COMMODORE 64™ BASICS

A SELF-TEACHING GUIDE

ANN HARRIS

Harris

COMMODORE 64™ BASICS A SELF-TEACHING GUIDE

Wiley Press

Commodore 64™ Basics

Ann Harris

A Wiley Press Book
John Wiley & Sons, Inc.
New York • Chichester • Brisbane • Toronto • Singapore

Publisher: Judy V. Wilson
Editor: Theron Shreve
Managing Editor: Katherine Schowalter
Electronic Book Publishing Services: The Publisher's Network, Morrisville, PA

Commodore 64™ is a registered trademark of Commodore Electronics, Ltd.

Copyright© 1985 by John Wiley & Sons, Inc.

All right reserved. Published simultaneously in Canada.

Reproduction or translation of any part of this work beyond that permitted by Section 107 or 108 of the 1976 United States Copyright Act without the permission of the copyright owner is unlawful. Requests for permission or further information should be addressed to the Permissions Department, John Wiley & Sons, Inc.

Library of Congress Cataloging in Publication Data

Harris, Ann, 1944-
Commodore 64 basics.

1. Commodore 64 (Computer)–Programming. 2. Basic (Computer program language) I. Title.
QA76.8.C64H36 1984 001.64'2 84-2241
ISBN 0-471-88008-6

85 86 10 9 8 7 6 5 4 3 2 1

Acknowledgments

I would not have written this book without the help of some wonderful people. I would like to take this opportunity to thank some of them.

The photographs were taken by Karen Stockard Larssen Photography of Davis. Joan Aller and Pat Below helped me to write down the music for Chapter 10. My young friend Jim Diel came up with several of the ideas for the programs in Chapter 9 and 11. Martha Todd Parks spent many hours typing for me when my two fingers gave out. A special thank you to my family, Martin, Mary, and Megan, for the advice, love and support only you can provide.

This book is dedicated to my family and friends.

Table of Contents

CHAPTER 1 Exploring the Commodore 64 1
Hardware Review 2
Exploring the Keyboard 22
Suggestions for Further Exploration 31

CHAPTER 2 Beginning BASIC 32
The BASIC Language 33
Calculator Functions 34
Computer Programs 45
Input, Process and Output 47
A Math Facts Program 49
Input Statements 49
Saving Programs 53
Formatting a Diskette 56
Disk Commands 60
The Disk Directory 63
More Short Programs 64
Using a Printer 69
Suggestions for More Programming with Your New Skills 73

CHAPTER 3 Flowcharts 74
Drawing Flowcharts 74
Planning a Flowchart 80
Continuation Symbol 95
Suggestions for More Flowchart Practice 106

CHAPTER 4 Decisions 107
Branching Programs 107
IF Statements 112
RND and INT Functions 121
ON-GOTO Statements 132
REM Statements 136
GOTO Statements 137
Suggestions for Further Decision-Making 147

CHAPTER 5 Repetitions 148
Programming Repetitions 149
FOR-NEXT Loops 152
DO-WHILE Loops 163
DATA Statements 168
READ Statements 169
RESTORE Statements 171
Arrays 173
User-Defined Functions 183
Subroutines 186
Suggestions for Further Exploration 210

CHAPTER 6 Programming Style 211
Documentation 212
Backup Copies 213
Remark Statements 213
Indentation 214
Spacing 215
Debugging 217
Top-Down Design 224
Streamlining a Program 225

CHAPTER 7 Computer Math 228
Decimal Numbering System 228
Binary Math 230
Bits 231
Bytes 235
End-of-File Markers 237
Computer Memory Organization 238
Suggestions for Further Exploration 244

CHAPTER 8 Datafiles 245
Use of Datafiles 245
Fields 246
Sequential Files 247
Cassette Tape Datafiles 248
Disk Datafiles 252
Error Handling 255
Multiple Fields 257
Suggestions for Further Exploration 282

CHAPTER 9 Color and Graphics 283
Color from the Keyboard 284
CHR$ Codes 287
POKE Statements 289
Memory Maps 291
PEEK Statements 301
GET Statements 302
The Joystick 314
Suggestions for Further Exploration 321

CHAPTER 10 Sound and Music 322
Sound 322
The SID Chip 324
Music Using One Voice 328
Creating Rhythms 339
Music for One to Three Voices 342
Harmony 349
Sound Effects 361
Suggestions for Further Exploration with Sound 365

CHAPTER 11 Sprites 366
The VIC II Chip 367
Creating and Using Sprites 367
Ideas for Further Exploration with Sprites 415

APPENDIX A Error Messages 417

APPENDIX B Commodore 64 BASIC Keywords 420

APPENDIX C Flowcharting Symbols 421

APPENDIX D BASIC Functions for the Commodore 64 422

APPENDIX E BASIC Commands and Statements 426

APPENDIX F Abbreviations for BASIC Keywords 432

APPENDIX G CHR$ and ASCII Codes 434

APPENDIX H Screen Display Codes 436

APPENDIX I Memory Map for VIC II Graphics Chip 438

APPENDIX J Frequencies for Musical Notes 440

INDEX: 443

Preface

Welcome to *Commodore 64 Basics!*

This book is written for those Commodore 64 computer owners (or would-be owners) who would like to create their own programs but have had no programming instruction. In this book we will explore possibilities for programming all the features of the Commodore 64, using examples that will be of interest to all ages and all members of the family. The *Commodore 64 User's Guide* will be referred to, but you are not expected to have read it before we start.

The Commodore 64 is a good investment for a small price. It has all the features of the more expensive computers: 64K RAM, sixteen colors, hi-resolution graphics, moving figure graphics using sprites, polyphonic sound synthesis and programmable function keys. It has all the circuits and connections necessary for additional peripheral devices, such as a cassette recorder, disk drive, a printer, game paddles, a joystick, a light pen or a modem for telephone hook-up to another computer.

The Commodore 64 is suitable for business, educational, hobby and recreational programs. A current phrase that is used to praise a computer is 'user friendly,' and the Commodore 64 has the potential for being a very good friend indeed.

While the main focus of this book is on designing and writing programs in BASIC, we begin in Chapter 1 with a thorough description of all the hardware that can be used with the Commodore 64. Programming in BASIC begins in Chapter 2, and this chapter also includes information on storing and printing programs. Chapter 3 focuses on flowcharts as one method of preparing a program. Chapters 4 and 5 cover important logical concepts in BASIC. Chapter 6 is a short discussion of programming techniques and Chapter 7 covers some of the mathematical concepts needed to understand how computer programs

work. Chapter 8 covers datafiles for storing all kinds of information, and the last three chapters, 9, 10 and 11, introduce sound and graphics, including the wonderful "sprites."

While trying to anticipate all the questions that a beginner might have about programming the Commodore 64, the book also introduces some advanced information that provides for future growth and exploration.

There are two ways to make sure that you understand the material in this book. One is to try the programs as they are discussed; in fact, it is often assumed that you are doing just that. The second is to start creating your own ideas from the information as it is presented. There will be suggestions for doing this throughout the text.

I wish you every success and hope that you have as much fun with your '64' as I have had with mine.

CHAPTER 1

Exploring the Commodore 64

The Commodore 64 is an exciting computer—or more accurately, an exciting 'computer system.' In this first chapter, you will discover that the Commodore 64 computer system is made up of several parts that connect and work together. You will first become familiar with the different parts of the Commodore 64 computer system, while learning how to make the connections correctly every time.

By now you may have set up your computer system. If so, this chapter may be considered as a review and a way of becoming used to a programmed text using familiar material. Or, use it as a reference when you buy something new to add to the system.

This chapter brings together information that can be found in *The Commodore 64 User's Guide, The 1541 Single Drive Floppy Disc User's Manual* and *The 1525 User's Manual.*

After completing this chapter you will be able to:

- Identify all the connection slots on the Commodore 64.
- Connect the Commodore 64 to a monitor or a TV set with the optional connection to a hi-fidelity sound system.
- Connect the Commodore 64 to a C2N data cassette and perform the operational test.
- Connect the Commodore 64 to a 1541 disk drive and perform the operational test.

- Connect the Commodore 64 to the 1525 printer and perform the operational test.
- Understand the function of all the keys on the Commodore 64 keyboard.

Hardware Review

Besides the computer itself, a system may have:

- A screen, a TV or special monitor for display.
- A C2N cassette recorder, sometimes called a datasette, to store programs and information.
- A 1541 single drive floppy disk, another type of storage device.
- A 1525 graphics printer.
- A modem for data communication by phone.
- Joysticks and paddles for game playing.

It is not necessary for you to have all these 'peripheral' devices to use this book. The computer and a screen will be fine. If you have one of the storage devices, then you can keep all the exciting programs we are going to write.

A cassette recorder is the least expensive form of data storage. However, it is slow, and the user has to keep track of where programs and data are stored on the tape. A cassette recorder is a sequential device, which means that when a program is requested, the computer starts searching and the tape moves in one direction - forward - until the program is found. A disk drive is more expensive, but it is faster, and the computer keeps track of where data is stored on the diskette. A disk drive has direct access, which means that the computer can find the program requested without searching through all the programs on the diskette.

Questions & Answers

Your first question, fill in the blank:

A computer _____ is made up of several parts that work together.

Answer: system

One very common computer term that is used when talking about the parts of a computer system is HARDWARE. Anything you can touch, such as the computer keyboard or the screen, is hardware. The opposite of hardware is SOFTWARE, which you cannot touch, for example, computer programs.

Questions & Answers

Can you pick out the items in this list which are hardware?

_____(a) Cassette unit
_____(b) Power pack
_____(c) Test programs
_____(d) TV

Answer: (a), (b) and (d) are hardware. You cannot touch the test programs.

Another term to understand before we start is BUG. It is a favorite computer expression which covers any kind of error or problem. If the disk drive is not working, you have a hardware bug, and if the checkbook program will not balance an account correctly, you have a software bug. Software bugs can almost always be resolved by the user with some thought, patience and maybe some advice from a programming friend. Hardware bugs generally require the service of a computer technician at your nearest Commodore outlet.

Questions & Answers

Which of these statements best defines a bug?

_____(a) The person next to you who wants to use the computer.
_____(b) A problem or error with the computer.
_____(c) A creepy-crawly creature.

Answer: (b) This is the true computer answer - the others may be true under different circumstances!

Before you start unpacking the computer, make sure that you have a clean, flat and level surface that will be large enough to hold all the pieces. It is recommended that the monitor or TV be placed behind the computer keyboard so that the screen is easy to read when you are typing. You should also keep the computer system away from smoke, cooking fumes, dust, and magnets, as these tend to foul tape heads and disk drive.

The CPU

First, unpack the computer keyboard unit. This is the 'heart' of the computer and contains the CPU, or Central Processing Unit. On the top of the CPU is the computer keyboard. There are sockets on the right-hand side and at the back (Figure 1.1).

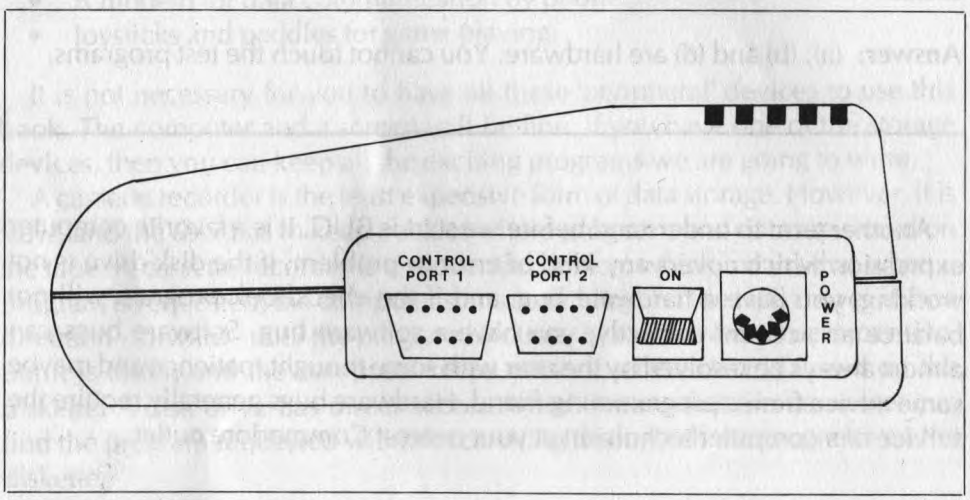


Figure 1.1 Commodore 64 Computer System with C2N Datasette, 1525 Printer, 1541 Disk Drive and Joystick.

Look at the right-hand side first.

There are three sockets and a switch on this side.

The switch is the power ON/OFF switch for the computer.

The socket at the back of the ON/OFF switch, called the POWER SOCKET, is for the power supply.

The other two sockets are marked 'CONTROL PORT 1' and 'CONTROL PORT 2.' They are used for game control devices, such as joysticks or paddles, or for a light pen. There are some devices made by companies other than Commodore that will also fit these ports. Of course you should check for compatibility before you buy any device.

Figure 1.2 shows the back of the computer. The long socket on the left, is called the CARTRIDGE SLOT, where you plug in program and game cartridges.

To the right of the cartridge slot is the CHANNEL SELECTOR switch. This is used to select a TV channel when a TV set is used as a monitor.

The TV CONNECTOR supplies the picture and sound to the TV set.

Next to the TV connector is the VIDEO PORT, which supplies the picture and sound to a monitor.

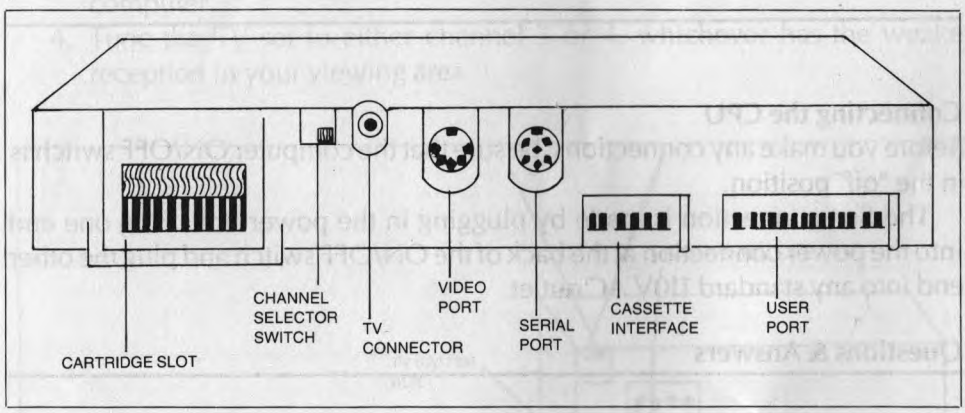


Figure 1.2 Side View of the CPU

Next to the video port is a SERIAL PORT. It is similar to the video port, but if you look closely you will notice that it has a different pin pattern. A disk drive or printer can be plugged in here.

On the right-hand side of the serial port is the CASSETTE INTERFACE and, if you are using a cassette unit, it is plugged in here. A Commodore C2N cassette unit is a special kind of cassette recorder that is used to store programs and data. Only a Commodore cassette recorder will fit in here.

Next to the cassette interface is the PARALLEL USER PORT, which is used to connect a modem or other user custom devices.

Questions & Answers

Match these sockets with their use:

_____(a) Cassette interface	1. Power pack
_____(b) Serial port	2. Joystick
_____(c) Video port	3. Disk drive
_____(d) Cartridge slot	4. Cassette unit
_____(e) Control port	5. Monitor output
_____(f) Parallel user port	6. Modem
_____(g) Power socket	7. Game cartridge

Answer: (a) 4, (b) 3, (c) 5, (d) 7, (e) 2, (f) 6, (g) 1.

Connecting the CPU

Before you make any connections, be sure that the computer ON/OFF switch is in the "off" position.

The first connection is made by plugging in the power unit. Plug one end into the power connection at the back of the ON/OFF switch and plug the other end into any standard 110V AC outlet.

Questions & Answers

Before connecting the power unit, the ON/OFF switch on the Commodore 64 computer should be in the _____ position.

Answer: off

Output Devices - TV or Monitor

The next step is to connect the computer to the output device. There are three possible ways to connect the Commodore 64 for output. It can be connected to a:

- TV, for both graphics and sound.

- special monitor, for both graphics and sound.
- TV or monitor for graphics, and the amplifier of a sound system for sound.

The picture resolution on a TV set is not as high as on a monitor. Letters and pictures will not appear as sharp, or clear, on a TV as on a monitor.

If you are using a TV set, you will need to use the TV Switch Box that came with your Commodore 64. Use a small screwdriver to connect the TV Switch Box. Follow these steps, and when you are finished the connections should be like those of Figure 1.3.

1. Connect the video cable to the phone jack on the TV Switch Box.
2. Connect the two cables on the TV Switch Box to the two VHF antenna leads on the back of the TV.
3. Plug the other end of the video cable into the video port at the back of the computer.
4. Tune the TV set to either channel 3 or 4, whichever has the weaker reception in your viewing area.

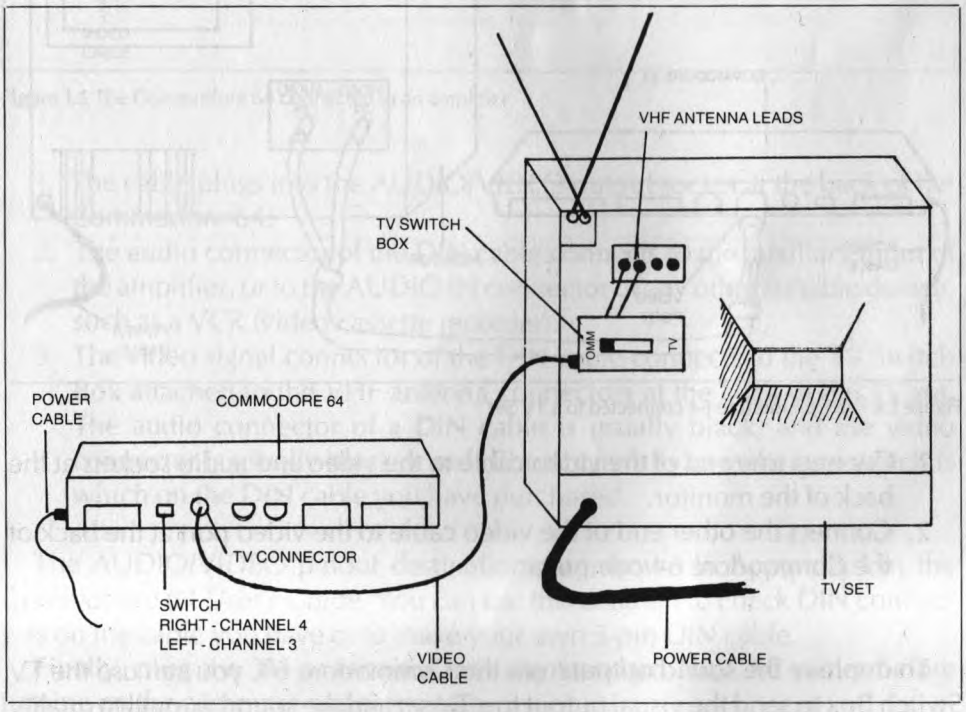


Figure 1.3 Back View of the CPU

5. Set the switch on the computer to the same number as the channel selected on the TV.
6. Set the switch on the front of the TV Switch Box to 'computer.'

That's it! The connection is made.

The second option is to use a monitor for the output from the Commodore 64. A monitor is similar to a TV except that it does not have a tuner, so it cannot receive regular TV channels. A monitor can convert the composite video signal produced by the Commodore 64 directly into picture and sound, so the TV Switch Box need not be used (Figure 1.4).

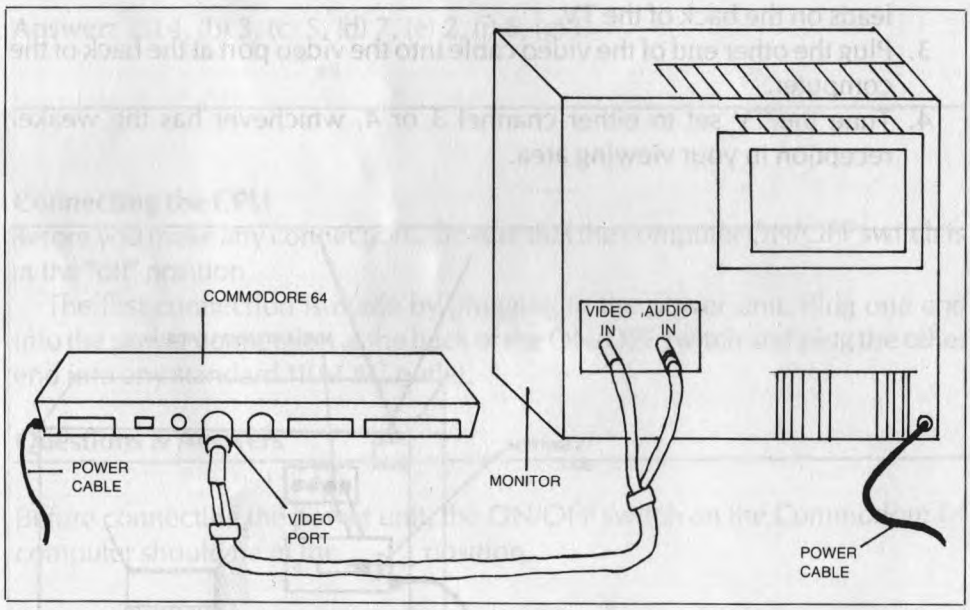


Figure 1.4 The Commodore 64 connected to a TV Set

1. Connect one end of the video cable to the video and audio sockets at the back of the monitor.
2. Connect the other end of the video cable to the video port at the back of the Commodore 64 computer.

To improve the sound output from the Commodore 64, you can use the TV Switch Box to send the visual output to a TV set and the sound through a quality amplifier in your sound system.

For this connection you will need to purchase a standard 5-pin DIN audio cable. Figure 1.5 shows the connections.

