(2)="U8R8D4L8BR8D4L8BR12"
← modified B
10030 L$(3)="U8R8BD8L8BR12"
10040 L$(4)="U8R8D8L8BR12"
← modified D
10050 L$(5)="U8R8BD4L8BD4R8BR4"
10060 L$(6)="U8R8BD4L8BD4BR12"
10070 L$(7)="U8R8BD4L4BR4D4L8BR12"
10080 L$(8)="U8BR8D8BU4L8BD4BR12"
10090 L$(9)="BU8R8BL4D8BL4R8BR4"
10100 L$(10)="U4BU4BR8D8L8BR12"
10110 L$(11)="U8BR8G4L4BR4F4BR4"
10120 L$(12)="U8BD8R8BR4"
10130 L$(13)="U8F4E4D8BR4"
10140 L$(14)="U8F8U8BD8BR4"
10150 L$(15)="U8R6D8L6BR12"
← modified O
10160 L$(16)="U8R8D4L8BR4D4BR12"
10170 L$(17)="U8R8D8H4BG4R8BR4"
10180 L$(18)="U8R8D4L8BR4F4BR4"
10190 L$(19)="BU4U4R8BD4L8BR8D4L8BR12"
10200 L$(20)="BU8R8BL4D8BR8"
10210 L$(21)="U8BR8D8L8BR12"
10220 L$(22)="BU8D4F4E4U4BD8BR4"
10230 L$(23)="U8BR8D8H4G4BR12"
10240 L$(24)="E8BL8F8BR4"
10250 L$(25)="BU8F4E4BG4D4BR8"
This program works in much the same way as program 7-2 with the exception of the timing functions and the tabulation of characters typed.

The dimensioning of the time array T(L) is added to line 1010. Line 1035 initializes the word count (CT) and the time (T). The timer is started at line 1055.

When a line is full (or terminated by the ENTER key), the word count and the time for the typed line are saved and the timer is reset at line 1185. Line 1187 calculates the total current time.

When the page is full, line 1200 calculates the time in approximate minutes. The number of words typed is calculated at line 1210, and the speed at line 1220. The speed is printed at line 1230.

Lines 1240-1260 take care of starting another page if requested.

Notice that the letters D, B, and O were modified in the character array. This was done to avoid some fractions that resulted from scaling by three-fourths.
The USR Function

Programs in this section may use any of the statements discussed in the first eight chapters. In addition, programs will contain some of the following statements and functions introduced in Chapter 9.

DEFUSR
USR
CL:ER
HEX$
&H

These last two programs demonstrate the beginnings of a mini machine language monitor written in BASIC. The development of the monitor will be expanded in a future Reston book by Kurt Inman and myself, *6809 Assembly Language for the TRS-80 Color Computer*.

The basis for these two programs is derived from the article "6809 Machine Code" by Bill Sias in the July/August 1981 issue of *Color Computer News*, a magazine produced by REMarkable Software, P.O. Box 1192, Muskegon, MI 49443. This magazine is a must for TRS-80 Color Computer users.

PROGRAM 9-1—A BEGINNING MONITOR

This program is not a complete monitor. It only lets you enter hexadecimal machine codes, beginning at decimal address 16001, and examine and correct entries. A section to execute the program will be added in program 9-2.

The program asks whether you want to enter or check previously entered values. First, a program is entered by typing ENTER. The CLEAR command is used to reserve the number of entries to be made beginning at memory location 16001. You are allowed to input this number.

Each address, with its contents, is printed (one pair at a time) and a request is made for the input of a hexadecimal machine code for the given address. The input is converted to decimal and poked into memory. This repeats until all entries have been made.

The screen is cleared and you again are asked whether you want to enter or check codes. After entering your program, you would type CHECK.
Addresses with the codes that you entered are displayed 12 at a time. When you have viewed the complete program, corrections may be made. Just answer the questions on the screen and follow directions.