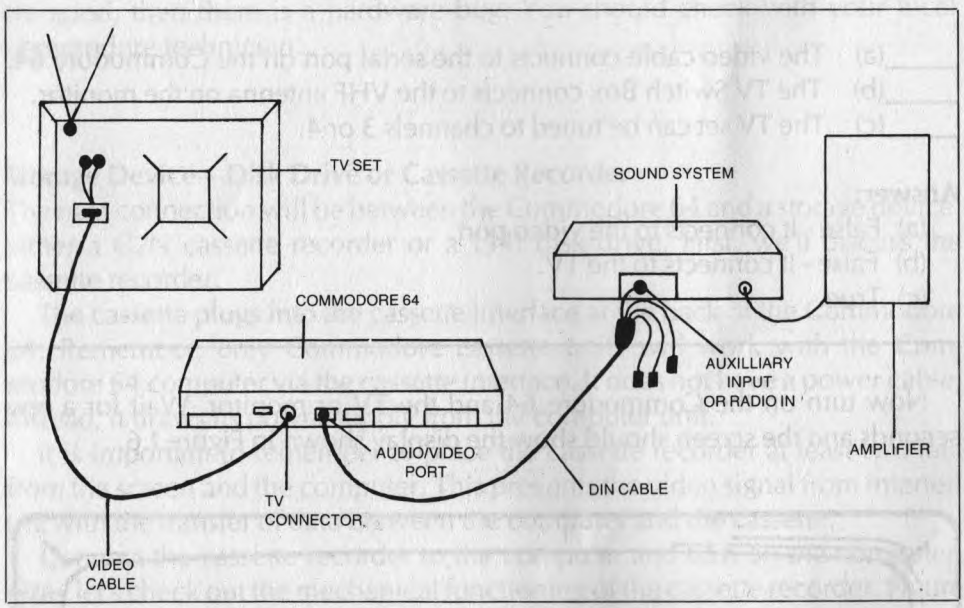


Figure 1.5 The Commodore 64 connected to an amplifier

1. The cable plugs into the AUDIO/VIDEO output socket at the back of the Commodore 64.
2. The audio connector of the DIN cable connects to the auxiliary input of the amplifier, or to the AUDIO IN connector of any other suitable device, such as a VCR (video cassette recorder).
3. The video signal connector of the DIN cable connects to the TV Switch Box attached to the VHF antenna connectors at the back of the TV set. The audio connector of a DIN cable is usually black, and the video connector is usually black or red. Check carefully to make sure which is which on the DIN cable you have purchased.

The AUDIO/VIDEO pinout destinations are shown in Appendix I in the *Commodore 64 User's Guide*. You can use this diagram to check DIN connectors on the cable you have or to make your own 5-pin DIN cable.

Finally, plug the TV or monitor into any standard 110V AC outlet before turning on the computer and the screen.

Questions & Answers

True or False:

_____(a) The video cable connects to the serial port on the Commodore 64.
_____(b) The TV Switch Box connects to the VHF antenna on the monitor.
_____(c) The TV set can be tuned to channels 3 or 4.

Answer:
(a) False - It connects to the video port.
(b) False - It connects to the TV.
(c) True.

Now turn on the Commodore 64 and the TV or monitor. Wait for a few seconds and the screen should show the display shown in Figure 1.6.

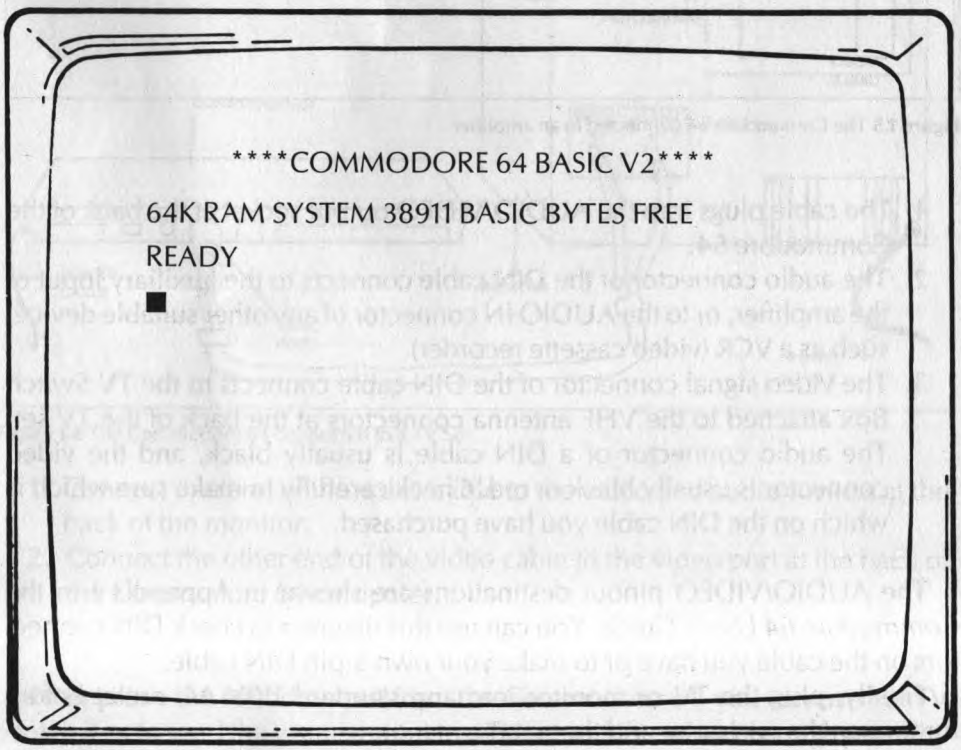


Figure 1.6 The screen display when the Commodore 64 is first turned on.

If you do not get this, try adjusting the screen contrast and vertical hold controls, and check that the TV and computer are both set to the same channel (i.e., 3 or 4). If these suggestions do not fix the problem, and the connections are good, then there is a hardware bug. You should check with your local Commodore technician.

Storage Device—Disk Drive or Cassette Recorder

The next connection will be between the Commodore 64 and a storage device, either a C2N cassette recorder or a 1541 disk drive. First, we'll discuss the cassette recorder.

The cassette plugs into the cassette interface at the back of the Commodore 64. Remember, only Commodore cassette units will work with the Commodore 64 computer via the cassette interface. It does not have a power cable; instead, it draws its power supply from the computer unit.

It is important to remember to place the cassette recorder at least two feet from the screen and the computer. This prevents the video signal from interfering with the transfer of data between the computer and the cassette.

Connect the cassette recorder to the computer and turn on the computer. Now let's check out the mechanical functioning of the cassette recorder. Figure 1.7 shows the controls.

The C2N cassette recorder functions mechanically, in the same way as an ordinary cassette tape recorder. The cassette motor should be off, unless one of the buttons is depressed.

The STOP/EJECT key should turn the motor off if it is running and open the cassette recorder.

The PLAY button engages the READ/WRITE heads and the take-up spindle on the right should move slowly and evenly in a counter-clockwise direction.

The REWIND button should leave the READ/WRITE heads disengaged and the supply spindle on the left should move rapidly and evenly in a clockwise direction.

The F.FWD (fast forward) button should leave the READ/WRITE heads disengaged, and the take-up spindle should move rapidly and evenly in a counter-clockwise direction.

If these operations do not proceed smoothly, there is a problem with the mechanical functioning of the cassette unit and it should be returned to the dealer for repair or replacement.

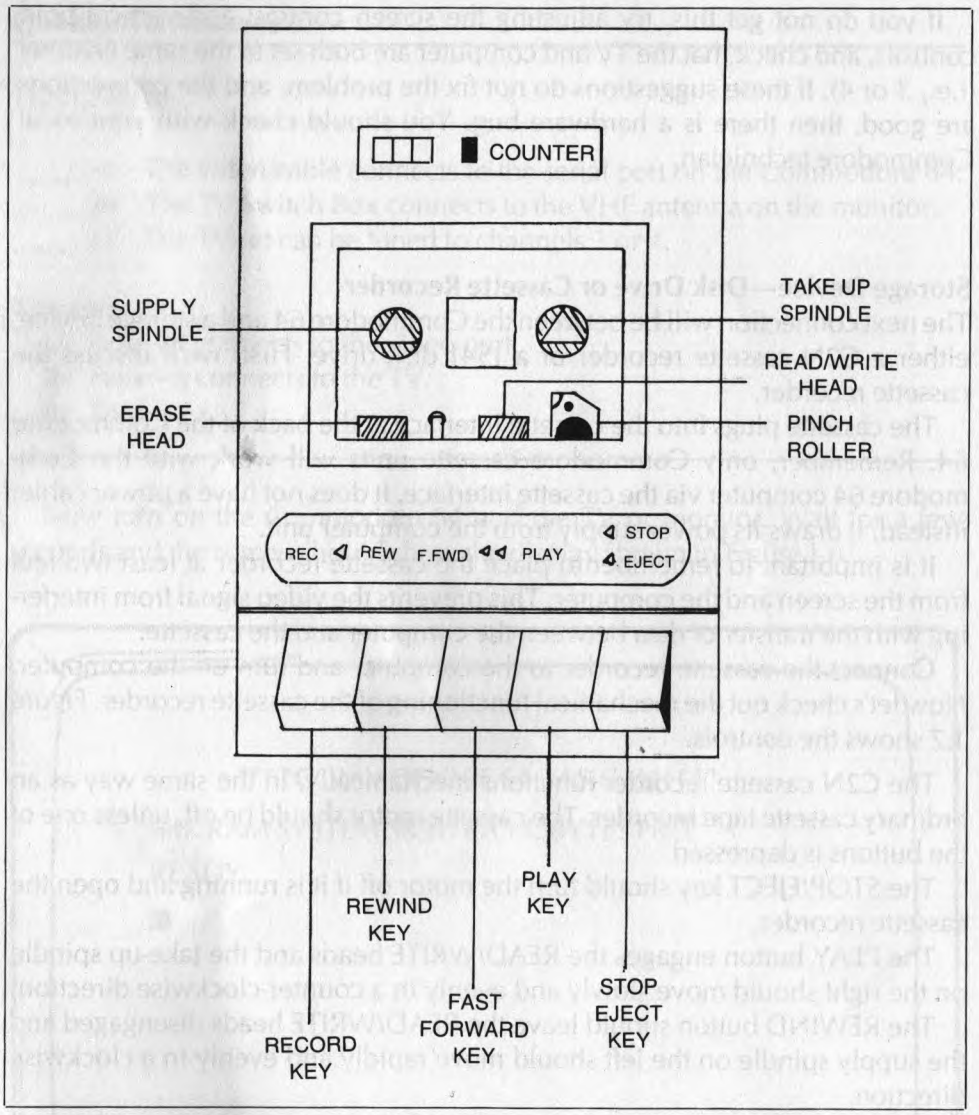


Figure 1.7 C2N Datassette.

Questions & Answers

The C2N does/does not have its own power source?

Answer: Does not.

If you do not have a Commodore 1541 Disk Drive or a Commodore 1525 Printer to connect at this time you may want to skip this discussion.

In this section we will connect the 1541 Commodore Single Drive Floppy Disk. This connection is always made without a diskette in the disk drive.

As shown in Figure 1.8 at the back of the disk drive there are:

- two interface connectors.
- a power socket.
- a slow blow fuse.
- the power ON/OFF switch.

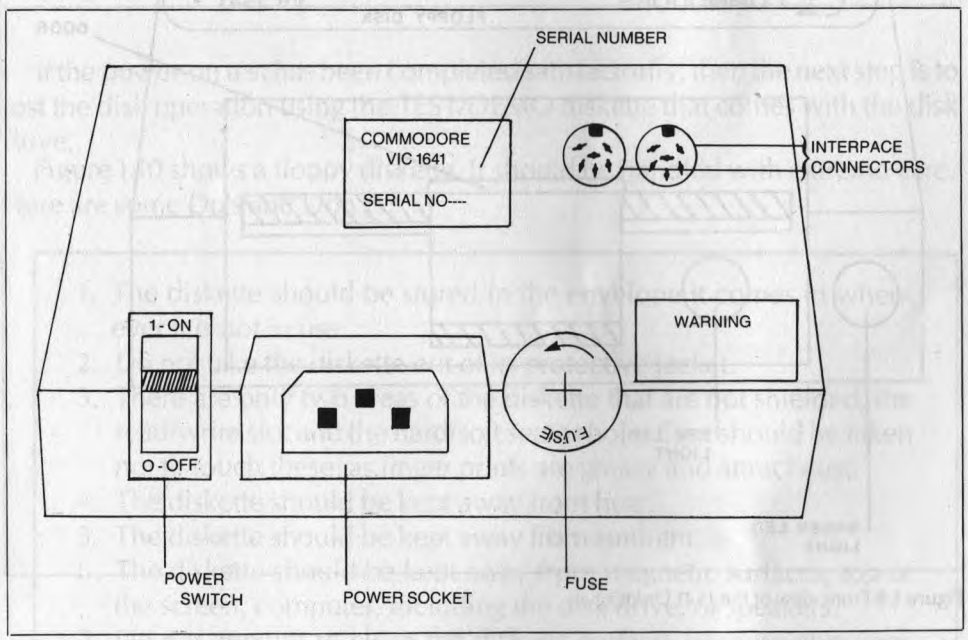


Figure 1.8 C2N controls.

The disk drive should be placed on a level surface as close to the computer as possible, and the following steps should be followed:

1. The power to the computer should be turned OFF.
2. The serial cable is connected to the interface connector at the back of the disk drive.
3. The other end of the serial cable is plugged into the serial port at the back of the Commodore 64.

4. The power cable is plugged into the back of the disk drive and to a standard 120V wall socket.

Figure 1.9 shows the front of the 1541 disk drive. There is a slot to insert the diskette and a door that closes down to secure the diskette onto the spindle hub. There are two LED lights. The green light is the power light and will be lit whenever the power to the disk drive is on. The red light comes on whenever the disk drive is active and will flash whenever there is a disk drive error.

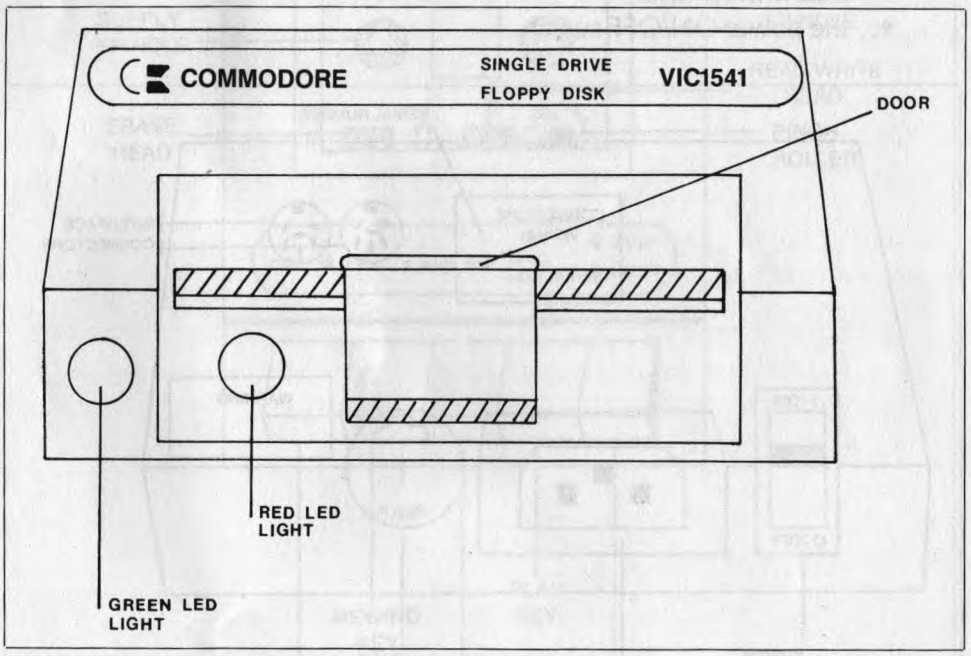


Figure 1.9 Front view of the 1541 Disk Drive.

Turn on the power to the computer and the disk drive. Both the green and red LED lights on the front of the disk drive will come on. After about ten seconds the red LED light will go off. If the red light stays on or flashes after a few seconds, turn the disk drive off and wait for a minute before trying again. If the problem persists, take the disk drive back to your Commodore dealer.

Questions & Answers

Which of the following features are found at the back of the 1541 disk drive?

_____(a) Slow burn fuse.
_____(b) LED lights.
_____(c) Serial interface.
_____(d) Power switch.

Answer: (a), (c) and (d) are found at the back of the disk drive. (b) is on the front.

If the power-on test has been completed satisfactorily, then the next step is to test the disk operation using the TEST/DEMO diskette that comes with the disk drive.

Figure 1.10 shows a floppy diskette. It should be handled with extreme care. Here are some Do's and Don't's:

1. The diskette should be stored in the envelope it comes in whenever it is not in use.
2. Do not take the diskette out of its protective jacket.
3. There are only two areas of the diskette that are not shielded, the read/write slot and the hard/soft sector hole. Care should be taken not to touch these, as finger prints are greasy and attract dust.
4. The diskette should be kept away from heat.
5. The diskette should be kept away from sunlight.
6. The diskette should be kept away from magnetic surfaces, top of the screen, computer, including the disk drive, or speakers.
7. Do not attempt to clean the diskette surface, because abrasions will also destroy data.
8. Do not write on the plastic jacket with a pencil or ballpoint pen. This will scratch the diskette. Make a habit of writing on the label before you stick the label on the diskette. Or, you may use a felt-tip pen, gently.
9. Use a special storage box for the diskettes. Keep it close to the computer so that it is easy to reach and return the diskettes to a safe place.

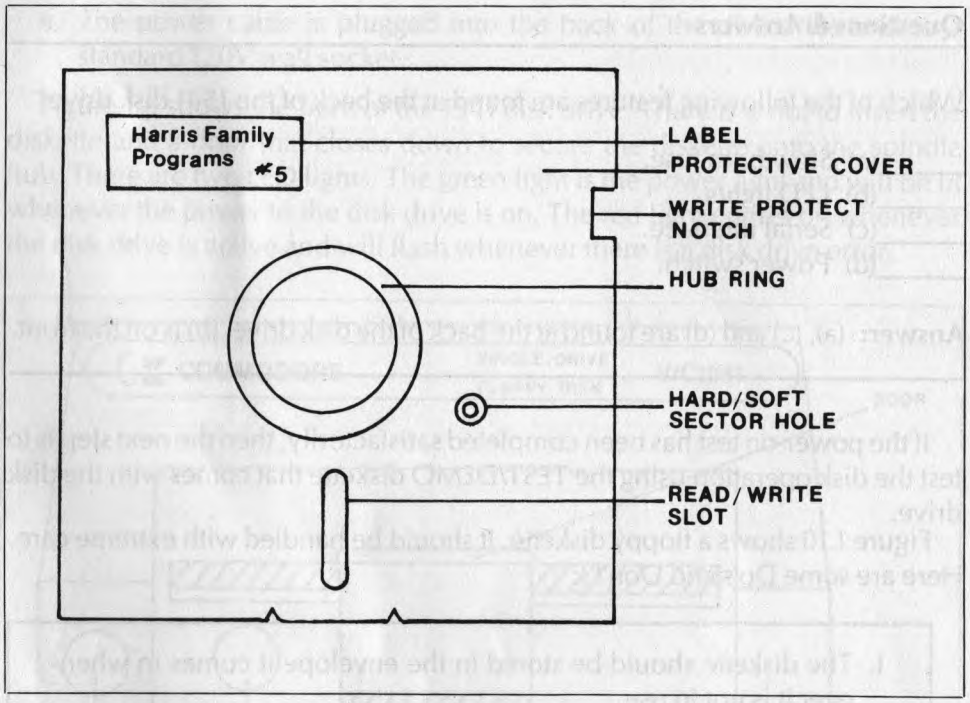


Figure 1.10 5 1/4" Soft Sectored Diskette.

With the power turned off, practice inserting the diskette in the disk drive and closing the door. The diskette should be inserted with the label facing up and the write/protect notch on the left-hand side. Push the diskette in until the door can be lowered to secure the disk drive. To release the diskette, push gently on the lip of the door. The door should open and the diskette slide forward. Sometimes the diskette does not slide forward and a small pair of tweezers is useful to gently pull it out.

Questions & Answers

Which of the following are good tips for taking care of your diskettes?

____(a) Always write on the diskette with a NO. 2 pencil.
____(b) Keep the diskettes in their envelopes and in a box when not in use.
____(c) Keep the diskettes away from magnetic surfaces.
____(d) Don't hold the diskette by the label.

Answer: (b) and (c) are good advice. (a) and (c) could end in disaster.

To perform the drive performance test, first turn on the power to the computer and the disk drive. Then insert the TEST/DEMO diskette. This test requires a 'scratch' diskette. Do not use a diskette containing any programs that you value; they will be destroyed during this test. Now type in the following statements:

```
OPEN 1,8,15
Press RETURN
LOAD "PERFORMANCE TEST",8
Press RETURN
```

The screen will display:

```
SEARCHING FOR PERFORMANCE TEST
LOADING
READY

Type RUN and press RETURN
```

The screen will show:

```
    PERFORMANCE TEST
INSERT SCRATCH DISKETTE IN DRIVE
PRESS RETURN WHEN READY
DISK NEW COMMAND WAIT ABOUT 80 SECONDS
```

During these 80 seconds the computer will reformat the scratch diskette now in the disk drive, erasing everything on it. After the performance test it will be available for use to store future programs or data. At the end of the formatting the screen displays:

```
0   OK  0 0
DRIVE PASS
```

Then the following statements appear as the computer completes the mechanical test:

```
MECHANICAL TEST

OPEN WRITE FILE
          0  OK  0 0
WRITING DATA
          0  OK  0 0
CLOSE WRITE FILE
          0  OK  0 0
OPEN READ FILE
          0  OK  0 0
READING DATA
          0  OK  0 0
SCRATCH FILE
          1 FILE S
SCRATCHED 1 0

WRITE TRACK 35
          0  OK  0 0
WRITE TRACK 1
          0  OK  0 0
READ TRACK 35
          0  OK  0 0
READ TRACK 1
          0  OK  0 0
UNIT HAS PASSED PERFORMANCE TEST!

PULL DISKETTE FROM DRIVE BEFORE TURNING POWER OFF.
```

When the test is completed, label the scratch diskette to remind you that it is now formatted and ready for use. If the disk drive did not complete the drive performance test, then you should return it to your Commodore dealer with a note stating where the drive failed the test.

Questions & Answers

Complete these statements:

(a) The green LED light is ON when the power is_____.

(b) The red LED light flashes when there is a _____.

Answer: (a) on. (b) disk error.

The Printer

Connecting the Commodore 1525 Printer is also easy. The printer cable connects to the back of the printer and to either the serial port at the back of the Commodore 64 or the interface connector at the back of the disk drive. When the printer and the disk drive are connected to the computer in this way, it is called 'daisy chaining.' The power cable is plugged into a standard 110V socket.

Questions & Answers

The computer power should be _____ when the printer is connected.