Program 9-1

1000  CLS
1010  INPUT "ENTER OR CHECK"; E$
1020  IF LEFT$(E$,2) = "EN" THEN 2000  ← to enter
1030  IF LEFT$(E$,2) = "CH" THEN 3000  ← to check
1040  PRINT "UNIDENTIFIED COMMAND"
1050  FOR TM = 1 TO 200: NEXT TM
1060  GOTO 1010
2000  INPUT "NUMBER OF ENTRIES"; EN
2010  POKE 16001,EN  number of entries, save
2020  CLEAR EN,16000  memory beyond 16000
2030  EN = PEEK(16001)
2040  FOR AD = 16001 TO 16000+EN
2050    D = PEEK(AD)
2060    A$ = HEX$(D)
2070    PRINT AD;A$;: INPUT H$  ← address, old value;
2080    BYTE = VAL("&H"+H$)  input new value
2090    POKE AD,BYTE
2100  NEXT AD
2110  GOTO 1000
3000  CLS
3010  A1 = 1: B1 = 12
3020  FOR AD = A1 TO B1
3030    N = PEEK(AD+16000)
3040    BY$ = HEX$(N)
3050    PRINT AD:BY$  ← print decimal address, hex code
3060  NEXT AD
3070  A1 = A1 + 12: B1 = B1+12
3080  INPUT "MORE? – YES OR NO";Q$
3090  IF LEFT$(Q$,1) = "Y" THEN CLS: GOTO 3020
3100  CLS: INPUT "CORRECTIONS? – YES OR NO";Q$
3110  IF LEFT$(Q$,1) = "N" THEN 4000
3120  INPUT "ADDRESS"; AD
3130  PRINT AD:HEX$(PEEK(AD))
3140  INPUT "NEW VALUE IN HEX";H$
3150  BYTE = VAL("&H"+H$)
3160  POKE AD, BYTE
3170  INPUT "MORE CORRECTIONS? – YES OR NO";Q$
3180  IF LEFT$(Q$,1) = "Y" THEN 3120
3190  INPUT "CHECK ENTRIES AGAIN? – YES OR NO";QS$
3200  IF LEFT$(QS,1) = "Y" THEN 3000
4000  END

Lines 1000-1060 allow you to select the ENTER or the CHECK part of the program. Lines 1020 and 1030 check your selection and go to the appropriate place in the program. Line 1040 assures that one of the two choices is made correctly.

Lines 2000-2090 provide for the entry of a program. Line 2000 checks for the number of entries (EN). Line 2020 saves that number of entries starting 1 place beyond 16000.

The FOR-NEXT loop at lines 2040-2100 shows an address and its contents and asks for an entry. Enter a two-digit hex code, say 8E. The next address and contents are then displayed. Continue entering your program.

When all values have been entered, the screen clears, and you are returned to the beginning for another choice.

Type: CHECK

The program then goes to line 3000. Twelve addresses and the hex codes contained are displayed (lines 3020-3060) along with the question, “MORE? – YES OR NO?” (line 3080).

If there are more and you type YES, 12 more pairs will be displayed with the same question.

Note. Write down any errors and their addresses. Corrections cannot be made until all entries are checked.

If you type NO, the computer will clear the display and ask if there are any corrections to be

```
ENTER OR CHECK?

ENTER OR CHECK? ENTER NUMBER OF ENTRIES?

ENTER OR CHECK? ENTER NUMBER OF ENTRIES? 10
16001 8E
16002 3E
16003 8E
16004 A6
16005 80
16006 27
16007 5
16008 BD
16009 A3
16010 A
16011 20
16012 F7
MORE? – YES OR NO?
```
If you have no corrections, type NO. The program will end.

If you type YES for corrections, the computer will ask you for the address and the new value (in hex). This process goes on until all corrections have been made. Then it will ask if you want to look at the entries again for a final check. If you do, the computer returns to line 3000 and repeats. If you do not want to check again, the program will end when you type NO.

Use the machine codes of Figure 9-2 in Chapter 9 to test the mini monitor.