Answer: OFF. . . always, when a connection is being made.

When the printer is connected and the power turned on, the red power indicator light should be on and the printer head should move into the center of the carriage and then return to its original position at the left-hand side.

Before the printer head test can be performed, the ribbon cassette and paper must be in place.

Remove the cover and install the right side of the ribbon first. Figure 1.11 shows where the ribbon fits into the printer. There are holes in the ribbon holder of the printer that match the pins on the bottom of each half of the ribbon cassette. Make sure that the ribbon is not twisted and slowly pull to the left, threading the ribbon around the ribbon cam before attaching the left-hand side.

CAUTION

Do not try to move the the print head manually as this will damage the printer.

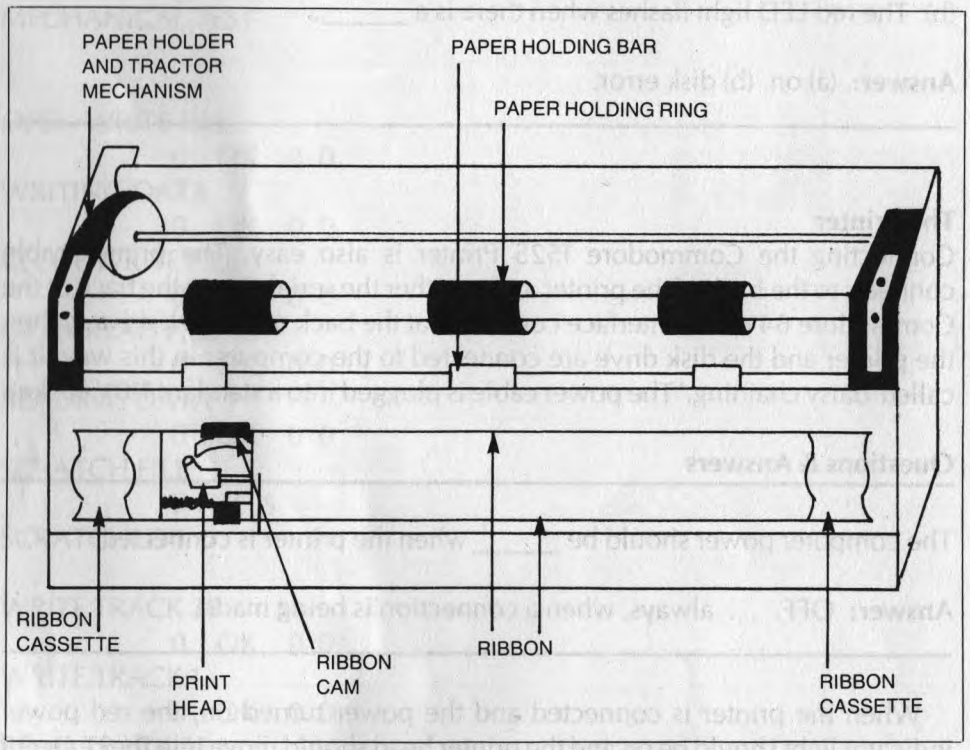


Figure 1.11 Diagram for installing the ribbon and paper in the 1525 Printer.

Once the ribbon is in place, insert the paper from the rear of the printer. Lift up the paper-holders so that the paper can fit over the sprocket pins. The paper should feed smoothly between the ribbon and the platen and under the paper-holding bar. The three paper-holding rings can be moved along the bar and should be positioned so that they are not directly under the smaller rings on the paper-holding bar. This holds the paper tighter and lessens the vibration and noise of the printer. The sprockets can be moved until they are the right distance apart to hold the paper taut. When they are adjusted, the paper-holding bar is lowered and the paper-holders are set over the sprocket pins.

Now the paper can be advanced using the paper-feed dial. There is a lever on the print head that can be adjusted according to the thickness of the paper. This is in position 1 for single thickness but can be adjusted if more than one part form is being fed into the printer.

The print head test is always performed with the ribbon and paper installed in the printer. Printing without paper can damage the print head.

Figure 1.12 shows the back of the 1525 printer, where there is a self-diagnosis

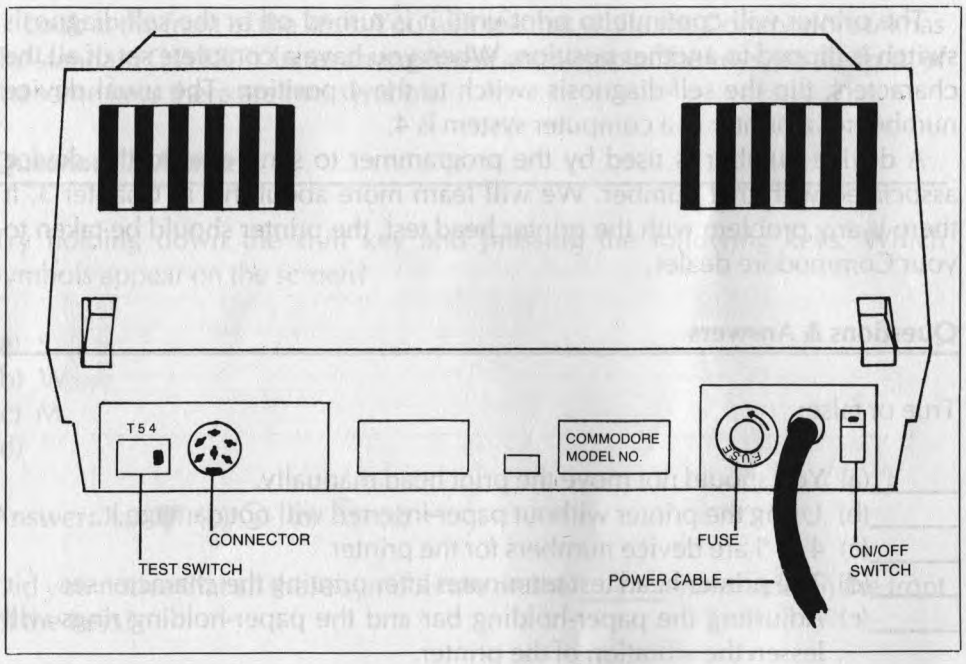


Figure 1.12 Back View of the 1525 printer.

test switch. It has three positions: T, 5, and 4. 5 and 4 are the device numbers that the printer can assume, and T is the test position. Move the switch to the T position. The printer will start to print all the characters available on the Commodore 64, as shown in Figure 1.13.

Figure 1.13 A print test showing all the characters available on the Commodore 64.

The printer will continue to print until it is turned off or the self-diagnosis switch is flipped to another position. When you have a complete set of all the characters, flip the self-diagnosis switch to the 4 position. The usual device number for a printer in a computer system is 4.

A device number is used by the programmer to send data to the device associated with that number. We will learn more about this in Chapter 3. If there is any problem with the printer head test, the printer should be taken to your Commodore dealer.

Questions & Answers

True or false:

_____(a) You should not move the print head manually.
_____(b) Using the printer without paper inserted will not damage it.
_____(c) 4 or 5 are device numbers for the printer.
_____(d) The printer head test terminates after printing the character set.
_____(e) Adjusting the paper-holding bar and the paper-holding rings will lessen the vibration of the printer.

Answer:

(a), (c), and (e) are true.
(b) is false. Using the printer without paper WILL damage it.
(d) is false. The printer has to be switched to the number 4 to terminate the print test.

Exploring the Keyboard

You cannot break a computer by typing on it! You may lose some data or a program by typing the wrong thing at the wrong time, but you will not break the computer. The worst that will happen is that you will have to spend time reconstructing the lost information.

We now have the system assembled, and it is time to start using it. Turn on the Commodore 64 and the monitor and type in something: your name and address or the Pledge of Allegiance—anything that pleases you. Notice that the Commodore 64 uses upper-case letters.

If you have tried using the shift key, you will have found that instead of switching from uppercase to lowercase, the Commodore 64 printed a symbol.

Look at the front of the keys. You will see that while the Commodore 64 has the standard QWERTY letter arrangement on the top of the keys, on the front of the same keys there are two symbols.

Questions & Answers

Try holding down the shift key and pressing the following keys. Which symbols appear on the screen?

(a) S
(b) W
(c) M
(d) *

Answer: (a) ♥ (b) ○ (c) \ (d) —

Did you notice that all these symbols are on the _____ -hand side on the front of the keys?

Answer: Right.

At the bottom left-hand corner of the keyboard, you will find the Commodore key, C<. Hold this down and press these keys. Which symbols appear?

(a) S
(b) W
(c) M
(d) *

Answer: (a) ┐ (b) ┼ (c) | (d) ◥

All these symbols are on the _____ -hand side on the front of the keys.

Answer: Left.

To summarize our findings so far, when the Commodore 64 is first turned on, the keyboard is in uppercase. The shift key accesses the symbols on the right-hand side on the front of the keys, and the Commodore key accesses the symbols on the left-hand side.

The Commodore 64 automatically moves to the beginning of the next line when it reaches the 40th print position on line.

If you try using the RETURN key as a carriage return, you will have an error message on the screen:

```
? SYNTAX ERROR
```

The RETURN key sends what is on the screen to the computer memory. If it is not a legal BASIC statement, the computer prints an error message on the screen to inform you that it was unable to process the information. There is a complete listing of the Commodore 64 error messages in Appendix A.

The next step is to activate the lower-case type. Press the Commodore key and the shift key at the same time and all the letters on the screen will switch to lowercase. Everything typed now will be in lowercase, with the shift key acting in the way we would expect on a standard typewriter. Type in the following to convince yourself that this is indeed the case:

```
HRH Queen Elizabeth II
```

The cursor is the flashing square that tells you where you are on the screen.

At the bottom right-hand side of the keyboard, there are two cursor control keys. The horizontal key moves the cursor to the right on the screen; the shift key moves it to the left. The vertical cursor control key moves the cursor down the screen; the shift key moves the cursor up. With these two keys you can move the cursor to any position. This is a very useful editing function.

Type in the following exactly, without making any corrections:

```
The last day to fail your income tax is april 15th.
```

Using the cursor control keys, move the cursor until it is over the 'a' in fail and type 'ile' to change 'fail' into 'file'. Move the cursor to the 'a' in 'april' and make it a capital 'A.'

Questions & Answers

Match the key(s) with the direction of cursor movement:

_____ (a) Horizontal cursor key
_____ (b) Vertical cursor key
_____ (c) Horizontal key and shift
_____ (d) Vertical key and shift

(1) Up
(2) Down
(3) Right
(4) Left

Answer: (a) 3, (b) 2, (c) 4, (d) 1.

At the top right-hand corner of the keyboard is another very useful key. It is marked INST DEL. This is the delete key. With the shift, it becomes an insert key.

Type in the following statement:

The interest rate on passbook savings is 46%.

When the bank manager sees the line of customers outside the bank, she notices the mistake and quickly changes the interest rate to 4%!

To do this, use the cursor keys to move the cursor until it is over the 6. Then press the INST DEL key. The 6 disappears and the % is now next to the 4. The cursor must be to the immediate right of the character to be deleted.

Now if the central office decides that the interest rate should really be 14%, we need to insert a '1' before the '4'. Move the cursor until it is over the 4. Hold down the shift and press the INST DEL key. A space opens up in the line and the '1' can be typed in. When inserting, the cursor must be exactly where the space is to be inserted.

Questions & Answers

Saute onioins until golden brown.
The cursor must be over the _____ in 'onioins' to delete the second 'i'?

Answer: Febuary 14th is Valentine's Day.

The cursor must be over the _____ to insert the missing 'r' in February?

Answer: n

Next to the INST DEL key is the CLR HOME key. If you press this key, the cursor moves to the "home" position in the top left-hand corner of the screen. If the shift is held down and this key is pressed, then the cursor returns to the home position and the screen is cleared.

There is another way to clear the screen: push the RUN STOP key and the RESTORE key at the same time. The RESTORE key will only work with the RUN STOP key. This time the screen is cleared and the BASIC prompt, READY, appears in the top left-hand corner of the screen. The RUN STOP and RESTORE keys clear the screen and reset the workspace in the computer memory. They do not erase any program that is in memory.

Questions & Answers

Match the keys with the results produced:

_____ (a) CLR HOME
_____ (b) RUN STOP and RESTORE
_____ (c) SHIFT and CLR HOME

(1) READY in screen top left-hand corner and clear screen.
(2) Cursor in top left-hand corner and screen cleared.
(3) Cursor in screen top left-hand corner.

Answer: (a) 3, (b) 1, (c) 2.

The final feature of the Commodore 64 keyboard for this chapter is the color control. The top row of the keyboard has the standard number keys. The zero has a slash through it to distinguish it from the letter O. Also on the top row are the arithmetic plus and minus signs and an English monetary pound sign, £. The shift key will access the symbols printed above the numbers on the top of the keys.

On the front of the number keys 1 through 8 are the names of eight colors. To change the color of the characters on the screen, press the CTRL key on the left-hand side of the keyboard and the color key of your choice. The cursor will change color and now anything typed on the screen will be in that color.

Questions & Answers

Roses are red
Violets are blue

_____ (a) Which two keys would you use to type the first line in red?
_____ (b) Which two keys would you use to type the second line in blue?

Answer: (a) CTRL 3 (b) CTRL7.

There are sixteen colors on the Commodore 64 and all of them can be accessed from the keyboard. The other eight colors, which are not shown on the front of the number keys, are accessed by pressing the Commodore key, C<, and a number key at the same time. Figure 1.14 shows a complete listing of the sixteen colors and the access keys:

Keys	Reverse Symbol	Color
CTRL 1		Black
CTRL 2		White
CTRL 3		Red
CTRL 4		Cyan
CTRL 5		Purple
CTRL 6		Green
CTRL 7		Blue
CTRL 8		Yellow
C= 1		Orange
C= 2		Brown
C= 3		Light Red
C= 4		Gray 1
C= 5		Gray 2
C= 6		Light Green
C= 7		Light Blue
C= 8		Gray 3

Figure 1.14 Colors available from the Commodore 64 keyboard.

Questions & Answers

What colors will you see when you press the following keys?

_____ (a) CTRL 8
_____ (b) Commodore Key 1
_____ (c) Commodore Key 2
_____ (d) Commodore Key 6
_____ (e) CTRL 1

Answer: (a) yellow, (b) orange, (c) brown, (d) light green, (e) black.

The last two keys to try are the numbers 9 and 0. On the front of the 9 key is RVS ON. When used with the CTRL key, this key reverses the foreground and background colors.

Here is a good way to see the full range of the Commodore 64 colors - in our family this is called "color bar."

1. Clear the screen.
2. Press CTRL and the RVS ON key.
3. Choose a color using the CTRL and number keys.
4. Press the space bar.

After you have a color bar of a length that pleases you, change the color and repeat the process until you have reviewed all the sixteen colors available. When you have finished, press the CTRL and the RVS OFF key. The foreground and background will be returned to the original colors.

Questions & Answers

How would you clear the screen and return the characters to the normal color?

Answer: Press RUN STOP and RESTORE.

There are four other keys that are set apart from the keyboard on the right-hand side. These are function keys. They can be used with the shift key to give a total of eight function keys. Some programs that you buy or copy from maga-

zines may use them, but we will not use them in this book. We have already mentioned the British pound key, £. The Commodore 64 is very popular in Great Britain, but this key also makes very pretty designs!

Before going on to Chapter 2, please be sure to work the following self-check.

SELF-CHECK

Do not panic. This check is ungraded, non-threatening and your answers will not be shared with your friends, family or relations!

Please take the time to work through these questions. Some questions may refer to pieces of hardware that you do not have in your system, so you should skip those questions.

The first 5 questions refer to the following list:

power pack	monitor
printer	keyboard
TV	C2N cassette
disk drive	joystick

1. Which items are used to store information?
2. Which items are used to display computer output?
3. Which items are hardware?
4. Which are software?
5. Which items are part of a Commodore 64 computer system?
6. True or false:
   (a) You can use the Commodore 64 without a power pack.
   (b) The Commodore 64 may be connected to a TV or a monitor.
   (c) The C2N cassette unit plugs into the cartridge slot.
   (d) The 1525 printer may be plugged into the interface connector at the back of the 1541 disk drive.
   (e) The green LED light will flash when the 1541 disk drive is working.
7. The power to the computer should always be _____ when a connection is made.

8. The _____ is the flashing square that tells you where you are on the screen.

9. Match the following keys with their actions:

(a) CLR HOME	(1) delete
(b) RUN STOP RESTORE	(2) cursor to top left of screen
(c) INST DEL	(3) clear screen
(d) INST DEL SHIFT	(4) insert
(e) CLR HOME SHIFT	(5) clear screen and computer memory.

10. Type in the following phrases using uppercase and lowercase and the color mentioned in each phrase:

Blue suede shoes
Yellow is the color of my true love's hair
The purple-headed mountains
What is black and white and red all over?

11. Take your computer system apart and put it together again. Also try explaining this procedure to your best friend, boss, or significant other!

Answers to SELF-CHECK

1. The C2N cassette and disk drive.
2. TV, monitor or printer.
3. All of them. Hardware is something you can touch, and these are all solid items.
4. None of them. You can't touch software.
5. All of them. By definition, a system is made of parts that connect together, and all of these things can be connected together.
6. (a) F The power pack must be used to supply the correct voltage to the computer.
   (b) T
   (c) F The C2N cassette connects to the cassette interface.
   (d) T This is called daisy-chaining. The printer can also be connected directly to the computer serial port.
   (e) F The green LED light is on when the power to the disk drive is on. The red LED light flashes when there is a disk error.

7. OFF—probably the most important lesson in this chapter.
8. Cursor.
9. (a) 2, (b) 5, (c) 1, (d) 4, (e) e. All of these keyboard functions are used frequently. If you have trouble remembering which keys do what, please practice some more.
10. Use the Commodore and Shift key to move into lower-case print. The shift key will now access uppercase, and the CTRL and the color keys will change the color of the print.
11. If you have trouble with this one, re-read the applicable discussions.

Suggestions for Further Exploration

1. Take some time to find the right spot to set up your Commodore 64 and a safe place to keep your cassettes or tapes.
2. Explore the keyboard until you are familiar with all the keys.Use the color and graphics on the Commodore 64 to make a pattern all over the screen, such as a quilt design or a mandala.

CHAPTER 2

Beginning BASIC

In this chapter we are going to start writing programs in BASIC for the Commodore 64. We will start by using the computer, as if it were a calculator, to perform one instruction at a time and work toward writing longer and more complex programs. We will take a little time to look at how the computer handles information. Also, there will be a detailed discussion of how to save programs on tapes and disks. Finally, we will involve the printer in one of our programs.

After completing this chapter, you will be able to:

- Write direct statements using PRINT.
- Use the computer to perform mathematical operations.
- Understand the terms *integer, decimal, exponent* and *scientific notation.*
- Understand the terms *alphanumeric* and *string.*
- Understand the concept of constants and variables.
- Create legal variable names.
- Identify the three parts of a computer program.
- Use the BASIC system commands NEW, RUN, LIST, SAVE and LOAD.
- Write BASIC programs using the statements INPUT, PRINT and LET.
- SAVE and LOAD a program using a data cassette recorder.
- Understand the DOS commands for the 1541 disk drive.
- SAVE and LOAD a program using a disk drive.
- Load the directory from a diskette.
- Use the printer in a program.

The BASIC Language

We do not need to do anything special to load BASIC into the Commodore 64. It is there when the computer is turned on. (Don't forget to turn on the screen. From now on, 'turning on the computer' is going to mean turning on every part of the system.) Figure 2.1 shows the screen when the computer is first turned on.

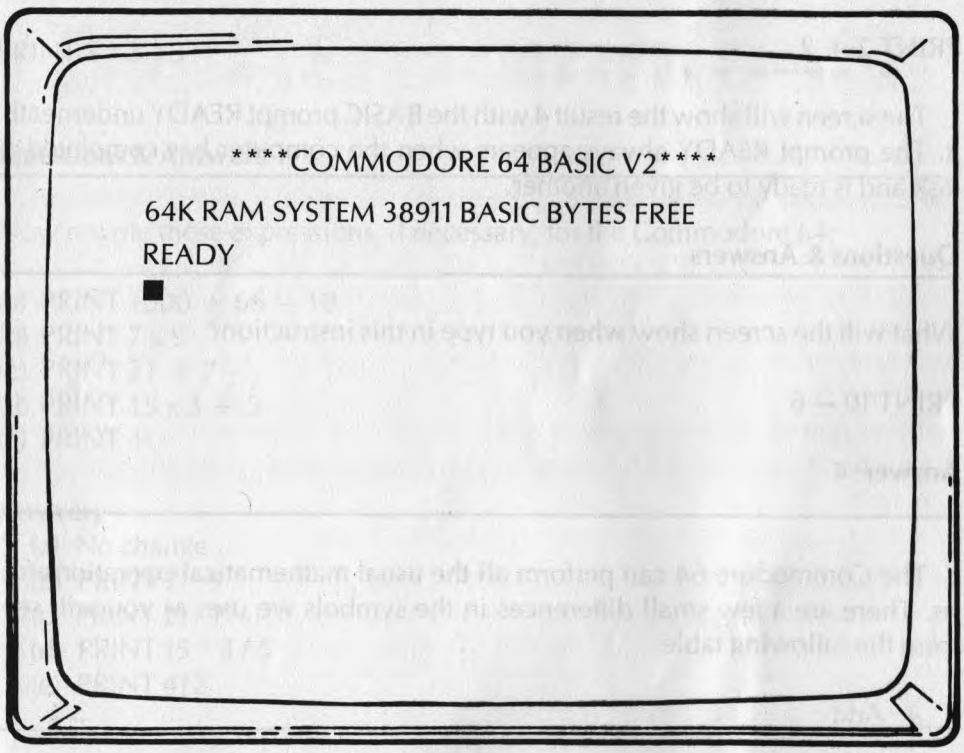


Figure 2.1 Commodore 64 initial screen display.

The word to recognize at the moment is BASIC. BASIC stands for *Beginner's All-Purpose Symbolic Instruction Code.* BASIC is the programming 'language' that is used on most microcomputers. All the programs in this book will be written using BASIC. There are other programming languages, such as PILOT, FORTH and LOGO, that are available for the Commodore 64.

The initial screen tells us how much memory the system has, 64K, that BASIC is in the computer memory, and that the amount of memory left for use

is 38911 bytes. That will be more than enough memory for the programs in this book. We will discuss bytes in Chapter 7, when we are ready to use the information.

Calculator Functions

When the computer is turned on, we are ready to begin. Type in the following line and press RETURN.

```
PRINT 2 + 2
```

The screen will show the result 4 with the BASIC prompt READY underneath it. The prompt READY always appears when the computer has completed a task and is ready to be given another.

Questions & Answers

What will the screen show when you type in this instruction?

```
PRINT 10 – 6
```

Answer: 4

The Commodore 64 can perform all the usual mathematical operations for us. There are a few small differences in the symbols we use, as you will see from the following table:

+	Add
–	Subtract
*	Multiply
/	Divide
↑	Exponentiation

Notice that the sign for multiply is an asterisk "*" and not "X." The sign for divide is a slash "/" and not "÷." The sign for exponentiation is an up arrow "↑" The up arrow is at the right-hand end of the second row of the keyboard next to the RESTORE key.

The math operations are familiar to most of us, except for perhaps exponentiation. Here is an example of an exponentiation problem:

```
PRINT 3 ↑ 3
```

The result printed on the screen is 27. Exponentiation raises a number to a power. That is, the number is multiplied by itself a specified number of times. This example could have been written:

```
PRINT 3 * 3 * 3
```

Questions & Answers

Now rewrite these expressions, if necessary, for the Commodore 64:

(a) PRINT 1000 + 66 – 10
(b) PRINT 7 x 9
(c) PRINT 21 ÷ 7
(d) PRINT 15 x 3 ÷ 5
(e) PRINT 4^2

Answer:

(a) No change
(b) PRINT 7 * 9
(c) PRINT 21 / 7
(d) PRINT 15 * 3 / 5
(e) PRINT 4↑2

Sometimes the problem we want to solve has more than one operation. For example:

```
PRINT 4 + 3 / 100 * 6
```

The computer is a logical machine and it performs calculations in a set order, starting at the left of the equation and working to the right. There is also an order of precedence, in which the calculations are performed from left to right:

– Minus sign, meaning a negative number
↑ Exponentiation
* / Multiplication and division, left to right
+ – Addition and subtraction, left to right

The order of precedence means that the computer will do multiplication and division steps before it does addition and subtraction.

Here is an example to show the effect of the order of precedence:

```
PRINT 2 * 6 + 2
```

In this case, the result is fourteen and not sixteen. This is because the multiplication was performed before the addition. If we wanted the result sixteen, then the calculation 6 + 2 would have to be enclosed in parentheses. Parentheses override the normal logical progression of the computer and mean that the expressions enclosed within them are evaluated first. To see how this works, try this:

```
PRINT 2 * (6 + 2)
```

The result to this problem is sixteen.

You can have more than one set of parentheses in an expression. For example:

```
PRINT (6 + 3) * (5 –1)
```

Or the parentheses can be "nested" this way:

```
PRINT ((6+3) *6)/2
```

The Commodore 64 can evaluate up to a maximum of ten nested parentheses.

WARNING

Make sure that the number of left parentheses in an expression equals the number of right parentheses. Otherwise, you will get an error message.

Questions & Answers

What will the result of the following operations using the order of precedence and performing the calculations from left to right? You might like to try solving these with a pencil and paper before you use the computer.

(a) PRINT 6 / 2 + 3
(b) PRINT 6 / (2 + 3)
(c) PRINT 6 – 2 * 3 ^ 2 + 4 / 2
(d) PRINT 10 + 5 * 7
(e) PRINT 6↑2 / 6
(f) PRINT ((16 – 4) / 2) * 3

Answer: (a) 6, (b) 1.2, (c) –14, (d) 45, (e) 6, (f) 18.

We have seen that the Commodore 64 will handle both negative and positive numbers. Now let's take a little time to find the limits for the Commodore 64. How large or small a number will it handle?

Type this and press RETURN.

```
PRINT 12345678.90 + 11.22 + 333.444
```

The result is not as expected, 12346023.564; instead, the computer has rounded off the result to 12346023.6. The Commodore 64 can work with up to ten digits, but can display only nine. When a number exceeds nine digits, the ninth digit is rounded up if the tenth digit is five or more, and rounded down if the tenth digit is less than five.

Questions & Answers

Try rounding off the following numbers yourself before you try them on the computer:

(a) PRINT –98765.321098
(b) PRINT 123456789098
(c) PRINT –1234.543212
(d) PRINT 98765.67899

Answer:
(a) −98765.3211
(b) 123456789
(c) −1234.54321
(d) 98765.6790

Numbers between 0.01 and 999,999,999 are printed in the usual way. Numbers outside this range are printed using scientific notation. This means that they are expressed as an exponent, a power of 10. For example, try this:

```
PRINT 10000000000000000000
```

The result is 1.0E + 20. That is another way of saying 1.0 * 10 * 20. The plus sign indicates which way to move the decimal point. For example:

```
123.456E + 3 = 123456
123.456E − 3 = 0.123456
```

We will not use scientific notation in this book, but you should be able to recognize numbers written this way if they appear when you are writing programs of your own, or if you accidentally use too large a number in one of the programs.

Questions & Answers

Here are some numbers written in scientific notation for you to rewrite:

(a) − 1.12233E + 3
(b) 1.0E − 5
(c) 29.96E + 6

Answer:
(a) −1122.33
(b) .000001
(c) 29960000

Alphanumeric or String Information

Of course, not everything we want to use our computer for involves numbers. We want to be able to use letters, words and other characters, such as punctuation marks or a $ sign. When we mix numbers, punctuation and letters the information is called *alphanumeric*.

This is alphanumeric information:

1600 Pennsylvania Ave.

Another computer term that is frequently used for this kind of information is STRING.

We can print strings on the screen very easily. Try this:

```
PRINT "HI"
```

The Commodore 64 should say "Hi" right back to you. Notice the quotation marks. These are essential when we want to print string information. Try repeating the last statement without the quotation marks, and you will see a zero printed on the screen. The Commodore 64 needs quotation marks around string information.

The longest string that we can use on the Commodore 64 is 255 characters long. Spaces count as characters. In fact, any symbol, letter, number or space is considered a character.

Questions & Answers

What will the Commodore 64 print on the screen when you type in this statement?

```
PRINT "WHAT IS YOUR NAME"
```

Answer: WHAT IS YOUR NAME
(The Commodore 64 will not answer the question!)

Variables

So far we have divided information into numbers and non-numbers or strings, but there is another way of looking at information. We can think of information

as being unchanging or CONSTANT and varying or VARIABLE. Constants will be the same whenever a program is used. Some examples of constants are:

52 weeks in a year.
The numerical value of π.
100 cents in a dollar.

The other kind of information is a VARIABLE. This information is not the same each time the program is used and may vary while the program is being used. Examples of variables are:

Someone's name.
The price of bubble gum.
The temperature in San Francisco.

Questions & Answers

Which of the following items of information are constants and which are variables?

(a) Monthly utility bill.
(b) Number of days in March.
(c) Points gained in a football match.
(d) Student's age on January 1st.
(e) Temperature of boiling water at sea-level.

Answer: (b) and (e) are constants. The other are variables.

The Commodore 64 will store many different variables for us as long as we give each separate item of information a name. Here are some rules for naming variables:

- Each name may be up to two characters long, either two letters or a letter and a number.
- A name must always begin with a letter.
- Each name in a program must be a unique name.
- If the name is longer than two letters, then the extra is ignored.

- The variable name must not include any of the BASIC reserved words. These are listed in Appendix B.

Sometimes it makes a program clearer to the reader to use longer variable names, but care must be taken to keep the first two letters unique. The variables DATE and DAILY BALANCE, for example, will have the same name, DA, in memory because the computer will ignore everything after the first two characters.

Questions & Answers

Which of the following are valid variable names?

(a) SI
(b) TI
(c) IF
(d) IN
(e) ON
(f) OR
(g) OF

Answer: (a), (d), and (g) are valid variable names.

In some programs all the calculations are kept as integer, or whole number, calculations. In this case, a percent sign, %, is added to the end of the variable name to indicate that the number stored under that variable name will be an integer. We need to be careful when mixing integer and decimal numeric variables, as the result will always be truncated if it is stored as an integer numeric variable. This means that the numbers after the decimal point will be dropped.

```
A% = 3.12 -2
PRINT A%
```

A% in this example will be 1 because 1.12, the result of 3.12 – 2, has been truncated to 1.

Questions & Answers

What will the result to this problem be?

X% = 3.12 – 1.5
(a) 1
(b) 1.5
(c) 2

Answer: (a) The result 1.62 is truncated to 1.

How would you check this answer using the computer?

Answer: Type in these two direct statements and press RETURN after each one:

```
X% = 3.12 –1.5
PRINT X%
```

Storing information as an integer variable will sometimes save memory space. This is a consideration with large programs, but will not be essential for the programs included in this book.

Not all the information that we deal with is in the form of numbers. If we want to program a mailing list of names and addresses, then we will want the computer to store alphanumeric or string information.

The following statement will print a zero screen:

```
A = "ALBERT EINSTEIN"
```

The computer needs an addition to the variable name to indicate that the information to be stored is alphanumeric. To do this, the variable name is chosen in the usual way and a dollar sign is added. Also, quotation marks need to be added. Try these statements:

```
A$ = "ALBERT EINSTEIN"
PRINT A$
```

Questions & Answers

Which of the following variable names is correct?

(a) B$ = "MAY 6TH"
(b) MC$ = COMMODORE
(c) AD = "2005 REGIS DR"

Answer: (a) is correct.

Normal mathematical operations cannot be carried out on string variables, but you can concatenate them (join them together). Try this:

```
A$ = "E PLURIBUS"
B$ = "UNUM"
PRINT A$ + B$
```

The screen should display E PLURIBUS UNUM.

Questions & Answers

IF N$ = "64" and C$ = "COMMODORE" can you write a PRINT command to print COMMODORE 64 on the screen?

Answer: PRINT C$ + N$

When you type information between quotation marks, you are using the QUOTE MODE. In the quote mode you can create "special effects" by changing the color or using the cursor controls inside the quotation marks.

Try typing in this statement:

```
PRINT "<CTRL 1> H <CRSR> I"
```

You should see the letters H and I, in white, arranged like this:

H
I

This, however, does have its disadvantages. Once you have typed the first quotation mark you have entered the quote mode and cannot back up with the cursor keys to change a mistake. But, there are other ways of correcting mistakes. You can use the DEL key to back up to the mistake, destroying characters as you go, and then type everything over. You can use SHIFT and RETURN to escape from the line, or you can finish the statement, press RETURN, and then go back to the line with the cursor keys.

Try correcting this statement after you have typed the word "QUOUTE" and before you type MODE:

```
PRINT "THIS IS QUOUTE MODE"
```

When you are correcting inside quotation marks and use the INST key to create extra spaces, you move into the INSERT MODE. Be sure to create only as many spaces as you will need. If you add too many spaces and then move forward with the cursor key, the blank spaces will be filled with reverse characters. To remove them, you will have to backspace with the DEL key.

Experiment with making changes inside quotation marks until you are familiar with the necessary procedures.

Questions & Answers

Are these statements true or false?

(a) The insert mode occurs when you use the insert function.
(b) The 'cursor up' key will move the cursor up the screen in the quote mode.
(c) The INST DEL key is not affected by the quote mode.
(d) The INST DEL key is not affected by the insert mode.

Answer:

(a) and (c) are true.
(b) The cursor up key will not print a reversed character to represent its function in the quote mode.
(c) The INST DEL key is affected by the insert mode.

Computer Programs

The direct commands that we have been writing could be considered simple programs. A computer program is a list of instructions in logical sequence that completes a task. Direct commands do complete a one-step task!

Writing a computer program is not difficult. If you are worried because you were not very good at math in school or think that only a genius can write computer programs, then you are in for a pleasant surprise. Anybody can write a program. In fact, you perform many programs each day without writing them down; making the bed, setting the table, and driving to work are all tasks with a logical sequence of steps. Here is a program for making popcorn:

1. Put two tablespoons of oil in a heavy pan.
2. Add one-quarter cup of popcorn.
3. Turn heat on high.
4. Turn off heat when all the popcorn has popped.
5. Eat and enjoy.

I am sure you can think of others, such as how to make coffee, play tic-tac-toe with a friend, or walk to school.

Questions & Answers

The important point to remember is that a program must follow a logical sequence. Can you put the following steps in the correct order to help a friendly Martian phone home?

(a) Listen for the dial tone.
(b) Hang up the receiver.
(c) Dial the number.
(d) Listen for phone ringing.
(e) Pick up the receiver.
(f) Speak to the person.

Answer: e,a,c,d,f,b.

A computer program differs from an everyday program in that it has to be written in a language the computer understands, not in English.

In BASIC, statement numbers are given to each instruction in the program. If

we wanted to use the PRINT statement in a program we could type in this line:

```
10 PRINT "THIS IS A PROGRAM"
```

When RETURN is pressed at the end of this statement the READY prompt appears, but the computer does not print THIS IS A PROGRAM on the screen. To execute a program, that is, to make the program do each of the steps in sequence, we must use the BASIC command RUN. RUN does not require a statement number.

Type RUN and press RETURN and THIS IS A PROGRAM will be printed on the screen. Now type LIST and press RETURN. This time the computer prints our program to the screen. LIST is another command that does not require a statement number.

Questions & Answers

True or false:

(a) RUN and LIST require statement numbers.
(b) RUN executes the program.
(c) LIST displays the program on the screen.

Answer:

(a) False. They do not require statement numbers.
(b) True.
(c) True.

Now we will add another statement to our program. It can be typed below the READY prompt on the screen. It is not necessary to move the cursor to the line below 10 PRINT "THIS IS A PROGRAM." The computer will automatically add the new statement to the program in memory in the correct number sequence. This is the next statement:

```
20 PRINT "THE END"
```

Enter this line and run the program again. This time there should be two lines of output on the screen.

```
THIS IS A PROGRAM
THE END
```

Notice that we did not number the statements in the program with consecutive numbers. This allows us to add a statement in the middle of the program if we want to later. Try adding this line:

```
15 PRINT "ISN'T THIS EASY"
```

Questions & Answers

Where do you put the cursor before typing the line?

(a) Immediately under line 10.
(b) Immediately under line 20.
(c) Immediately under READY.

Answer: (c) is the best answer. Always make additions at the end of the program listing; the computer will re-order the statements.

What will the screen display when you run the program now?

(a) THIS IS A PROGRAM
    THE END
    ISN'T THIS EASY
(b) THIS IS A PROGRAM
    ISN'T THIS EASY
    THE END

Answer: (b) Now LIST the program and notice that the statements are in numerical sequence.

Input, Process and Output

All computer programs have three parts: INPUT, PROCESS and OUTPUT.

The popcorn-making example can be shown as having input, process and output.

INPUT	popcorn
	oil

PROCESS	put ingredients in pot apply heat turn off heat
OUTPUT	popped corn

The divisions between input, process and output in the program using PRINT statements are not so obvious. The input was the messages in quotations in the PRINT statements. The process was to store the messages in the computer memory, and the output was the messages on the screen.

It may seem that we are stating the obvious, but you will be surprised to find out how easy it is to forget to input a piece of information or to assume that the computer will automatically perform a process.

Questions & Answers

The three parts of a program are _____, _____, and _____.

Answer: Input, process, output.

Now let's apply this to one of my daughter's homework problems:

9 + 5 =

The input is the nine and five. The process is addition, and the output will be the result, fourteen.

Questions & Answers

Here is another one of her problems:

7 – 3 =

(a) In this case the process is _____.
(b) The output will be _____.

Answer: (a) subtraction, (b) the result, 4.

A Math Facts Program

Now we are going to write a program that will do math facts. Before we start a program we need to make sure that the work area in the computer memory is cleared. To do this, enter the BASIC command NEW. Any program that may have been in memory will be lost, and the command LIST will return only the BASIC prompt READY.

The first step in programming is to clearly state the steps to be completed. The list of steps is called an ALGORITHM. For my daughter's homework problem, the algorithm is to add any two numbers and print the result on the screen. Notice that the algorithm says ANY two numbers; we do not want to write a separate program for every combination! In other words, we want to add two VARIABLES - two numeric variables.

Here are the steps for our program:

1. Take two numbers. (INPUT)
2. Add the two numbers together. (PROCESS)
3. Print the results on the screen. (OUTPUT)

INPUT Statements

The INPUT statement in BASIC is used to enter information from the keyboard. We will use it to accomplish step 1 of our algorithm to get two numbers. INPUT has this format:

INPUT ["<prompt message>"]; <variable>, <variable> ...

- INPUT is the BASIC statement and must always be included.
- The optional message may be any short (40 characters or less) message to prompt the typist to enter the relevant information.
- The ';' semicolon is required punctuation after a prompt message.
- The variable name for the information being entered is last. The usual rules for variables apply. If there is more than one variable to be entered, then the list of variable names is separated by commas.

The first statement for the addition program will be:

```
10 INPUT "GIVE ME TWO NUMBERS"; A, B
```

Questions & Answers

Which of the following INPUT statements is written correctly?

(a) 40 INPUT "TELL ME YOUR HEIGHT AND WEIGHT"; H
(b) 40 INPUT "TELL ME YOUR HEIGHT AND WEIGHT", H; W
(c) 40 INPUT "TELL ME YOUR HEIGHT AND WEIGHT"; H, W
(d) 40 INPUT "TELL ME YOUR HEIGHT AND WEIGHT"; H; W

Answer:

(c) is correct.
(a) has only one variable name instead of two.
(b) and (d) have the wrong punctuation. They should have a semicolon after the prompt message and a comma between the variable name.

The process statement for the math facts program will be an equation and a variable name must be chosen to store the result. In this program we will use the letter C to represent the result.

```
20 C = A + B
```

There is a BASIC statement LET that can be used for calculations. Statement 20 could be changed to:

```
20 LET C = A + B
```

The word LET is optional. Most experienced programmers leave it out because it takes up memory space. Look at the two statements:

```
20 LET C = A + B
20 C = A + B
```

In the second statement, LET is implied but the meaning is still clear, so LET may be considered redundant. However, it you feel that LET makes it easier for you to follow the logic in your program, then use it. I will not be using it again.

Questions & Answers

Write a process statement to add four numbers, W,X,Y,Z, together and store the result in T.

Answer: 20 T = W + X + Y + Z

The output statement for the math facts program will be a PRINT statement.

```
30 PRINT A "+" B "=" C
```

The complete program looks like this:

```
10 INPUT "GIVE ME TWO NUMBERS ";A, B
20 C = A + B
30 PRINT A "+" B "=" C
```

Type in these three statements and remember to press RETURN at the end of each one. Now type RUN and press RETURN. This should appear on the screen:

```
"GIVE ME TWO NUMBERS ?"
```

The computer is waiting for you to type in two numbers separated by a comma. If you forget the comma or enter anything other than a number, you will see the following error message: ? REDO FROM START. If this happens, enter two numbers separated by a comma and press RETURN again. If the program runs successfully, the problem and the result should appear on the screen. For example:

```
2+ 2 = 4
```

If this did not happen and there is an ERROR IN LINE NUMBER message on the screen, then you have a program bug. Type LIST and the line number in the error message and press RETURN. Check the statement carefully and retype the statement and press RETURN. Or, edit the statement printed on the screen by moving the cursor to the error and making the correction. Don't forget to

press RETURN after you have made the correction. Now run the program again.

One frequent cause of problems in an INPUT statement is the prompt message.

WARNING

The ?REDO FROM START message will also occur if the prompt message in an INPUT statement is more than forty characters long. This is one screen line in length. If this happens you will not be able to enter any data.

If you want to use a long prompt message, then it should be in a PRINT statement before the INPUT, like this:

```
10 PRINT "THIS IS A VERY LONG PROMPT MESSAGE TO THE USER TO
   ASK FOR YOUR AGE"
20 INPUT A
30 PRINT "THANK YOU"
```

Notice that because there is no prompt message in the INPUT statement, there is no semicolon in front of the variable name. Run this program and then rewrite it to include the prompt message in the INPUT statement. Run it again and see what happens.

Questions & Answers

How would you get the READY prompt back on the screen?

Answer: Use RUN STOP and RESTORE.

Once the addition program has run successfully, which statement would you change to make this a subtraction program?

Answer: 20

What would the new statement look like?

Answer: 20 C = A – B

Make the changes and RUN the program a few times.

Saving Programs

Now that you have a successful program, you may want to save it for future use. In the next section of this chapter we will cover in detail the commands used to send information to, and get information from, the storage devices. While you are working on this, keep in mind that we still have our math program in the computer memory and that the commands in the following section will not erase it.

Saving Programs on a C2N Datasette Recorder

First we will talk about a cassette recorder, the C2N Datasette. If you are storing your programs on a diskette, you may wish to skip this section.

To save a program on the C2N Datassette recorder, make sure that a tape is loaded. A short tape, 5-15 minutes, is best because this reduces the wear on the tape heads from winding and rewinding the tape. More than one program can be stored on a tape. For this reason, and because we want to be able to retrieve the program at a later date, we need a name for the program. A name is not essential, but it is extremely helpful. In this case we will call the program MATH.

Enter the following instruction:

```
SAVE "MATH"
```

Notice the quotation marks around the program name. The closing quote is optional. SAVE is the BASIC command used to send the program to a storage device such as a cassette recorder. The screen should now show:

```
PRESS RECORD AND PLAY ON TAPE
```

Carry out this instruction by pressing the record and play buttons on the cassette recorder. The screen will go blank while the program is being saved and then display:

```
SAVING MATH
READY
```

The BASIC command VERIFY can be used to check that the program stored on the tape is the same as the program in the computer memory. To use VERIFY, rewind the tape and enter the following command:

```
VERIFY "MATH"
```

The screen will go blank while the computer searches for the MATH program. SEARCHING FOR MATH and FOUND MATH will be displayed. Then there will be a short pause before the screen goes blank again, and finally the message VERIFYING OK will appear. Now the operation is complete and the BASIC prompt READY will appear on the screen. If there is a verifying error, then repeat the steps to SAVE the program. If the error persists, try a new tape. If this too fails, consult your Commodore dealer.

A copy of the program MATH has been written on the tape. The original program is still in the computer memory. Type LIST and the program will be displayed on the screen. Now enter the command NEW and then LIST; the program has been erased and the memory is clear.

To load the program back into the computer from the tape, first rewind the tape. The LOAD command looks like this:

```
LOAD "program name"
```

For the MATH program enter:

```
LOAD "MATH"
```

The computer will respond with PRESS PLAY ON TAPE. Carry out that instruction and the screen will go blank while the computer searches for the program called MATH. When it finds the program the screen will display:

```
FOUND MATH
LOADING MATH
READY
```

If there is more than one program on the tape, and the one you want is not first, then the screen will display:

```
FOUND program name
```

Where the program name is the first program on the tape, there will be a short pause before the screen goes blank and the computer continues the search. It will continue searching and telling you which programs it has found until it finds the program requested. This program will then be loaded into the computer memory.