PROGRAM 9-2—ADDING EXECUTE TO THE MONITOR

This program adds an execution section to the mini monitor so that you can run programs that you have entered and checked. We will keep this section simple although you might want to add some things of your own.

Program 9-2 (to modify program 9-1)

<table>
<thead>
<tr>
<th>Change/Line</th>
<th>Original Code</th>
<th>New Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1010</td>
<td>INPUT &quot;ENTER, CHECK, OR EXECUTE&quot;;E$</td>
<td></td>
</tr>
<tr>
<td>1035</td>
<td>IF LEFT$(E$,2) = &quot;EX&quot; THEN 4000</td>
<td></td>
</tr>
<tr>
<td>3110</td>
<td>IF LEFT$(Q$,1) = &quot;N&quot; THEN 1010</td>
<td></td>
</tr>
<tr>
<td>3195</td>
<td>GOTO 1000</td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>INPUT &quot;ARGUMENT PASSED&quot;;AR</td>
<td></td>
</tr>
<tr>
<td>4010</td>
<td>CLS: DEFUSR0 = 16001</td>
<td>define USR</td>
</tr>
<tr>
<td>4020</td>
<td>RV = USR0(AR)</td>
<td>call USR</td>
</tr>
<tr>
<td>4030</td>
<td>PRINT &quot;RETURNED VALUE&quot;;RV</td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>END</td>
<td></td>
</tr>
</tbody>
</table>

Line 1010 is altered and line 1035 is added to provide access to the execution section. Line 3110 is changed to return you to line 1010 after all corrections have been made.

Line 4000 allows you to pass an argument if you wish (more about this in the Reston assembly language book). Input a zero if you are not passing an argument.
Line 4010 defines the entry address of the machine language program. You could use an INPUT statement to input a different entry address.

Line 4020 "calls" the machine language program for execution. If a value is to be returned from the machine language program, it would be printed by line 4030.

The program ends at line 5000. You could change the GOTO to return to the monitor if you desired.
Appendix A

Color Computer Connections

Back of Color Computer

Back of Television Set

TV channel selector
Tape Jack
RS-232-C Jack
Joysticks right left
Power Switch

TV cable

RESET

From TV antenna (or cable)

VHF
UHF

To computer

287
To computer

Black to EAR

Big gray to AUX

Little gray to small hole

Side of Cassette Recorder
## Memory Map for 16K RAM—Extended Color Basic Computer

<table>
<thead>
<tr>
<th>Decimal Address</th>
<th>Hexadecimal Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1023</td>
<td>0-3FF</td>
<td>System Use</td>
</tr>
<tr>
<td>0-255</td>
<td>0-0FF</td>
<td>Direct Page RAM</td>
</tr>
<tr>
<td>256-1023</td>
<td>100-3FF</td>
<td>Extended Page RAM</td>
</tr>
<tr>
<td>1024-1535</td>
<td>400-5FF</td>
<td>Text Screen</td>
</tr>
<tr>
<td>1536-13823</td>
<td>600-35FF</td>
<td>Graphic Screen</td>
</tr>
<tr>
<td>1536-3071</td>
<td>600-BFF</td>
<td>Page 1</td>
</tr>
<tr>
<td>3072-4607</td>
<td>C00-11FF</td>
<td>Page 2</td>
</tr>
<tr>
<td>4608-6143</td>
<td>1200-17FF</td>
<td>Page 3</td>
</tr>
<tr>
<td>6144-7679</td>
<td>1800-1DFF</td>
<td>Page 4</td>
</tr>
<tr>
<td>7680-9215</td>
<td>1E00-23FF</td>
<td>Page 5</td>
</tr>
<tr>
<td>9216-10751</td>
<td>2400-29FF</td>
<td>Page 6</td>
</tr>
<tr>
<td>10752-12287</td>
<td>2A00-2FFF</td>
<td>Page 7</td>
</tr>
<tr>
<td>12288-13823</td>
<td>3000-35FF</td>
<td>Page 8</td>
</tr>
<tr>
<td>13824-16383</td>
<td>3600-3FFF</td>
<td>Program and Variables</td>
</tr>
<tr>
<td>32768-40959</td>
<td>8000-9FFF</td>
<td>Extended Color BASIC</td>
</tr>
<tr>
<td>49152-65279</td>
<td>C000-FEFP</td>
<td>Cartridge</td>
</tr>
<tr>
<td>65280-65535</td>
<td>FF00-FFFF</td>
<td>Input/Output</td>
</tr>
</tbody>
</table>
Memory Map for 16K RAM
Extended Color Basic
Computer