If you type in the command LOAD but no program name, the computer will search and then load the first program it finds on the tape.

The command LOAD puts a copy of the program into the computer memory. The program remains on the tape.

One problem with storing programs on tape is knowing where the last program is on the tape so that new programs are not stored on top of existing programs. The BASIC command VERIFY can be used to eliminate this problem. Before saving a program, rewind the tape and enter the command VERIFY with the program name for the new program you want to write on the tape. The computer will search through the programs on the tape and if it doesn't find the new program it will send a VERIFY ERROR message to the screen. You can ignore this error message and save your new program. It will be on the tape immediately after the last program saved.

Another way of knowing where the programs are on the tape is to create a written directory. It should include the program name and the numbers shown on the recorder's counter at the beginning and end of the SAVE operation. Then you can press the 'fast forward' or 'rewind' keys to position the tape at the start of the program to be loaded.

Questions & Answers

The three BASIC commands for the C2N data cassette are _____, _____ and _____.

Answer: SAVE, LOAD and VERIFY

Saving Programs Using a Disk Drive

If you do not have a disk drive you may want to skip this section, which covers the input/output commands to the 1541 single drive floppy disk. The disk drive has its own small computer, and there is a set of Disk Operating System (DOS) commands that control input and output to the disk drive.

To save a program using the disk drive, you will need a 5 1/4 inch soft sector

format diskette. The disk drive and the computer should always be turned ON before the diskette is inserted in the drive.

The first command to the disk drive should always be an OPEN command. This will open a logical input/output file so that data can be transferred between the computer and the disk drive. The OPEN command for the disk drive looks like this:

```
OPEN 15,8,15
```

Fifteen is the logical file number and eight is the device number. All the peripheral devices have device numbers and the number for the first disk drive is always eight. If you intend to use more than one disk drive with the Commodore 64, then you will need to change the device number for the second drive. The device number for the second drive can be 9, 10 or 11. This is a software adjustment. Commodore has published the program to do this on the Commodore Test/Demo disk, available from your Commodore dealer.

Formatting a Diskette

The Commodore 1541 disk drive uses only SOFT-SECTORED diskettes. This is because the disk operating system divides the tracks on the diskette into sector's and blocks. A HARD-SECTORED diskette has sectors already formed on it - do not use this kind.

Figure 2-2 shows that each diskette is divided into 35 TRACKS. As you can see, inside tracks have fewer sectors than outside ones. The sectors are a standard length, 256 bytes. The maximum storage capacity for one diskette 176,640 bytes. There will be more information about bytes in Chapter 7.

The DIRECTORY of all the programs stored on the diskette and the Block Availability Map, BAM, showing the space available on the diskette, are always stored on track 18. Most of the time the programmer is unaware of this organization because these 'housekeeping chores' are controlled by the computer and the disk drive.

Before the diskette can be used, it must be formatted. This should be done only once, when the diskette is first used. Formatting will erase any programs that are on the diskette. The test diskette you used is already formatted. There are two ways to format a disk. If you have already used an OPEN statement, then use:

```
PRINT#15, "N0: filename, identifier"
```

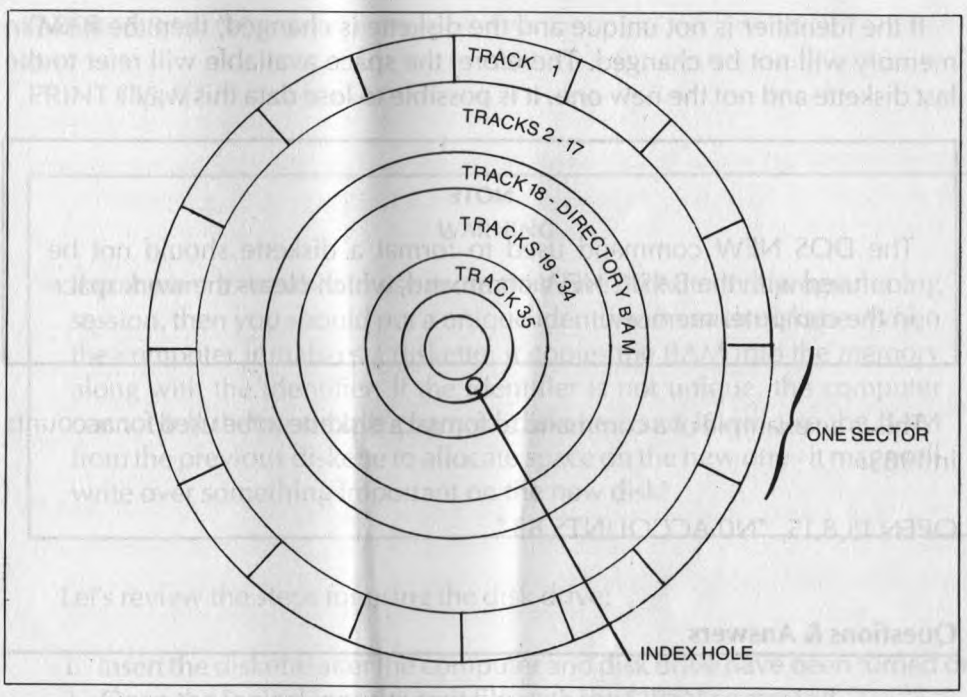


Figure 2.2 Diskette track and sector organization.

PRINT #15 is a BASIC statement to write information to file number 15. File number 15 is the logical input/output file opened in the OPEN statement.

If you have not yet opened your files, then this command will both open files and initialize a new diskette:

OPEN 15,8,15, "N0: filename, identifier"

- In both statements the N stands for NEW.
- The 0 is the disk drive number.
  If you have only one disk drive the default number is zero and may be omitted. If you are using more than one disk drive then the drive number must be included.
- The filename is a name for the diskette, and it may be any name you choose but not more than sixteen characters.
- The identifier is unique two-character alphanumeric identifying sequence for the diskette.

If the identifier is not unique and the diskette is changed, then the BAM in memory will not be changed. Therefore, the space available will refer to the last diskette and not the new one. It is possible to lose data this way!

> **NOTE**
>
> The DOS NEW command used to format a diskette should not be confused with the BASIC NEW command, which clears the work space in the computer memory.

This is an example of a command to format a diskette to be used for accounts in 1983:

```
OPEN 15,8,15, "N0:ACCOUNTS,83"
```

Questions & Answers

Which of the following statements is correct?

(a) PRINT#15, "N:HOUSEHOLD ACCOUNTS,83"
(b) PRINT#15, "N:ACCOUNTS,1983"
(C) OPEN 15,8,15,"N0:ACCOUNTS,A1"

Answer:
(c) is correct.
(a) The filename is more than 16 characters.
(b) The identifier is more than two characters.

If you are using more than one diskette in a programming session, then you should use the INITIALIZE command when you swap disks. This will ensure that you have the correct directory and BAM in memory. It is a good idea to make a habit of initializing a diskette before you start to use it. The initialize command can be used with an OPEN command or a PRINT#15 statement. It looks like this:

OPEN 15,8,15, "I"
or
PRINT #15, "I"

WARNING

If you want to be able to use more than one diskette in a programming session, then you should put a unique identifier on each diskette. When the computer initializes a diskette, it copies the BAM into the memory along with the identifier. If the identifier is not unique, the computer cannot tell that you have changed diskettes, and it will use the BAM from the previous diskette to allocate space on the new one - it may well write over something important on the new disk!

Let's review the steps for using the disk drive:

1. Insert the diskette after the computer and disk drive have been turned on.
2. Open the logical input/output file with the OPEN command.
3. Format a diskette with the NEW command, if necessary.
4. Initialize a diskette every time you use it.

Questions & Answers

Now match the following:

(a) OPEN	(1) Formats a diskette.
(b) Initialize	(2) Writes to a logical file.
(c) New	(3) Opens a logical file.
(d) PRINT #	(4) Loads the BAM into memory.

Answer: (a) 3, (b) 4, (c) 1, (d) 2.

Always check that the red LED Light on the front of the disk drive goes off at the end of each operation. It should only be lit while the operation is being executed. If it remains on and flashing, then the disk drive did not complete the

operation. This is a DOS, Disk Operating System, error. An error message will not always appear on the screen, so it is important to check the red light. Here are some of the most common error messages for the disk drive:

SYNTAX ERROR - There is an incorrect command, usually created by a typing error or a command that is more than 58 characters.
FILE NOT FOUND - The file or program requested is not on the disk. Check that you have the correct disk in the drive and that the name is spelled correctly.
FILE NOT OPEN - The logical file is not open. Use an OPEN command.
FILE OPEN - The OPEN command was sent when the file was already open.
DISK ID MISMATCH - The diskette was not properly initialized when it was inserted in the drive.

None of these DOS commands will erase a program in the computer memory: OPEN, Initialize, NEW, and PRINT#. We can change disks at any time, remembering to initialize the new one.

Disk Commands

The DOS command SAVE is used to save a program on a diskette.

SAVE "program name,"8

The program name may be anything the programmer chooses, but not longer than sixteen characters, including spaces.

Notice the eight at the end of the command. This is the device number. This is the second DOS command that requires a device number.

If the device number is omitted then the computer assumes that you want it to be 1, the device number for a cassette recorder. The screen will show the message PRESS RECORD AND PLAY ON TAPE. If this happens, you can escape by using the RUN STOP key.

To save the MATH program, send the following command:

SAVE "MATH,"8

This will cause a copy of the program in memory to be written on the diskette.

If the program that you want to SAVE is a revision of a program already on the

disk, then an '@' before the name will save the revised version of the program. The command will look like this:

SAVE"@:program name,"8

For example:

SAVE"@:MATH,"8

This is a very useful addition to the SAVE command, and one that you may need to use frequently as you develop your own programs.

To be sure that a SAVE command has occurred accurately, we can use another DOS command to VERIFY the program:

VERIFY "MATH," 8

This command makes a comparison between the MATH program on the diskette and the program in the computer memory.

A shorthand way of writing this command is:

VERIFY "*," 8

This causes the last program saved to be verified.

When the VERIFY command is sent the screen displays:

SEARCHING FOR program name
VERIFYING

If the verification operation is successful, the READY prompt appears. Otherwise, one of these error messages is sent to the screen.

FILE NOT FOUND - The program with that name is not on the diskette.
VERIFY ERROR - The program saved does not match the program in memory.

If there is an error, make sure the program is still in memory with the LIST command. Then save and verify again.

You cannot abbreviate the VERIFY command to V. V is the abbreviation for another DOS command, VALIDATE. The validate command reconstructs the

BAM. It traces every block on the diskette and deletes files that are not closed or have other errors. It looks like this:

PRINT#15, "V"
or
PRINT#15, "VALIDATE"

Once the MATH program has been saved and verified on the diskette, we can use the BASIC NEW command to erase the program from the computer memory.

Questions & Answers

Which BASIC command would you use to check that the program has been erased?

Answer: LIST or RUN

Now the LOAD command can be used to retrieve the program from the diskette. The LOAD command is as follows:

LOAD "program name,"8

Questions & Answers

Can you write the command to load the MATH program?

Answer: LOAD "MATH,"8

The screen will show:

SEARCHING FOR MATH
LOADING MATH
READY

To check that the LOAD command has been executed satisfactorily, either RUN or LIST the program.

The Disk Directory

Now that we have a program stored on the disk, we can check the directory. The directory has the name $. Here is the command to load the directory:

```
LOAD "$",8
```

When the directory has been loaded, it can be listed to the screen using the LIST command.

Figure 2.3 shows the listing of a directory. The top line shows the disk name and disk identifier. The individual programs are listed by name, including their block size and program type. At the end is the number of blocks that are unused, or free, on the diskette. All this information is stored on track 18 of the diskette and is written and updated by the Disk Operating System.

```
READY.

0 [redacted]
 2A
1    "ADD"              PRG
1    "SUBTRACT"         PRG
2    "BIRTHDAY"         PRG
5    "CINDERELLA"       PRG
2    "ALPHASORT"        PRG
653 BLOCKS FREE.

READY.
```

Figure 2.3 Printout of a disk directory listing

WARNING

Loading the directory will erase any program that is in memory.

Before we return to the MATH program, there is one more disk command that we should discuss. To remove a program from the disk, use the SCRATCH command. It looks like this:

```
PRINT #15,"SCRATCH 0:program name"
```

It can be abbreviated:

```
PRINT#15,"S0:program name"
```

More Short Programs

The rest of this chapter will refer to both disk and cassette use.

Now back to the MATH program. It was designed to add variable numbers and to print the answer on the screen. An INPUT statement was used to input the two numeric variables. String or alphanumeric variables can also be included in an INPUT statement. Here is another short program that will store a name and print a message that includes the name on the screen.

Before you type in these statements be sure to clear the workspace in the computer memory.

```
10 INPUT "YOUR NAME"; N$
20 PRINT "HELLO";N$
```

Enter these statements and RUN the program. In this program N$ is the string variable name. The variable name could have been two characters long, but in this case, choosing the first letter of "name," the data it will store, makes sense. As programs become longer, it is a good policy to choose variable names that contain some clue about the data they contain. (Remember that you can use longer names, but be sure that the first two characters of each name are different.)

Questions & Answers

Anything that is between quotation marks will be printed exactly as it is on the screen. What happens if there is not a space before the closing quotation mark in "HELLO"?

Answer: The computer does not leave space between HELLO and the name.

How would you change statement 20 to print a different message to the person?

Answer: Change the message inside quotes.

Can you save this program on a diskette or a tape? If you need help, refer to the earlier sections in this chapter.

It is possible to have more than one kind of variable in a program. Let's add to the program so that it will calculate a person's age in the year 2000.

First we need to add an INPUT statement to ask for the person's age.

Questions & Answers

What kind of variable will the person's age be?

Answer: Numeric - a variable that has only numbers.

We can use an INPUT statement like this:

```
30 INPUT "YOUR AGE"; A
```

A, the first letter of "age," is used as the variable name.

The next statement will calculate the age in the year 2000.

```
40 Y = A + 2000-1984
```

Questions & Answers

Where is the age in the year 2000 stored?

Answer: In the numeric variable Y.

Now we need to add a last statement that will print the result to the screen.

```
50 PRINT N$; " in the year 2000 you will be "; Y
```

Type this statement carefully, so that the spacing between the words is correct.

This program will work very well and accurately as long as the year in question is 1984. The program could be changed to include another INPUT statement that will provide an accurate calculation for any year up to 2000.

Questions & Answers

Can you make the changes?

Answer:
Here is my version of the program:

```
10 INPUT "YOUR NAME"; N$
20 PRINT "HELLO"; N$
30 INPUT "YOUR AGE"; A
40 INPUT "WHAT IS THE CURRENT YEAR"; C
50 Y = A + 2000 - C
60 PRINT N$; " IN THE YEAR 2000 YOU WILL BE "; Y
```

Here is another problem that will require both numeric and string variables. Ask someone what his favorite snack is and how much he eats in one week. Calculate how much he will consume in one year and print it on the screen.

Questions & Answers

How many variables do you need to input?

Answer: Two: the favorite snack and how much is eaten in one week.

Write two INPUT statements to do this.

Answer:

```
10 INPUT "WHAT IS YOUR FAVORITE SNACK"; F$
20 INPUT "HOW MANY DO YOU EAT IN ONE WEEK"; W
```

Now write the calculation statement.

Answer:
Mine looks like this:

```
30 Y = W * 52
```

Statements 10 and 20 are the input. Statement 30 is the process, so the last part of the program will be _____ ?

Answer: Output

Now write the output statement to include the amount eaten in one year and the name of the snack.

Answer:

Here is mine:

40 PRINT "YOU WILL EAT ";Y; F$; " IN ONE YEAR"

Now RUN Tthe program and see if it works.

There is no limit to the number of variables that can be included in the program. The computer will store them all away, each at a unique address, until they are called up later in the program. This makes it possible to write programs that personalize stories for yourself, your children or your friends.

The program shown in the listing below contains some features that have not previously been explained.

Here is a variation of a popular fairy tale:

```
10 INPUT" A NOUN ";N1$
20 INPUT" AN ADJECTIVE";A1$
30 INPUT" ANOTHER ADJECTIVE";A2$
40 INPUT" ANOTHER NOUN";N2$
50 INPUT" AN ADVERB";D1$
60 INPUT" ANOTHER ADJECTIVE";A3$
70 INPUT" A PLURAL NOUN";N3$
80 INPUT" ANOTHER NOUN";N4$
90 INPUT" A VERB";V1$
100 INPUT" ANOTHER ADVERB";D2$
110 INPUT"ANOTHER ADJECTIVE";A4$
120 INPUT"ANOTHER VERB";V2$
130 INPUT" ANOTHER NOUN";N5$
140 INPUT" ANOTHER ADJECTIVE";A5$
150 INPUT" ANOTHER VERB";V3$
160 INPUT" ANOTHER ADJECTIVE";A6$
170 PRINT"□"
```

(continued)

```
180 PRINT TAB(14)"A FAIRY TALE"
190 PRINT
200 PRINT"ONCE UPON A TIME THERE WAS A LITTLE GIRL CALLED CINDERELLA"
210 PRINT"SHE SAT AMONG THE "N1$" CRYING. SUDDENLY HER "A1$" GODMOTHER ";
220 PRINT"APPEARED.WHEN THE "A2$" GODMOTHER KNEW THAT SHE WANTED " D1$;
230 PRINT" TO GO TO THE BALL. SHE TOOK A LARGE "N2$" AND TURNED IT INTO A ";
240 PRINT" COACH.THEN SHE TURNED SIX "A3$" "N3$" INTO HORSES AND A BIG "N4$;
250 PRINT" SHE "V1$" INTO A COACHMAN. OF COURSE CINDERELLA "D2$" NEEDED A ";
260 PRINT A4$" DRESS FOR THE BALL. HER FAIRY GODMOTHER "V2$" HER MAGIC "N5$;
270 PRINT" AND CINDERELLA'S RAGS WERE CHANGED TO A "A5$" DRESS. "
280 PRINT"YOU KNOW THE REST OF THE STORY, CINDERELLA "V3$" THE "A6$" PRINCE!"
```

Statement 170 is one way that the screen can be cleared in the middle of a program. This statement will allow the story to be printed on a new screen. To make the reversed heart symbol, hold down the shift and the CLR HOME key at the same time.

Statement 180 is a PRINT statement for the title of the story. The TAB function is similar to the tab on a typewriter. The screen is forty columns wide and the title is twelve characters long. So, to center the title, the TAB function needs to put the first character of the title in position 14.

Statement 190 is an 'empty' PRINT statement which will leave one blank line between the title and the text of the story.

Statements 200-280 will print the story, including the words input at the beginning of the program. PRINT statements cannot be longer than two lines, or they will trigger a syntax error.

The semicolon at the end of each PRINT statement will make the printout on the screen continuous. Because the length of the words input at the beginning of the program may vary, there is no easy way to control the length of the line; some words may be split between lines.

If you decide to write your own version of this program, remember that the screen holds only twenty four lines of print. If your story is longer than this, the screen will start to scroll up and may move too fast to be read.

Questions & Answers

Which of the following statements will print CINDERELLA in the center of the screen?

(a) 100 PRINT TAB(10) "CINDERELLA"
(b) 100 PRINT TAB (15) "CINDERELLA"
(c) 100 PRINT TAB (20) "CINDERELLA"

Answer: (b) CINDERELLA is 10 characters long and the screen is 40 characters wide, so the TAB must start printing at position 15.

Using a Printer

If you do not have a printer, please skip to the Self-Check at the end of the chapter - but DO NOT skip that!

When this kind of program is being used by a group, it is fun to send the output to the Commodore 1525 Printer so that everyone can have a 'hard copy' of their run to keep. To do this, the program needs to switch from printing on the screen for the INPUT statements to printing on the printer for the story. A statement needs to be inserted before statement 180 to transfer the output. Here it is:

```
174 OPEN 3,4
```

Statement 174 opens a logical input/output file between the computer and the printer. The logical file number is 3 and the device number for the printer is 4. Now all the PRINT statements need to be changed to PRINT # 3 so that the story will be sent to the printer.

Returning the control to the screen at the end of the program requires two statements:

```
290 PRINT # 3
300 CLOSE 3
```

The PRINT # 3 statement sends a blank line as a signal to the device, and the CLOSE 3 statement closes the logical file and returns the control to the screen.

Here is the program listing with all the changes for using a printer:

```
10 INPUT" A NOUN ";N1$
20 INPUT" AN ADJECTIVE";A1$
30 INPUT" ANOTHER ADJECTIVE";A2$
40 INPUT" ANOTHER NOUN";N2$
50 INPUT" AN ADVERB";D1$
60 INPUT" ANOTHER ADJECTIVE";A3$
70 INPUT" A PLURAL NOUN";N3$
80 INPUT" ANOTHER NOUN";N4$
90 INPUT" A VERB";V1$
100 INPUT" ANOTHER ADVERB";D2$
110 INPUT"ANOTHER ADJECTIVE";A4$
120 INPUT"ANOTHER VERB";V2$
130 INPUT" ANOTHER NOUN";N5$
140 INPUT" ANOTHER ADJECTIVE";A5$
150 INPUT" ANOTHER VERB";V3$
160 INPUT" ANOTHER ADJECTIVE";A6$
170 OPEN3,4
180 PRINT#3,TAB(14)"A FAIRY TALE"
190 PRINT#3,
200 PRINT#3,"ONCE UPON A TIME THERE WAS A LITTLE GIRL CALLED CINDERELLA"
210 PRINT#3,"SHE SAT AMONG THE "N1$" CRYING. SUDDENLY HER "A1$" GODMOTHER ";
220 PRINT#3,"APPEARED.WHEN THE "A2$" GODMOTHER KNEW THAT SHE WANTED " D1$;
230 PRINT#3," TO GO TO THE BALL. SHE TOOK A LARGE "N2$" AND TURNED IT INTO A ";
240 PRINT#3," COACH.THEN SHE TURNED SIX "A3$" "N3$" INTO HORSES AND A BIG "N4$;
250 PRINT#3," SHE "V1$" INTO A COACHMAN. OF COURSE CINDERELLA "D2$" NEEDED A ";
260 PRINT#3,A4$" DRESS FOR THE BALL. HER FAIRY GODMOTHER "V2$" HER MAGIC "N5$;
270 PRINT#3," AND CINDERELLA'S RAGS WERE CHANGED TO A "A5$" DRESS. "
280 PRINT#3,"YOU KNOW THE REST OF THE STORY, CINDERELLA "V3$" THE "A6$" PRINCE!"
300 CLOSE 3
```

The same printer command statements can be used with the LIST command to send a listing of a program to the printer. This is the sequence of commands. Notice that they do not require statement numbers because they are being used in the direct mode.

```
OPEN 3,4
CMD 3
LIST
PRINT # 3
CLOSE 3
```

All of these commands, except CMD3, were used in the Fairy Tale program. CMD3 is a command to switch any output to the printer instead of to the

screen, until PRINT # 3 and CLOSE statements are received.

A good way to keep track of the programs on a diskette is to load the directory and then send the listing to the printer. The listing can then be taped to the envelope of the diskette.

Questions & Answers

Put the following commands in the correct order to transfer output control to the printer. Then return the control to the screen.

_____ (a) OPEN 3,4
_____ (b) PRINT # 3
_____ (c) CMD 3
_____ (d) CLOSE 3

Answer: 1 (a), 2 (c), 3 (b), 4 (d).

Before moving on to the next chapter, please try the following Self-Check.

SELF-CHECK

1. Use paper and pencil to solve the following problems, and then check your answers on the computer using PRINT in the direct mode.
   (a) 7* +10
   (b) 7*(3 +1)
   (c) (7*(3 +10))/5
   (d) 7*3 +10/5
   (e) 7*(3 +10)/5
2. What will you see on the screen when you enter the following statements?
   (a) PRINT ALONE AT LAST!
   (b) 10 PRINT "ALONE AT LAST!"
   (c) PRINT "ALONE AT LAST!"
3. What are the three parts of any program?

4. Match the following:

(a) RUN	(1) Print the program on the screen.
(b) LOAD	(2) Store the program on tape or disk.
(c) LIST	(3) Execute the program.
(d) NEW	(4) Retrieve a program from a tape or disk.
(e) SAVE	(5) Clear the computer memory.

5. When do we put a $ after a variable name?
6. Write an INPUT statement to input phone numbers.
7. Write a statement to save a program called 'CHECKERS.' Write it for either a cassette or a disk, whichever one you use.
8. Write a program to do the following:
   (a) Input a person's name.
   (b) Input his/her age.
   (c) Input his/her birthdate.
   (d) Print a birthday message to the person including the name, age next birthday, and the date.

Answers to SELF-CHECK

From now on, the answers to the Self-Check are not necessarily the only possible answers. Once you start writing your own programs, your creations are as valid as mine.

1. (a) 31
   (b) 91
   (c) 18.2
   (d) 23
   (e) 18.2

2. (a) 0
   ?SYNTAX ERROR

There are no quotation marks around the message. The computer did its best and looked to see if there was a value in the numeric variable AL - that's the 0 - and then it printed an error message to signal its confusion!

   (b) READY This is a program and needs to be RUN.
   (c) ALONE AT LAST! This is a direct statement.

3. Input, Process, Output
4. (a) 3, (b) 4, (c) 1, (d) 5, (e) 2.
5. A '$' indicates a string variable, one that stores alphanumeric information.
6. 10 INPUT "Phone number"; P$
   (Note that I used a string variable because many people use parentheses and dashes in phone numbers.)
7. Cassette - SAVE"CHECKERS" Disk - SAVE"CHECHERS",8
8. This is my version:
   10 INPUT "YOUR NAME";N$
   20 INPUT "HOW OLD ARE YOU";A
   30 INPUT"WHEN IS YOUR BIRTHDAY";B$
   40 NA = A +1
   50 PRINT "HAPPY BIRTHDAY " "N$" NEXT "B$" YOU WILL BE "NA

Suggestions for More Programming with Your New Skills

1. SAVE every program you write even if it isn't a masterpiece, so that you become used to moving programs between the computer and the storage device.
2. Try sending the output from your program to the printer or making a record of your programs by printing out the listing.
3. Write a newsletter for the family, school or club that will include people's names and other variables that you think of.
4. Write a program that will take the amount of ingredients for a recipe and tell you how much you will need for various numbers of guests. For instance, if there are 50 Boy Scouts, how many packages of hot dogs will you need? If your grandmother's recipe for Chicken Supreme for 4 people calls for 2 cups of butter, how many cups will you need when eight relatives come to dine?

CHAPTER 3

Flowcharts

The programs we have written so far have been simple and straightforward. Given all of the things you can do with a Commodore 64 - make music, draw pictures, design games, etc. - it is tempting to just sit at the keyboard and start writing programs, planning as you go. This approach, however, will lead to hours of frustration as you try to "debug" (find errors in) a program that took only a few minutes to put together. In this chapter you will learn how to reduce the time spent debugging a program by creating a flowchart with paper and pencil before you turn on the computer.

By the end of the chapter you will be able to:

- Recognize six flowcharting symbols.
- Prepare an idea for a flowchart.
- Draw a flowchart for a program idea including decisions and repeats.

Drawing Flowcharts

A flowchart is a "picture" of a program and how it operates. It shows, in order, all the operations to be performed in the input, process and output parts of the program. You can buy a flowcharting template to draw the symbols, but this is not essential because none of them are particularly difficult. Figure 3.1 shows the symbols we will be using in this chapter. Other symbols will be introduced as needed in later chapters. Appendix B has a complete listing of all the symbols.

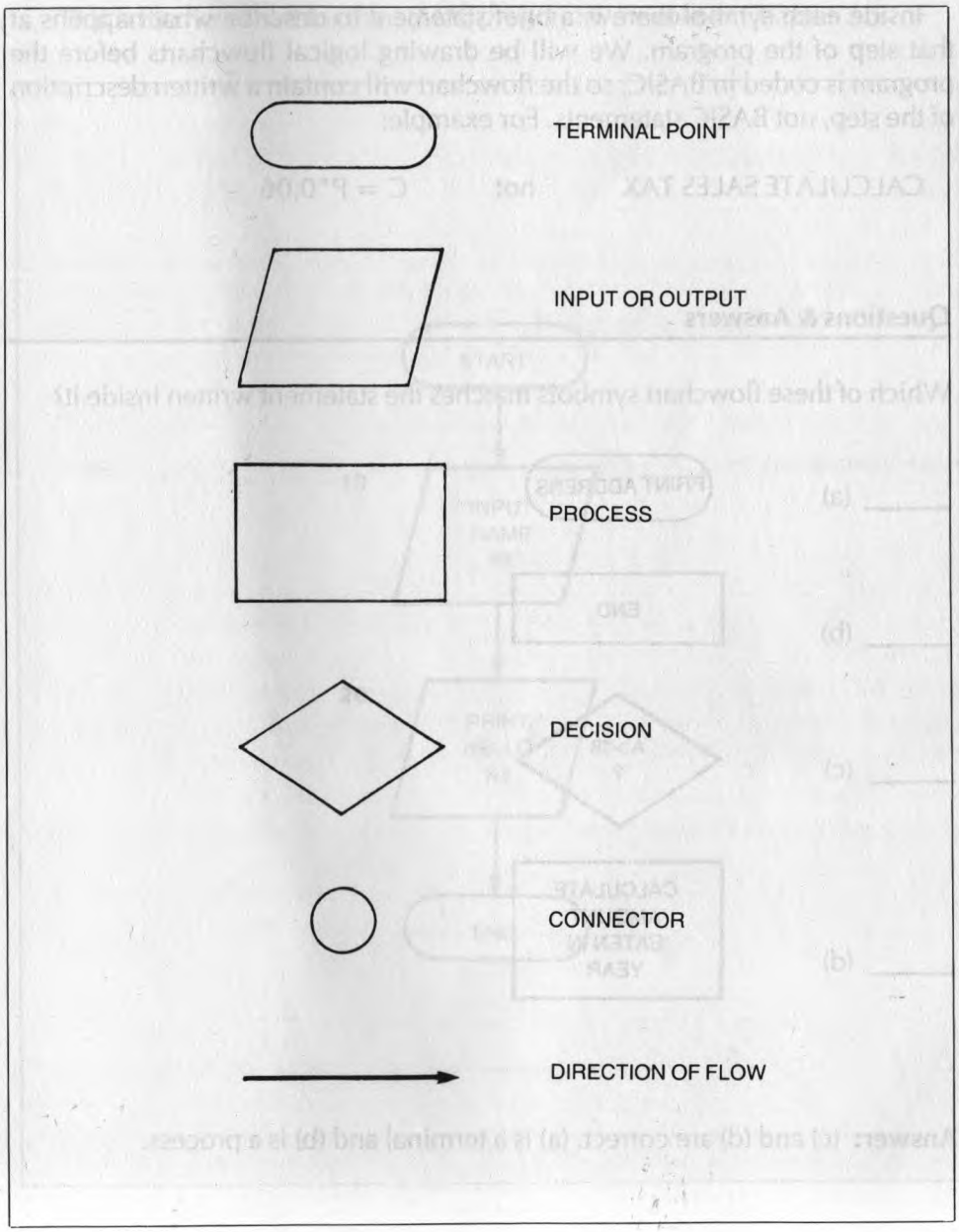


Figure 3.1 Flowchart Symbols

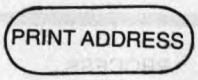
Inside each symbol there is a brief statement to describe what happens at that step of the program. We will be drawing logical flowcharts before the program is coded in BASIC, so the flowchart will contain a written description of the step, not BASIC statements. For example:

CALCULATE SALES TAX not C = P*0.06

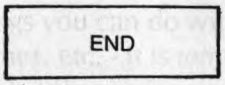
Questions & Answers

Which of these flowchart symbols matches the statement written inside it?

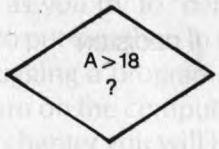
_____ (a)



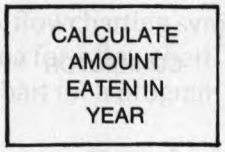
_____ (b)



_____ (c)

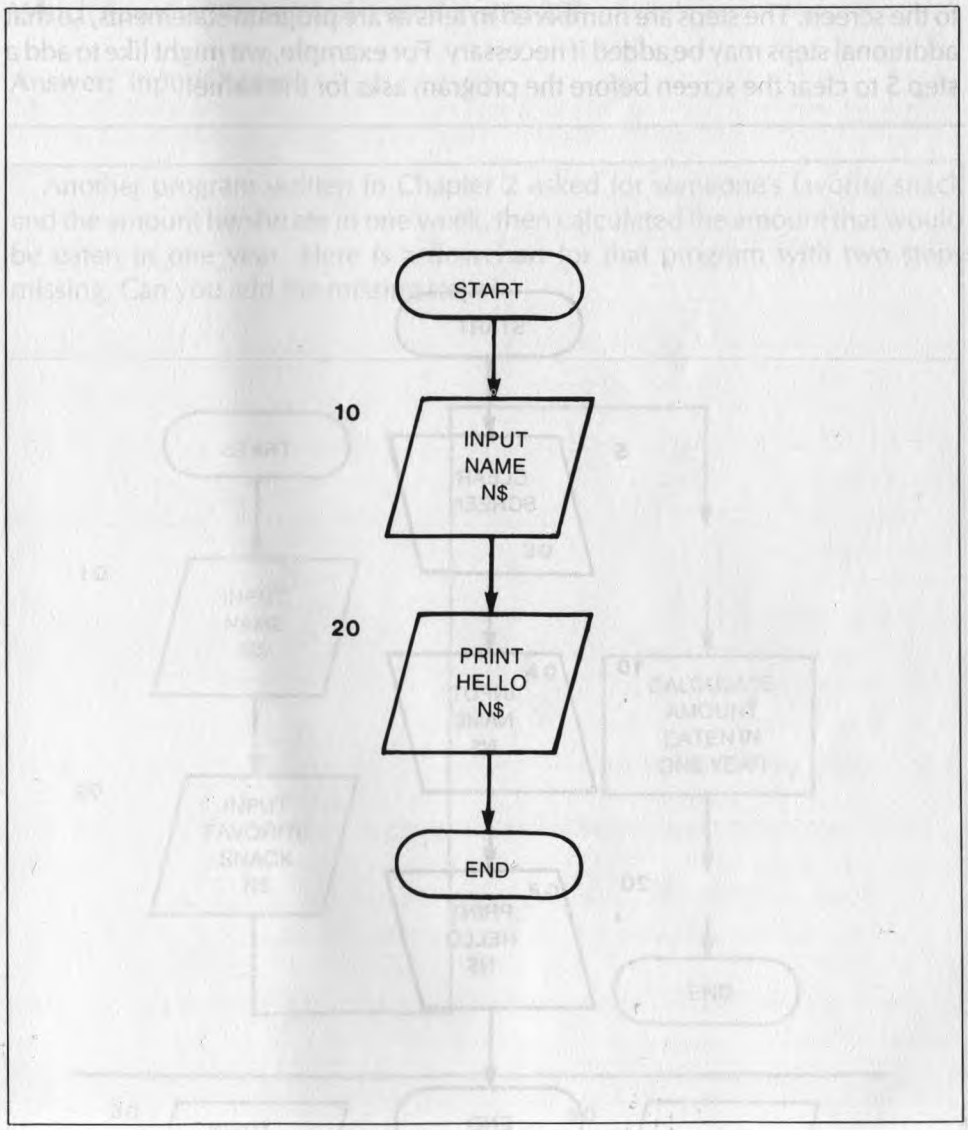


_____ (d)

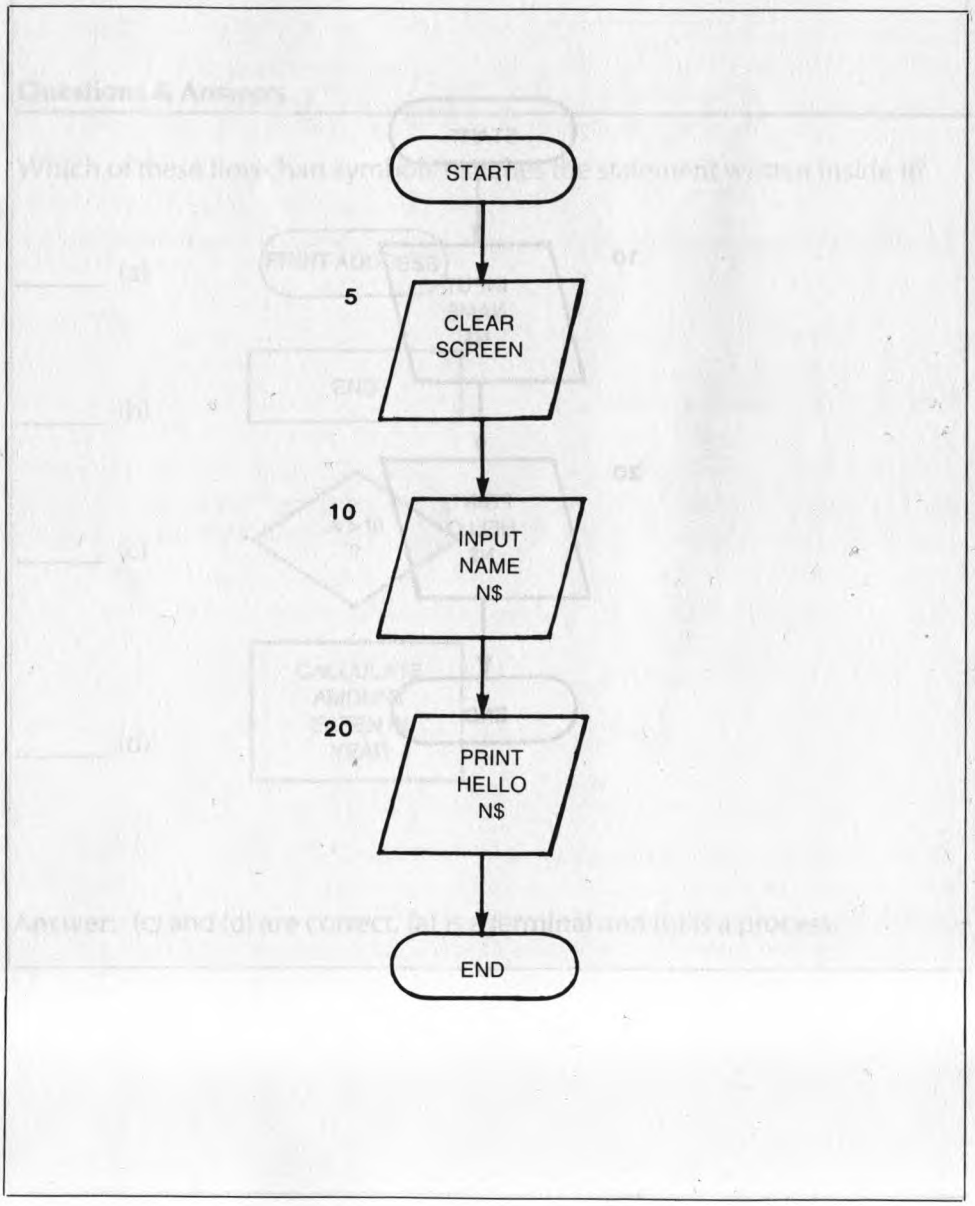


Answer: (c) and (d) are correct. (a) is a terminal and (b) is a process.

This is the flowchart for the NAME program we wrote in Chapter 2.



In this program there are two steps, excluding the two terminal points (start and end). The first step inputs information and the second step prints a message to the screen. The steps are numbered in tens as are program statements, so that additional steps may be added if necessary. For example, we might like to add a step 5 to clear the screen before the program asks for the name.

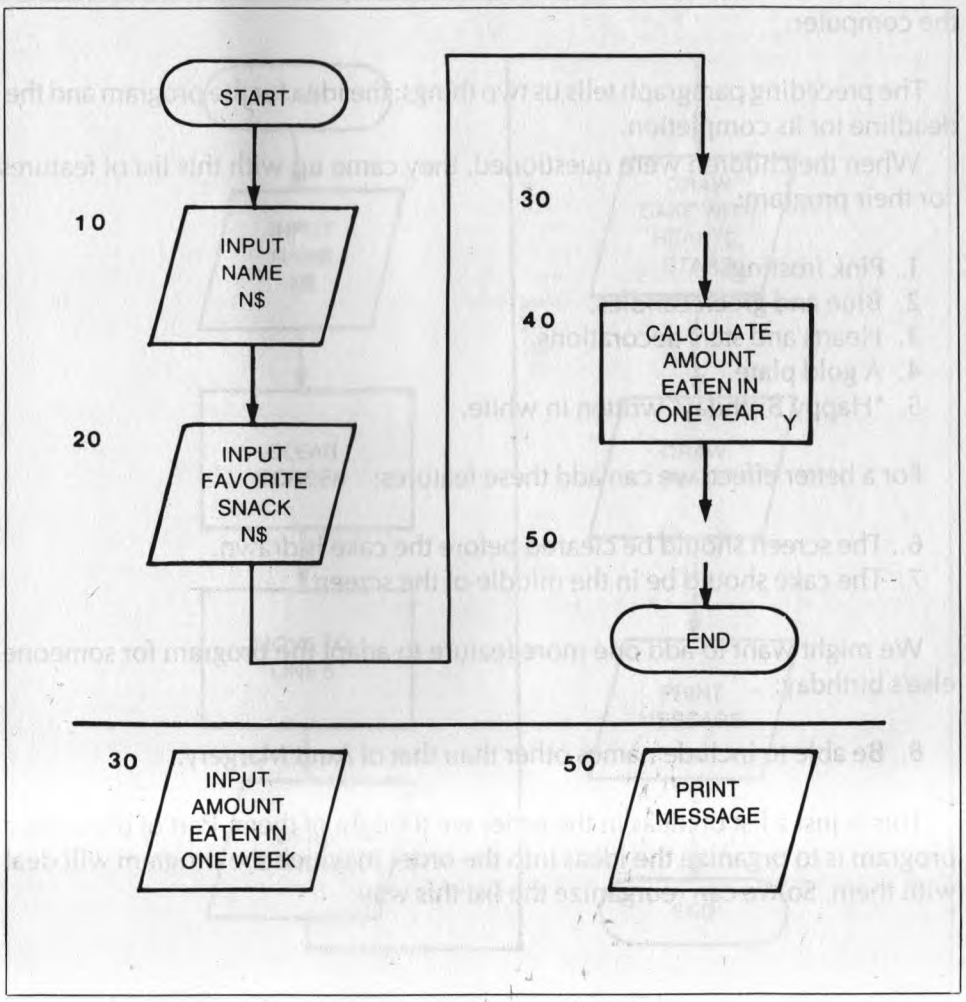


Questions & Answers

Which flowcharting symbol is used in step 20?

Answer: Input/Output

Another program written in Chapter 2 asked for someone's favorite snack and the amount he/she ate in one week, then calculated the amount that would be eaten in one year. Here is a flowchart for that program with two steps missing. Can you add the missing steps?



These flowcharts have been for programs already written, but the main value of a flowchart is in the preparation of a new program.

Planning a Flowchart

Even before the flowchart is drawn there is usually some planning to be done. This may involve writing a short outline of the goal of the program and possibly a list of the features to be included. Developing the practice of putting ideas down on paper helps to organize the logic and is especially helpful in sequencing larger programs. Here is an example:

Aunt Margery is coming to stay next Friday, her birthday. The children think that it would be a neat idea to welcome her with a picture of a birthday cake on the computer.

The preceding paragraph tells us two things: the idea for the program and the deadline for its completion.

When the children were questioned, they came up with this list of features for their program:

1. Pink frosting.
2. Blue and green candles.
3. Hearts and stars decorations.
4. A gold plate.
5. "Happy Birthday" written in white.