<table>
<thead>
<tr>
<th>Segment</th>
<th>Memory Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Area</td>
<td></td>
</tr>
<tr>
<td>System Area</td>
<td></td>
</tr>
<tr>
<td>Option Code</td>
<td>A000</td>
</tr>
<tr>
<td>Table</td>
<td>0000</td>
</tr>
<tr>
<td>Test</td>
<td>0010</td>
</tr>
<tr>
<td>Code</td>
<td>0020</td>
</tr>
<tr>
<td>Data</td>
<td>0030</td>
</tr>
<tr>
<td>Test</td>
<td>0040</td>
</tr>
<tr>
<td>Data</td>
<td>0050</td>
</tr>
<tr>
<td>Setup</td>
<td>0060</td>
</tr>
<tr>
<td>Setup</td>
<td>0070</td>
</tr>
<tr>
<td>Device Area</td>
<td></td>
</tr>
<tr>
<td>Printer</td>
<td>0120</td>
</tr>
<tr>
<td>Interface</td>
<td>0130</td>
</tr>
<tr>
<td>Diskette Drive</td>
<td>0140</td>
</tr>
<tr>
<td>Display</td>
<td>0150</td>
</tr>
</tbody>
</table>

Appendix B

EXTRANEOUS TABLES

Appendix C

EXTRANEOUS TABLES
Appendix C

Graphics Screen

EXTENDED COLOR BASIC COLORS

<table>
<thead>
<tr>
<th>Code</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Black</td>
</tr>
<tr>
<td>1</td>
<td>Green</td>
</tr>
<tr>
<td>2</td>
<td>Yellow</td>
</tr>
<tr>
<td>3</td>
<td>Blue</td>
</tr>
<tr>
<td>4</td>
<td>Red</td>
</tr>
<tr>
<td>5</td>
<td>Buff</td>
</tr>
<tr>
<td>6</td>
<td>Cyan</td>
</tr>
<tr>
<td>7</td>
<td>Magenta</td>
</tr>
<tr>
<td>8</td>
<td>Orange</td>
</tr>
</tbody>
</table>

GRAPHICS SCREEN RESOLUTION

<table>
<thead>
<tr>
<th>PMODE #</th>
<th>Grid Size</th>
<th>Color-Mode</th>
<th>Pages Used</th>
<th>Point Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>256 x 192</td>
<td>two-color</td>
<td>4</td>
<td>□</td>
</tr>
<tr>
<td>3</td>
<td>128 x 192</td>
<td>four-color</td>
<td>4</td>
<td>□ □</td>
</tr>
<tr>
<td>2</td>
<td>128 x 192</td>
<td>two-color</td>
<td>2</td>
<td>□ □ □</td>
</tr>
<tr>
<td>1</td>
<td>128 x 96</td>
<td>four-color</td>
<td>2</td>
<td>□ □ □ □</td>
</tr>
<tr>
<td>0</td>
<td>128 x 96</td>
<td>two-color</td>
<td>1</td>
<td>□ □ □ □ □</td>
</tr>
</tbody>
</table>

291
### COLOR-SETS

<table>
<thead>
<tr>
<th>PMODE #</th>
<th>Color-Set (see SCREEN)</th>
<th>Two-Color Combination</th>
<th>Four-Color Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>Black/Green</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Black/Buff</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td>Green/Yellow/Blue/Red</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>Buff/Cyan/Magenta/Orange</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>Black/Green</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Black/Buff</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>Green/Yellow/Blue/Red</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>Buff/Cyan/Magenta/Orange</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Black/Green</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Black/Buff</td>
<td></td>
</tr>
</tbody>
</table>
### SOUND and PLAY Tables