For a better effect, we can add these features:

6. The screen should be cleared before the cake is drawn.
7. The cake should be in the middle of the screen.

We might want to add one more feature to adapt the program for someone else's birthday:

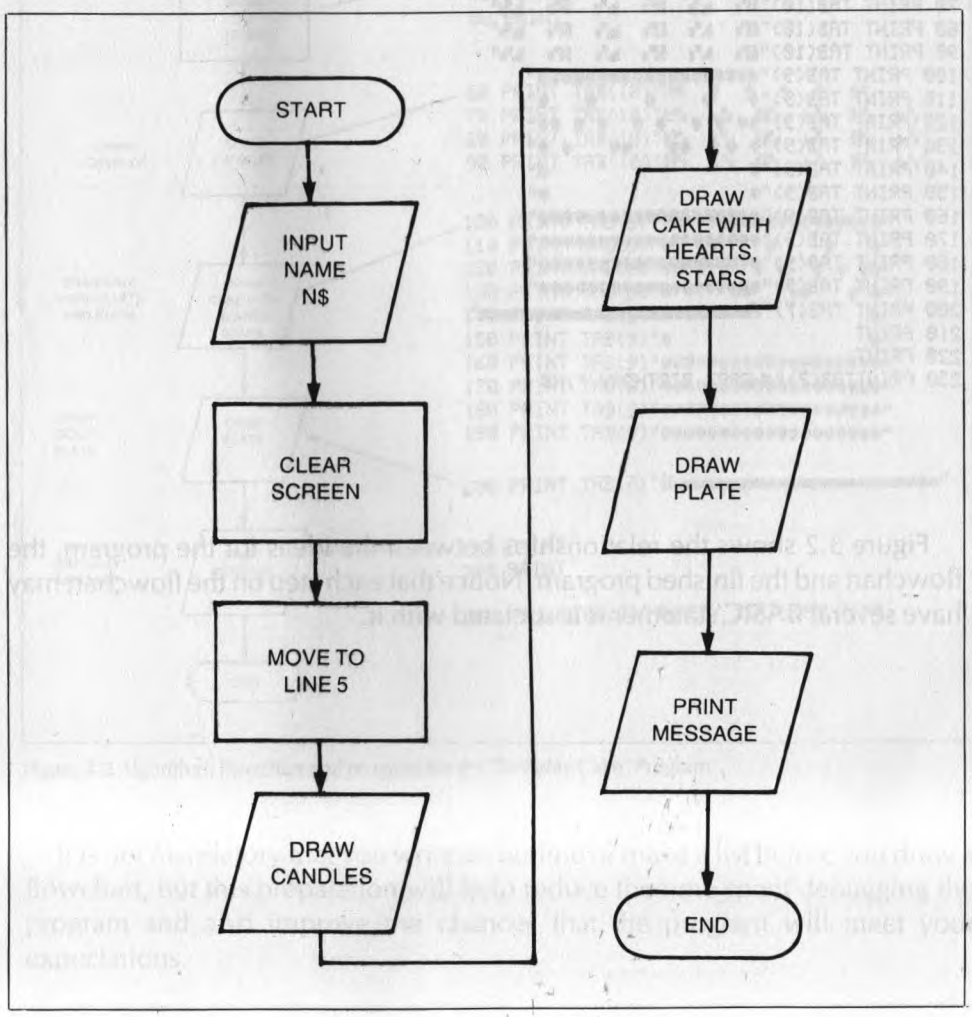
8. Be able to include names other than that of Aunt Margery.

This is just a list of ideas in the order we thought of them. Part of planning a program is to organize the ideas into the order in which the program will deal with them. So we can reorganize the list this way:

1. Input the birthday person's name.
2. Clear the screen.
3. Move to the correct place on the screen to begin drawing the cake.
4. Draw the candles.
5. Draw the cake with hearts and stars.
6. Draw the gold plate.
7. Print the birthday message.

Of course, this isn't the only order in which the picture could be drawn, but it is the one we'll use.

Each of these steps can now be represented in a flowchart.



The program is written using the CTRL or Commodore key with the color keys to change colors. The screen is cleared using the SHIFT and CLR HOME keys. Here is a listing of the program and a printout of the cake (sorry, no color!).

```
10 INPUT"NAME";N$
20 PRINT"■"
30 PRINT
40 PRINT
50 PRINT
60 PRINT TAB(10)"■■  ♦  ♦  ♦  ♦  ♦"
70 PRINT TAB(10)"▐▌ ▚▞ ▐▌ ▚▞ ▐▌ ▚▞"
80 PRINT TAB(10)"▐▌ ▚▞ ▐▌ ▚▞ ▐▌ ▚▞"
90 PRINT TAB(10)"▐▌ ▚▞ ▐▌ ▚▞ ▐▌ ▚▞"
100 PRINT TAB(9)"♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦"
110 PRINT TAB(9)"♦   ♦    ♦    ♦   ♦"
120 PRINT TAB(9)"♦♦ ♦ ♦  ♦ ♦  ♦ ♦ ♦♦"
130 PRINT TAB(9)"♦ ♦   ♦♦   ♦♦   ♦ ♦"
140 PRINT TAB(9)"♦                 ♦"
150 PRINT TAB(9)"♦                 ♦"
160 PRINT TAB(9)"♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦"
170 PRINT TAB(9)"♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦"
180 PRINT TAB(9)"♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦"
190 PRINT TAB(9)"♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦"
200 PRINT TAB(7)"■■■■■■■■■■■■■■■■■■■■■■■■■■■"
210 PRINT
220 PRINT
230 PRINTTAB(7)"■HAPPY BIRTHDAY ";N$
```

Figure 3.2 shows the relationships between the ideas for the program, the flowchart and the finished program. Notice that each step on the flowchart may have several BASIC statements associated with it.

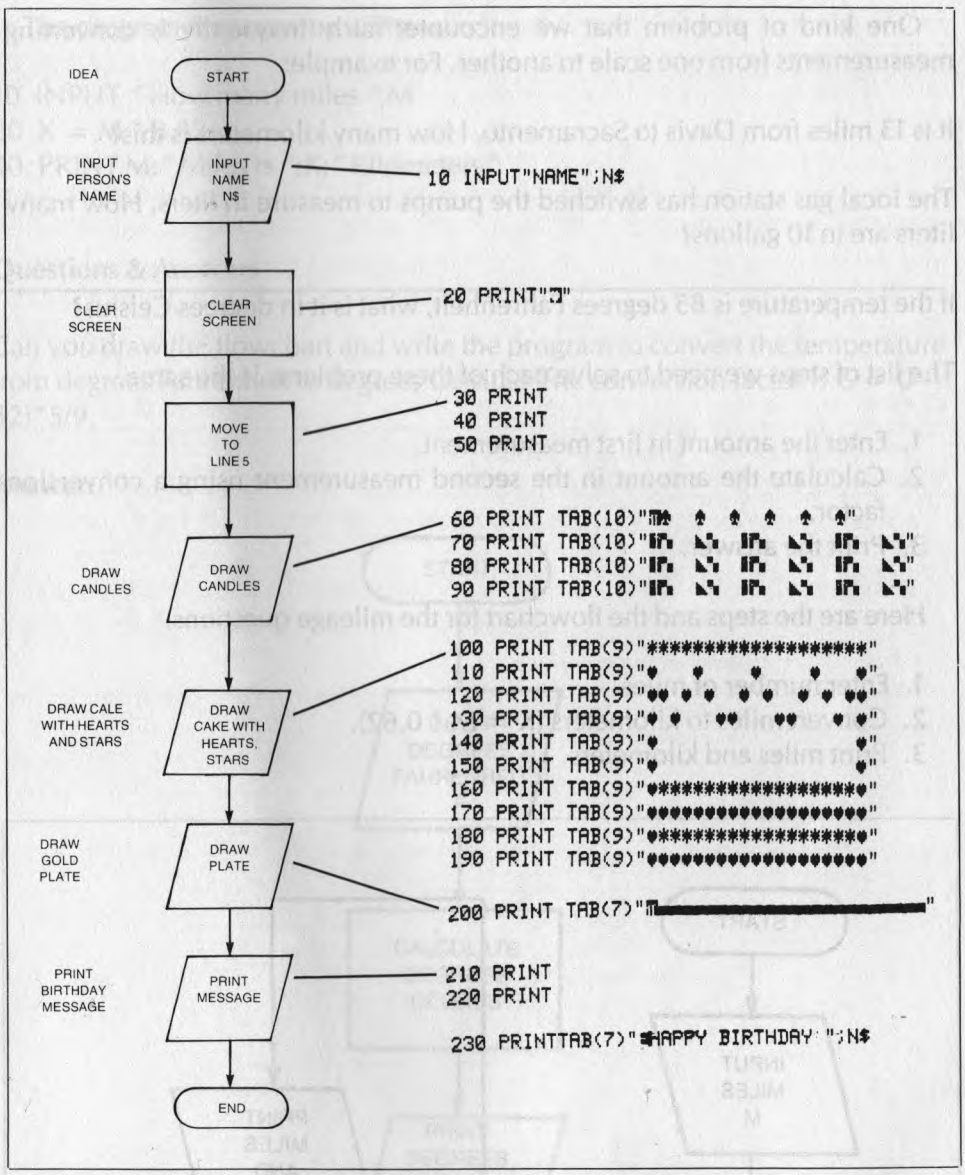


Figure 3.2 Algorithm, flowchart and program for the "Birthday Cake" Program.

It is not mandatory that you write an outline or make a list before you draw a flowchart, but this preparation will help reduce the time spent debugging the program and also improve the chances that the program will meet your expectations.

One kind of problem that we encounter fairly frequently is converting measurements from one scale to another. For example:

It is 13 miles from Davis to Sacramento. How many kilometers is this?

The local gas station has switched the pumps to measure in liters. How many liters are in 10 gallons?

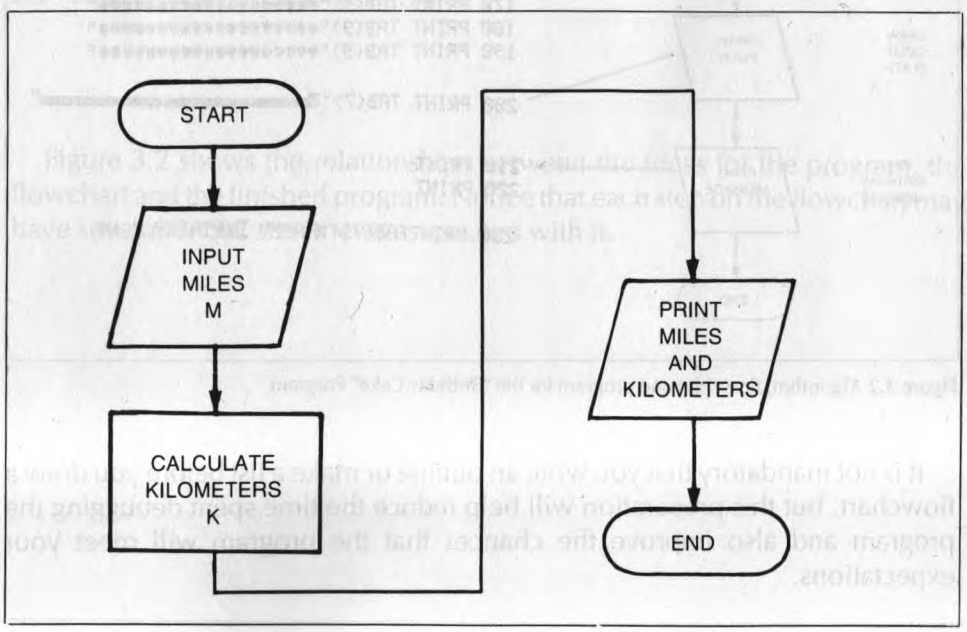
If the temperature is 85 degrees Fahrenheit, what is it in degrees Celsius?

The list of steps we need to solve each of these problems is the same.

1. Enter the amount in first measurement.
2. Calculate the amount in the second measurement using a conversion factor.
3. Print the answer.

Here are the steps and the flowchart for the mileage questions.

1. Enter number of miles.
2. Convert miles to kilometers (K = M * 0.62).
3. Print miles and kilometers.



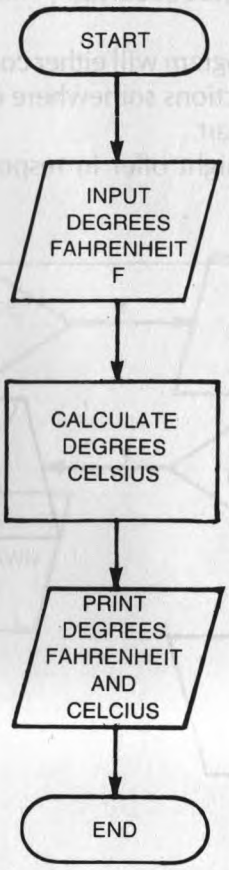
The program may now be written:

```
10 INPUT " How many miles ";M
20 K = M * 0.62
30 PRINT M:" Miles is ";K;" Kilometers"
```

Questions & Answers

Can you draw the flowchart and write the program to convert the temperature from degrees Fahrenheit to degrees Celsius? The conversion factor is C = (F–32)*5/9.

Answer:



```
10 INPUT " DEGREES FAHRENHEIT ";F
20 C = (F-32) * 5/9
30 PRINT F;" DEGREES FAHRENHEIT EQUALS "; C ; " DEGREES CELSIUS"
```

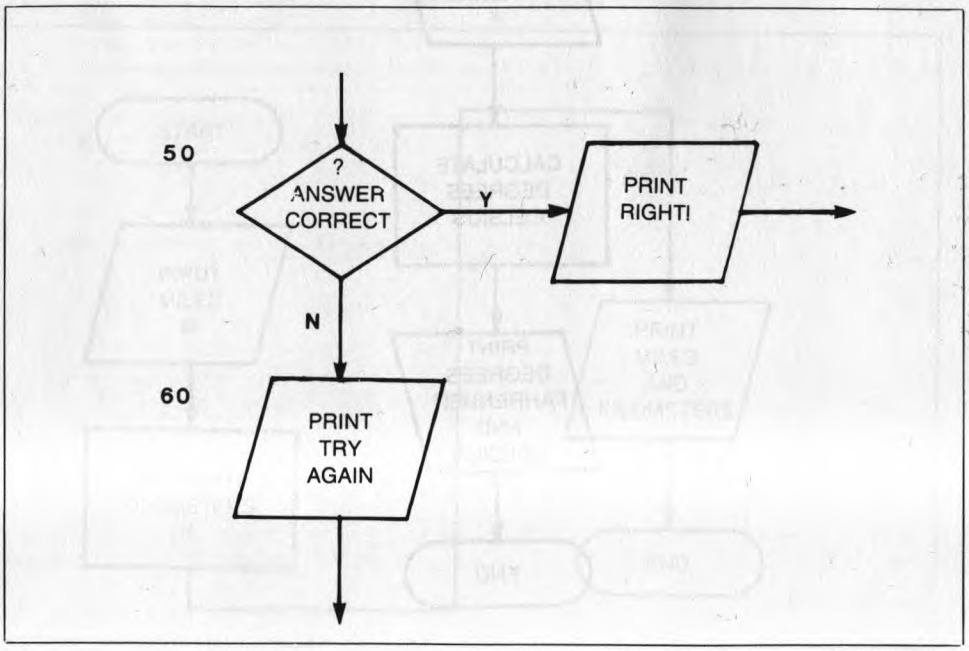
The flowchart for the birthday cake was a straight line chart. That is, the steps followed one after another in sequence. However, solutions to problems often involve asking questions and doing different things depending on the answer.

Asking the right questions helps us make decisions about what to do next. All but the simplest of programs require us to ask questions and make decisions. Here are some questions that have "yes/no" answers that might appear in a program.

- Is the answer to the math problem correct?
- Do you want to play again?
- Is the end-of-month balance positive?

Based on the answer, the program will either continue to the next instruction or skip to another set of instructions somewhere else in the program. But let's see how this looks on a flowchart.

Here are the options we might offer in response to the question. "Is the answer correct?"



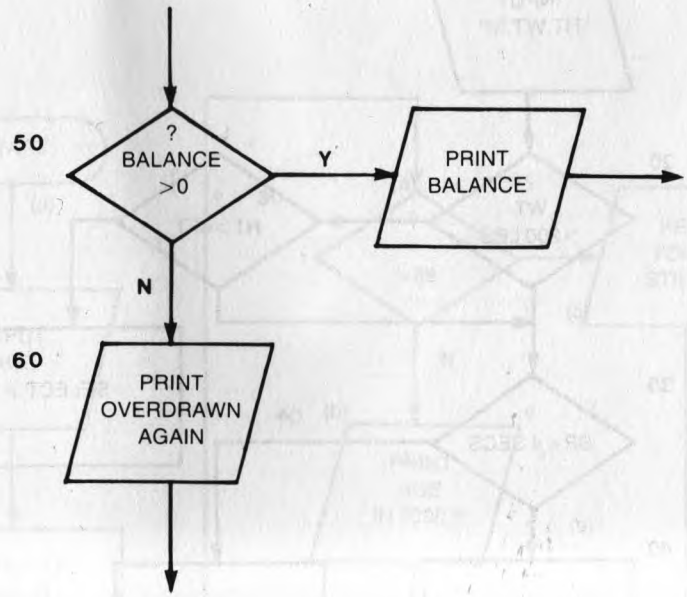
The decision symbol is used for any logical decision, and it may include one of the following relational operators:

=	Equal
< >	Not equal
>	Greater than
<	Less than
> =	Greater than or equal to
< =	Less than or equal to

Questions & Answers

Can you draw the section of the flowchart that will test the end of the month balance in your checkbook and print an appropriate message?

Answer:

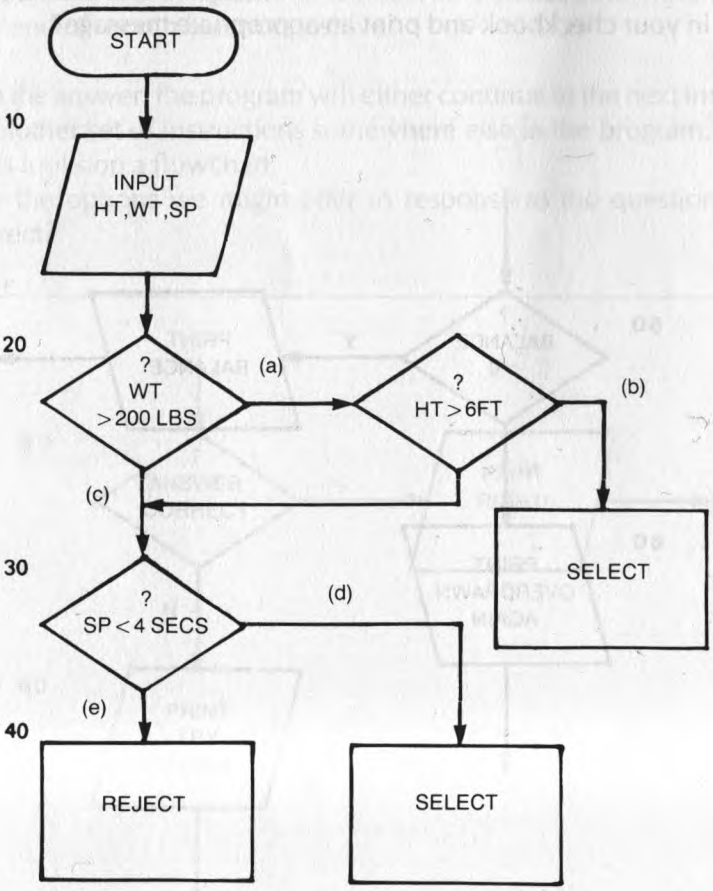


Here is a more complicated problem. The high school football coach wants to write a program to select players for the team tryouts. He has the players' weights, heights and times for the 40-yard dash. To make the team, they must meet at least one of the following criteria:

Weigh more than 200 pounds and be taller than 6 feet.
Run the 100-yard dash in less than 10 seconds.

Questions & Answers

Label the pathways in this flowchart to show whether they are for "yes" or "no" answers at the decision points.



Answer: (a) Yes (b) Yes (c) No (d) Yes (e) No

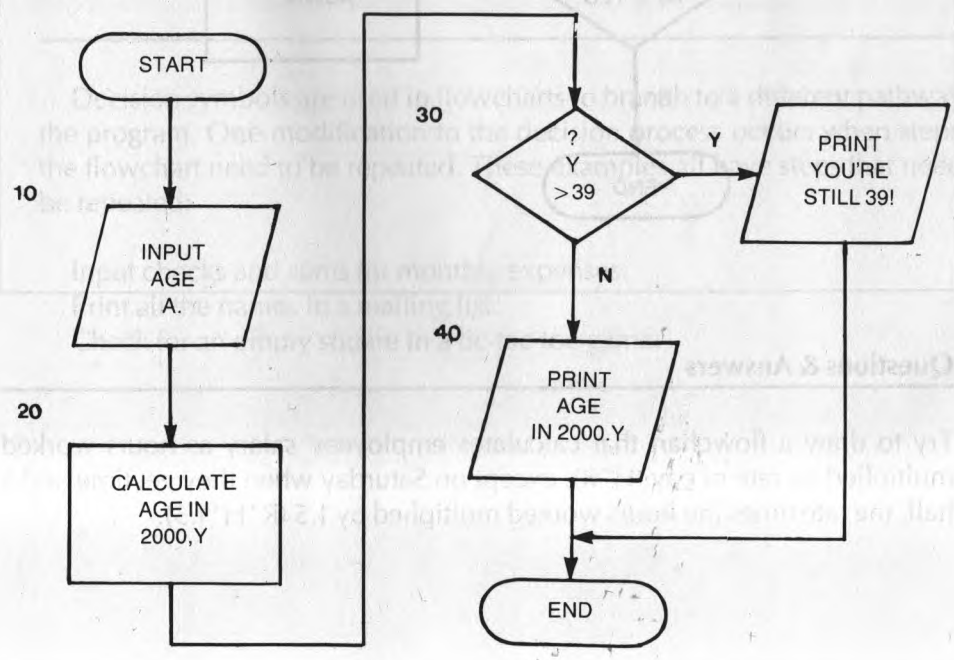
A program was written in Chapter 2 to calculate a person's age in the year 2000 A.D. A diplomatic touch can be added to the program by testing to see if the person will be over 39 in the year 2000. If this is true the program will print "You will still be 39!" Otherwise, it prints the real age. These are the steps for the program:

1. Input the person's age now.
2. Calculate his/her age in the year 2000.
3. If his/her age will be greater than 39, print, "You're still 39." If not, print his/her true age.