#### SOUND

<table>
<thead>
<tr>
<th>Tone Value</th>
<th>Note</th>
<th>Tone Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>F</td>
<td>197</td>
<td>F</td>
</tr>
<tr>
<td>19</td>
<td>G</td>
<td>200</td>
<td>G</td>
</tr>
<tr>
<td>32</td>
<td>A</td>
<td>204</td>
<td>A</td>
</tr>
<tr>
<td>45</td>
<td>B</td>
<td>207</td>
<td>B</td>
</tr>
<tr>
<td>58</td>
<td>C</td>
<td>210</td>
<td>C</td>
</tr>
<tr>
<td>69</td>
<td>D</td>
<td>213</td>
<td>D</td>
</tr>
<tr>
<td>78</td>
<td>E</td>
<td>216</td>
<td>E</td>
</tr>
<tr>
<td>89</td>
<td>F</td>
<td>218</td>
<td>F</td>
</tr>
<tr>
<td>99</td>
<td>G</td>
<td>221</td>
<td>G</td>
</tr>
<tr>
<td>108</td>
<td>A</td>
<td>223</td>
<td>A</td>
</tr>
<tr>
<td>117</td>
<td>B</td>
<td>225</td>
<td>B</td>
</tr>
<tr>
<td>125</td>
<td>C</td>
<td>227</td>
<td>C</td>
</tr>
<tr>
<td>133</td>
<td>D</td>
<td>229</td>
<td>D</td>
</tr>
<tr>
<td>140</td>
<td>E</td>
<td>231</td>
<td>E</td>
</tr>
<tr>
<td>147</td>
<td>F</td>
<td>232</td>
<td>F</td>
</tr>
<tr>
<td>153</td>
<td>G</td>
<td>234</td>
<td>G</td>
</tr>
<tr>
<td>159</td>
<td>A</td>
<td>236</td>
<td>A</td>
</tr>
<tr>
<td>165</td>
<td>B</td>
<td>237</td>
<td>B</td>
</tr>
<tr>
<td>170</td>
<td>C</td>
<td>238</td>
<td>C</td>
</tr>
<tr>
<td>176</td>
<td>D</td>
<td>239</td>
<td>D</td>
</tr>
<tr>
<td>180</td>
<td>E</td>
<td>241</td>
<td>E</td>
</tr>
<tr>
<td>185</td>
<td></td>
<td>242</td>
<td></td>
</tr>
<tr>
<td>189</td>
<td></td>
<td>243</td>
<td></td>
</tr>
<tr>
<td>193</td>
<td></td>
<td>244</td>
<td></td>
</tr>
</tbody>
</table>
## PLAY

<table>
<thead>
<tr>
<th>Number</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>C#/D -</td>
</tr>
<tr>
<td>3</td>
<td>D/E -</td>
</tr>
<tr>
<td>4</td>
<td>E - /D#</td>
</tr>
<tr>
<td>5</td>
<td>E/F -</td>
</tr>
<tr>
<td>6</td>
<td>F/E#</td>
</tr>
<tr>
<td>7</td>
<td>F#/G -</td>
</tr>
<tr>
<td>8</td>
<td>G</td>
</tr>
<tr>
<td>9</td>
<td>G#/A -</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>A#/B -</td>
</tr>
<tr>
<td>12</td>
<td>B</td>
</tr>
</tbody>
</table>

PLAY does not recognize the notation B# or C-. Use the numbers 1-12, respectively, or use C for B# and B for C- to avoid an RFC ERROR.
## Appendix E

### ASCII Character Codes

<table>
<thead>
<tr>
<th>Key</th>
<th>Hex #</th>
<th>Decimal #</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unshifted</td>
<td>Shifted</td>
</tr>
<tr>
<td>(BREAK)</td>
<td>03</td>
<td>03</td>
</tr>
<tr>
<td>(CLEAR)</td>
<td>0C</td>
<td>5C</td>
</tr>
<tr>
<td>(ENTER)</td>
<td>0D</td>
<td>0D</td>
</tr>
<tr>
<td>(SPACEBAR)</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>-</td>
<td>21</td>
</tr>
<tr>
<td>&quot;</td>
<td>-</td>
<td>22</td>
</tr>
<tr>
<td>#</td>
<td>-</td>
<td>23</td>
</tr>
<tr>
<td>$</td>
<td>-</td>
<td>24</td>
</tr>
<tr>
<td>%</td>
<td>-</td>
<td>25</td>
</tr>
<tr>
<td>&amp;</td>
<td>-</td>
<td>26</td>
</tr>
<tr>
<td>,</td>
<td>-</td>
<td>27</td>
</tr>
<tr>
<td>(</td>
<td>-</td>
<td>28</td>
</tr>
<tr>
<td>)</td>
<td>-</td>
<td>29</td>
</tr>
<tr>
<td>*</td>
<td>-</td>
<td>2A</td>
</tr>
<tr>
<td>+</td>
<td>-</td>
<td>2B</td>
</tr>
<tr>
<td>,</td>
<td>2C</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>2D</td>
<td>-</td>
</tr>
<tr>
<td>.</td>
<td>2E</td>
<td>-</td>
</tr>
<tr>
<td>/</td>
<td>2F</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>30</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>31</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>34</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>35</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>37</td>
<td>-</td>
</tr>
</tbody>
</table>
### APPENDIX E  ASCII CHARACTER CODES

<table>
<thead>
<tr>
<th>Key</th>
<th>Hex #</th>
<th>Decimal #</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unshifted</td>
<td>Shifted</td>
</tr>
<tr>
<td>8</td>
<td>38</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>39</td>
<td>-</td>
</tr>
<tr>
<td>:;</td>
<td>3A</td>
<td>-</td>
</tr>
<tr>
<td>;</td>
<td>3B</td>
<td>-</td>
</tr>
<tr>
<td>&lt;</td>
<td>3C</td>
<td>-</td>
</tr>
<tr>
<td>=</td>
<td>3D</td>
<td>-</td>
</tr>
<tr>
<td>&gt;</td>
<td>3E</td>
<td>-</td>
</tr>
<tr>
<td>?</td>
<td>3F</td>
<td>-</td>
</tr>
<tr>
<td>@</td>
<td>40</td>
<td>13</td>
</tr>
<tr>
<td>A</td>
<td>61</td>
<td>41</td>
</tr>
<tr>
<td>B</td>
<td>62</td>
<td>42</td>
</tr>
<tr>
<td>C</td>
<td>63</td>
<td>43</td>
</tr>
<tr>
<td>D</td>
<td>64</td>
<td>44</td>
</tr>
<tr>
<td>E</td>
<td>65</td>
<td>45</td>
</tr>
<tr>
<td>F</td>
<td>66</td>
<td>46</td>
</tr>
<tr>
<td>G</td>
<td>67</td>
<td>47</td>
</tr>
<tr>
<td>H</td>
<td>68</td>
<td>48</td>
</tr>
<tr>
<td>I</td>
<td>69</td>
<td>49</td>
</tr>
<tr>
<td>J</td>
<td>6A</td>
<td>4A</td>
</tr>
<tr>
<td>K</td>
<td>6B</td>
<td>4B</td>
</tr>
<tr>
<td>L</td>
<td>6C</td>
<td>4C</td>
</tr>
<tr>
<td>M</td>
<td>6D</td>
<td>4D</td>
</tr>
<tr>
<td>N</td>
<td>6E</td>
<td>4E</td>
</tr>
<tr>
<td>O</td>
<td>6F</td>
<td>4F</td>
</tr>
<tr>
<td>P</td>
<td>70</td>
<td>50</td>
</tr>
<tr>
<td>Q</td>
<td>71</td>
<td>51</td>
</tr>
<tr>
<td>R</td>
<td>72</td>
<td>52</td>
</tr>
<tr>
<td>S</td>
<td>73</td>
<td>53</td>
</tr>
<tr>
<td>T</td>
<td>74</td>
<td>54</td>
</tr>
<tr>
<td>U</td>
<td>75</td>
<td>55</td>
</tr>
<tr>
<td>V</td>
<td>76</td>
<td>56</td>
</tr>
<tr>
<td>W</td>
<td>77</td>
<td>57</td>
</tr>
<tr>
<td>X</td>
<td>78</td>
<td>58</td>
</tr>
<tr>
<td>Y</td>
<td>79</td>
<td>59</td>
</tr>
<tr>
<td>Z</td>
<td>7A</td>
<td>5A</td>
</tr>
<tr>
<td>🅰️</td>
<td>5E</td>
<td>5F</td>
</tr>
<tr>
<td>🅱️</td>
<td>OA</td>
<td>5B</td>
</tr>
<tr>
<td>🅳️</td>
<td>O8</td>
<td>15</td>
</tr>
<tr>
<td>🅸️</td>
<td>O9</td>
<td>5D</td>
</tr>
</tbody>
</table>

**Note:** For Characters A through Z, press the key combination of **SHIFT** to utilize the upper/lowercase option. The unshifted codes will then apply.
Index

A

Alphabet array ............................................................... 186
Angle option for DRAW .................................................. 80
Arcs for CIRCLE .......................................................... 51
Array storage of graphics ............................................... 129, 130
Assembly language mnemonics ......................................... 217, 218

B

Background color ........................................................... 2, 6, 13, 27
Binary numbers ............................................................ 212, 226
Binary-to-hexadecimal .................................................... 214
Bit ................................................................ 212, 226
Boxes ........................................................................ 33
Box option for DRAW ....................................................... 34
Byte ............................................................................ 212, 226

C

Call machine language ...................................................... 210
Catenation with DRAW .................................................... 89, 94
Changing colors .............................................................. 27
Character set for graphics ................................................ 188
CIRCLE ...................................................................... 37, 40, 47, 65
CLEAR ....................................................................... 208, 226
Clear graphics screen .......................................................... 6
Clearing graphics pages ....................................................... 108
CLS .................................................................................. 16
COLOR .................................................................................. .11, 16, 25
Color adjustment of TV .......................................................... 3, 4
Color BASIC ........................................................................... ix, 2
Color for CIRCLE ....................................................................... 47, 49, 65
Color option for DRAW ........................................................... 85, 94
Connecting lines ........................................................................ 30
Copying graphics pages ............................................................. 111
Cursor .................................................................................... 2

D
DEFUSR ................................................................................. .210, 226
Diagonal lines ................................................................................ 29
Diagonal of a square ..................................................................... 81
Dimensioning a graphics array ................................................ 130, 150
Directions for DRAW ............................................................... 76, 94
DRAW ..................................................................................... 73, 93
DRAWing boxes ........................................................................... 33
Drawing CIRCLES ....................................................................... 37
DRAWing lines ............................................................................... 23
DRAWing text characters ............................................................ 184, 201
Draw string .................................................................................. 73, 93
Duration values for SOUND .................................................... 158
Dymax Gazette ............................................................................. x

E
Eccentricity for a CIRCLE .......................................................... 48
Entry address of USR. ................................................................ 210, 226
Equipment used in this book ..................................................... 2
Erasing points ............................................................................. 16
Extended Color BASIC. ............................................................. ix, 2

F
Filling boxes with color .................................................................. 35
Fill option for LINE .................................................................... 35
Foreground colors ....................................................................... 6, 13
INDEX

G

GET ...................................................... 129, 150
Getting the PPOINT ..................................... 61
Graphics modes ........................................... 5
Graphics pages ........................................... 107, 117, 122
Graphics positions ....................................... 8
Graphics resolution ...................................... ix

H

Height/width ratio with CIRCLE ......................... 49, 66
Hexadecimal digits ........................................ 213, 227
Hexadecimal/decimal conversion ......................... 215, 227
High resolution graphics ................................ ix

J

Joystick connections ...................................... 137
Joystick music ........................................... 173
Joystick pushbutton ..................................... 141, 151
Joysticks .................................................. 136, 151
Joysticks with PLAY ..................................... 171
JOYSTK .................................................... 136, 151

L

LINE ....................................................... 23, 31, 33, 35, 40
Low-resolution graphics ................................ ix
Low-resolution graphics from machine language .... 222

M

Machine language approach ................................ 209
Machine language call .................................... 210
Machine language format ................................ 212
Machine language Op codes ................................ 217
Medium-resolution graphics ............................... ix
Memory after PCLEAR ................................... 109
Memory location 65280 ................................... 141, 151
Memory structure ................................................. 212, 226
Musical number/notes for PLAY .................................. 161

N
Noises for PLAY .................................................. 165
Note length for PLAY ............................................. 162
Note values for SOUND ........................................... 158
No update option for DRAW ....................................... 84, 94

O
Operation codes .................................................. 217
Options for PLAY .................................................. 160, 175

P
PAINT ................................................................. 54, 66
Pauses for PLAY ................................................... 163
PCLEAR ............................................................... 108, 123
PCLS ................................................................. 6, 7, 16
PCOPY ............................................................... 111, 123
PEEK (65280) for joystick pushbutton ......................... 141, 151
PLAY ................................................................. 160, 174
PLAY options ....................................................... 160, 175
PLAY substrings .................................................. 163
PLAY using joysticks ............................................. 171
PLAY with graphics .............................................. 173
PMODE ............................................................... 5, 7, 16
Point size ........................................................... 9, 16, 22
PPPOINT ......................................................... 61, 67
PRESET ............................................................... 16
PRINT TIMER ..................................................... 192, 202
PSET ................................................................. 8, 16, 22
PUT ................................................................. 129, 150

R
Random painting .................................................... 58
Relative moves for DRAW ......................................... 91
INDEX

S

Scale option for DRAW ........................................... .101, 123
SCREEN ........................................... 5, 7, 16, 201
Size of graphic points ........................................... 9, 21
SOUND ........................................... 158, 174
SOUND with graphics ........................................... 159
Spirolaterals ........................................... 77
Starting page of graphics ...................................... .5
String variables for DRAW ...................................... 86, 94
STR$ ........................................... 90, 102
Substrings for DRAW ........................................... 87, 94
Substrings for PLAY ........................................... 163
Switching from graphics to text ................................ .181, 201

T

Tempo option for PLAY ........................................... 162
Text mode ........................................... 3, 202
TIMER applications ........................................... 194
TIMER = n ........................................... .192, 202
TIMER function ........................................... .192, 202
Turning graphics pages ........................................... .107, 116

U

USR ........................................... .207, 225
Explore the creative and imaginative blending of computers and color using TRS-80 Color Computer Graphics. This exciting book will enable you to explore all the graphics capabilities of Extended Color BASIC. You'll learn how to create interesting graphics to enhance your own computer programs. The book also provides application programs and useful subroutines that will be invaluable when you begin writing your own graphics programs.

This excellent book is loaded with information and tips on topics such as filling boxes with color, generating many lines from the same point, using joysticks in games, switching from graphics to text mode, and hundreds more. Each chapter ends with a summary and practice exercises. If you want to get the most out of your TRS-80 Color Computer, this is just the book you're looking for. Get ready for countless hours of fun!

Table of Contents:
1. Fundamental Coloring
2. Coloring Lines and Circles
3. More Circles, PAINT and POINT DRAW
4. Page Functions and More DRAW
5. PUT, GET, and Joysticks
7. Sound with PLAY
8. Displaying Text and Timing
9. The USR Function
   Additional Programs
Appendixes

Cover by Debbie Balboni

RESTON PUBLISHING COMPANY, INC.
A Prentice-Hall Company
Reston, Virginia

0-8359-7864-8