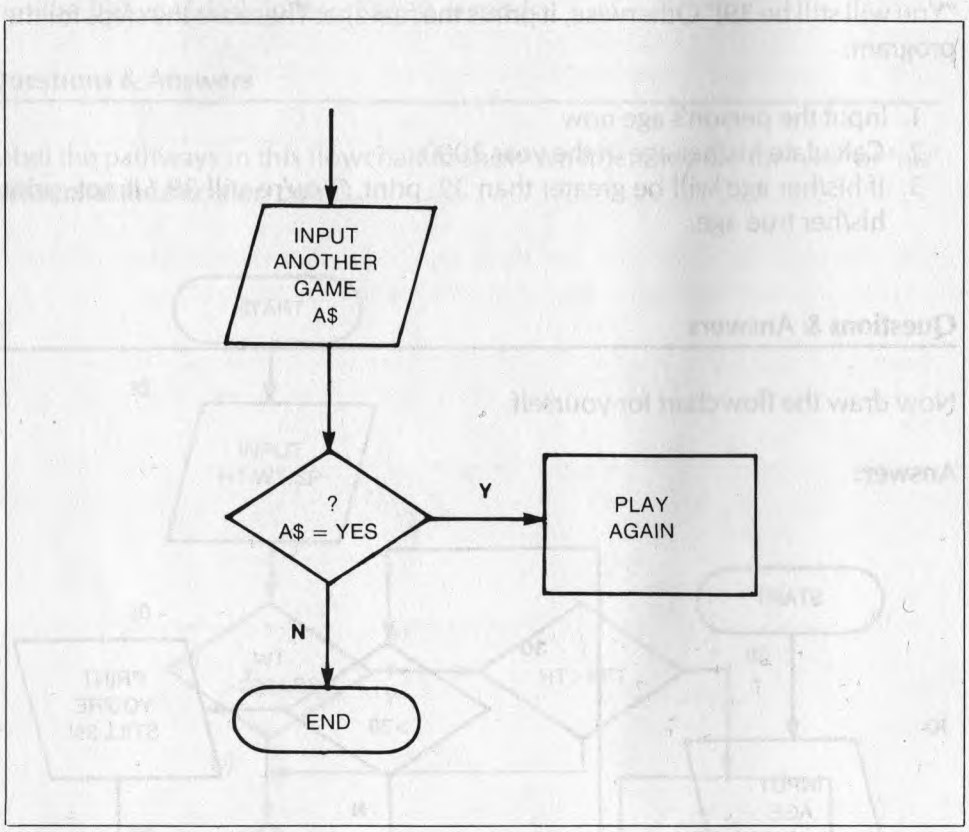
Questions & Answers

Now draw the flowchart for yourself.

Answer:



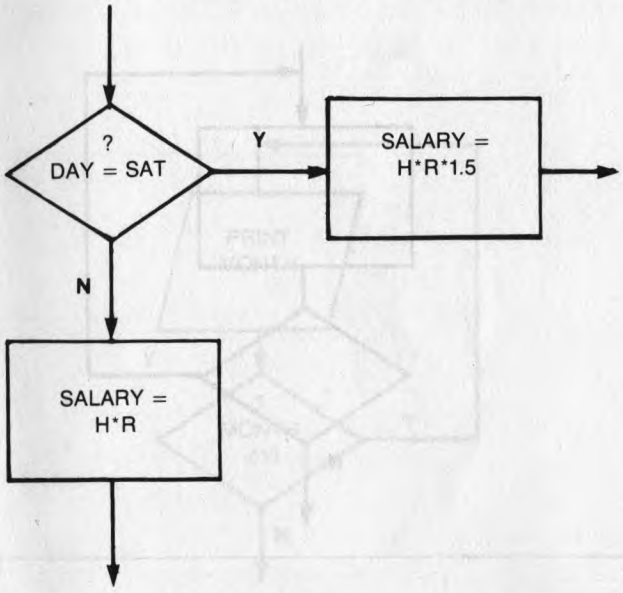
So far the decisions in the flowcharts we have written have tested numeric information, but you are not limited to this. Decisions can test alphanumeric information. For example, in a game program we might test to see whether the programmer wanted to play again or not. The flowchart would look something like this:



Questions & Answers

Try to draw a flowchart that calculates employees' salary as hours worked multiplied by rate of pay (H*R), except on Saturday when they get time and a half, the rate times the hours worked multiplied by 1.5 (R*H*1.5).

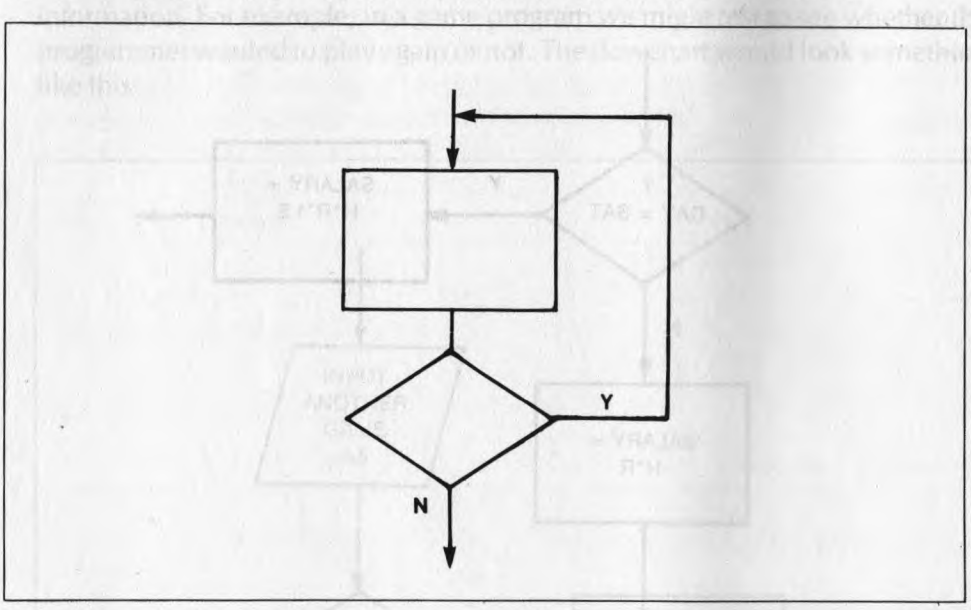
Answer:



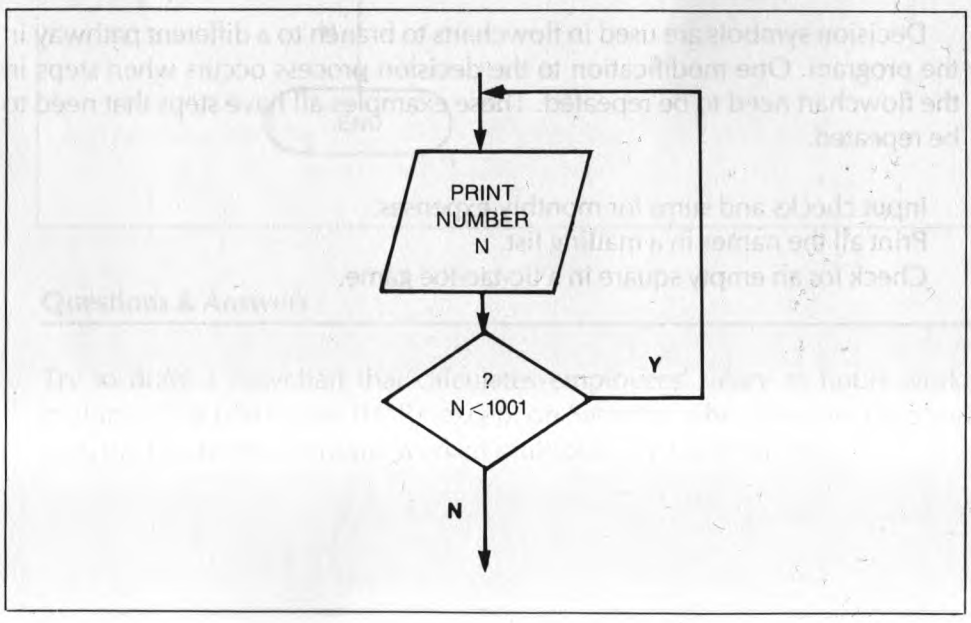
Decision symbols are used in flowcharts to branch to a different pathway in the program. One modification to the decision process occurs when steps in the flowchart need to be repeated. These examples all have steps that need to be repeated:

Input checks and sums for monthly expenses.
Print all the names in a mailing list.
Check for an empty square in a tic-tac-toe game.

Steps that are repeated are called loops. There are two kinds of loops. A DO-UNTIL loop that will always repeat the steps at least once looks like this:



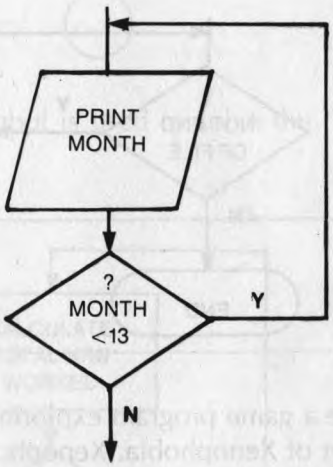
A DO-UNTIL loop to print the numbers from one to one thousand would look like this:



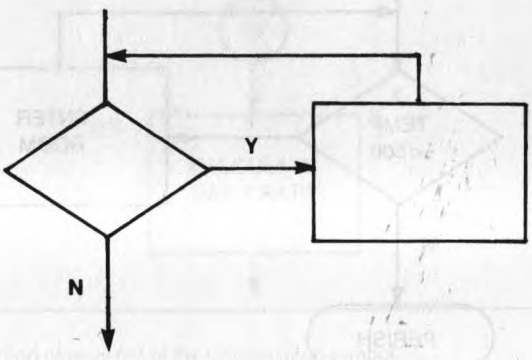
Questions & Answers

Draw a flowchart using a DO-UNTIL loop to print the names of the twelve months.

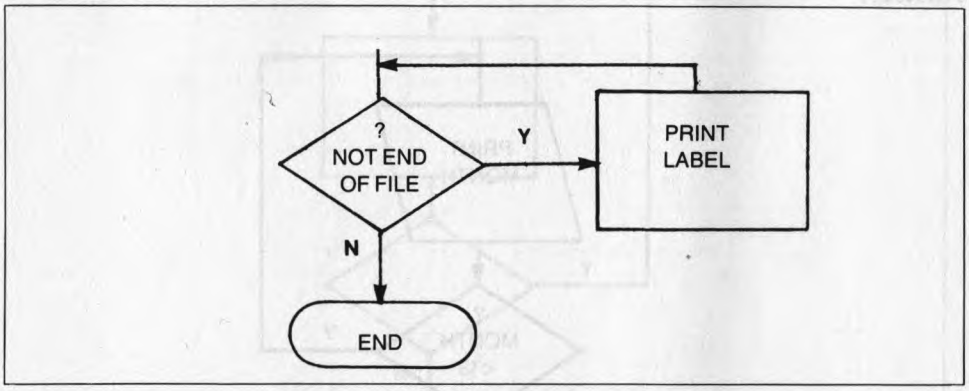
Answer:



The second kind of loop is a DO-WHILE loop that does not necessarily repeat; a question comes first and if the answer is "no," the loop is never entered.



A DO-WHILE loop is used frequently when data is being read from a data file on a diskette or cassette. Suppose we want to print a mailing list, but we are not sure how many addresses are in the list, or even if we have any at all. The DO-WHILE loop will check to see if there is anything in the file first before it prints a label.

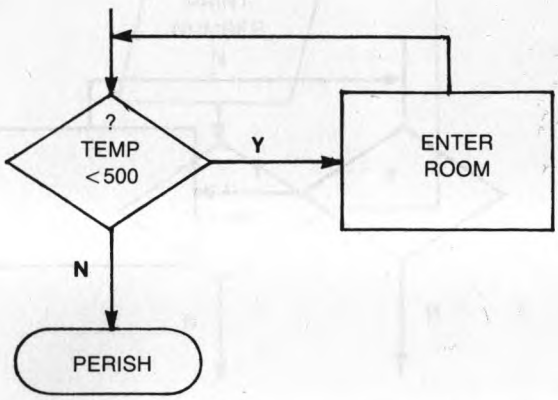


Suppose we have a game program exploring the rooms of an abandoned palace on the planet of Xenophobia. Xenophobia is very hot and our space suits can only cope with temperatures below 500 degrees. If we enter a room and the temperature is above 500 degrees, we perish.

Questions & Answers

Can you write a DO-WHILE loop for this situation?

Answer:



Continuation Symbol

Flowcharts are easy to follow for short programs with only one or two decisions, but they rapidly become complicated. Frequently they require more than one sheet of paper and the branching pathways cross over one another. To solve these problems we use the continuation symbol:

Here, the continuation symbol is used to solve the "out of room on the paper" problem.

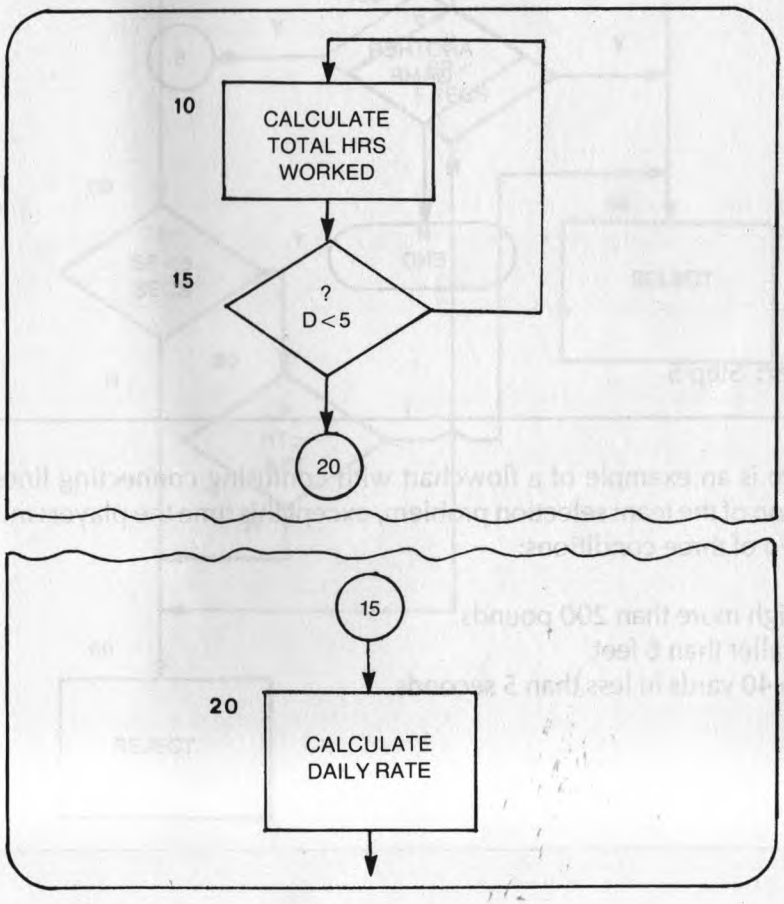
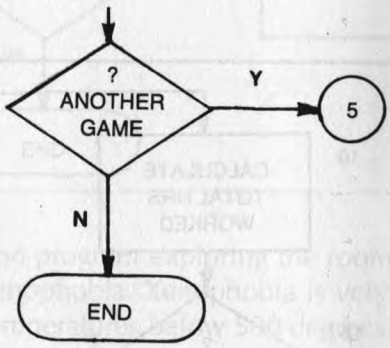


Figure 3.3 End-of-page use of the Continuation symbol.

Step 15 was an input operation at the very bottom of the page, so a continuation symbol was drawn with the next step number written inside. At the top of the new page the continuation symbol is the first symbol drawn. This time and the preceding step number, 15, is written inside it.

Questions & Answers

Which step will be executed in the following example if the player wants to play another game?

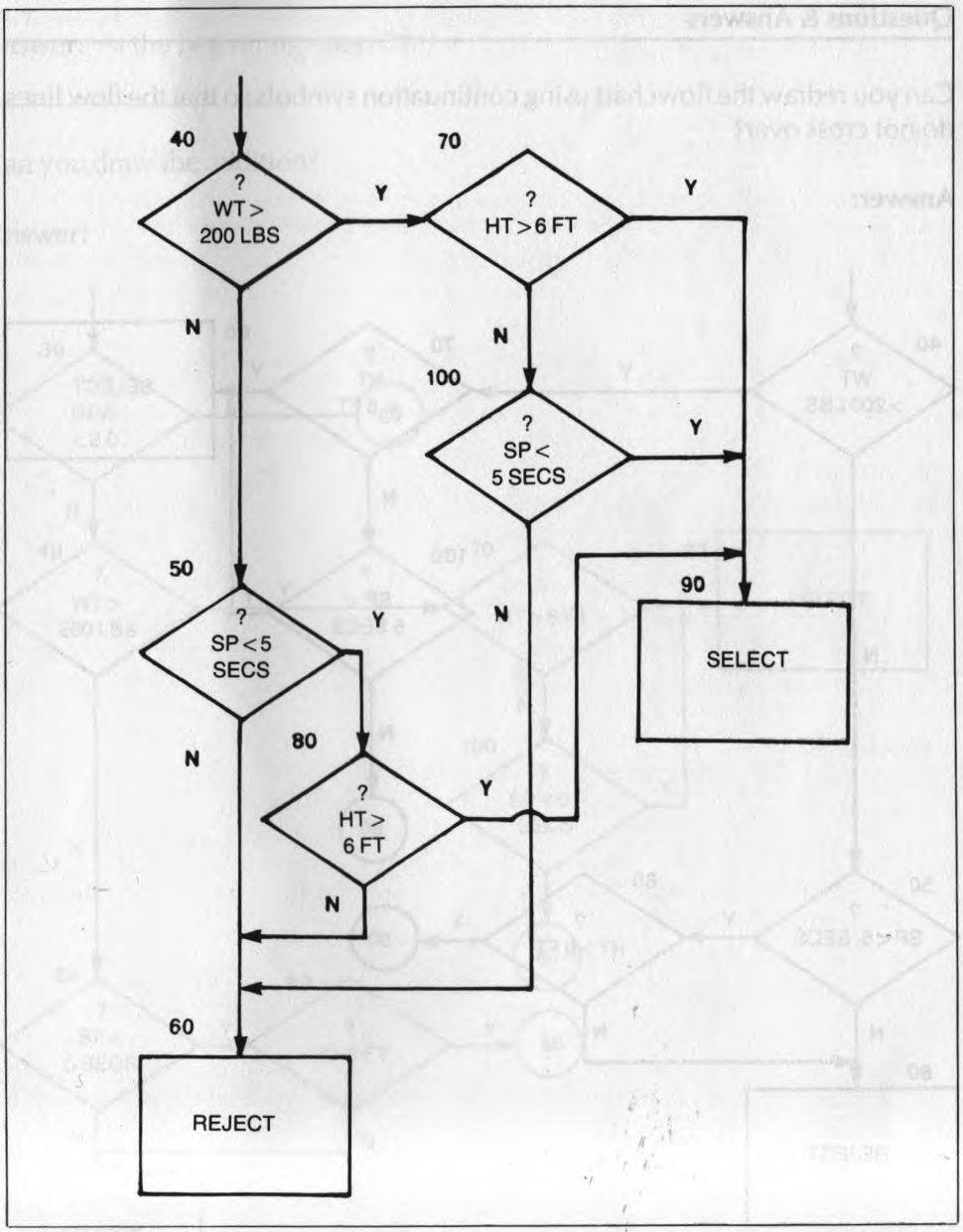


Answer: Step 5

Here is an example of a flowchart with confusing connecting lines. It is a variation of the team selection problem, except this time the players must meet any two of three conditions:

Weigh more than 200 pounds.
Be taller than 6 feet.
Run 40 yards in less than 5 seconds.

The part of the flowchart which selects the players looks like this:

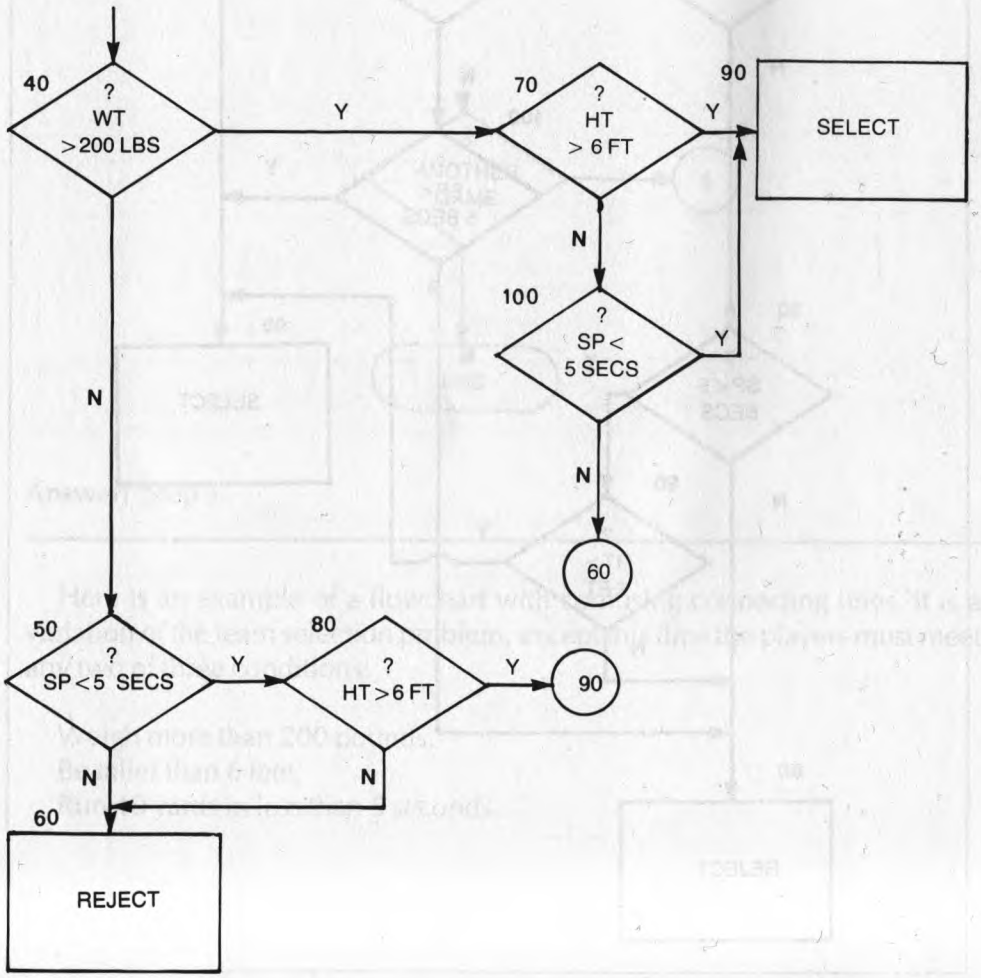


Notice that the SELECT process is step 90 and the REJECT process is step 60.

Questions & Answers

Can you redraw the flowchart using continuation symbols so that the flow lines do not cross over?

Answer:

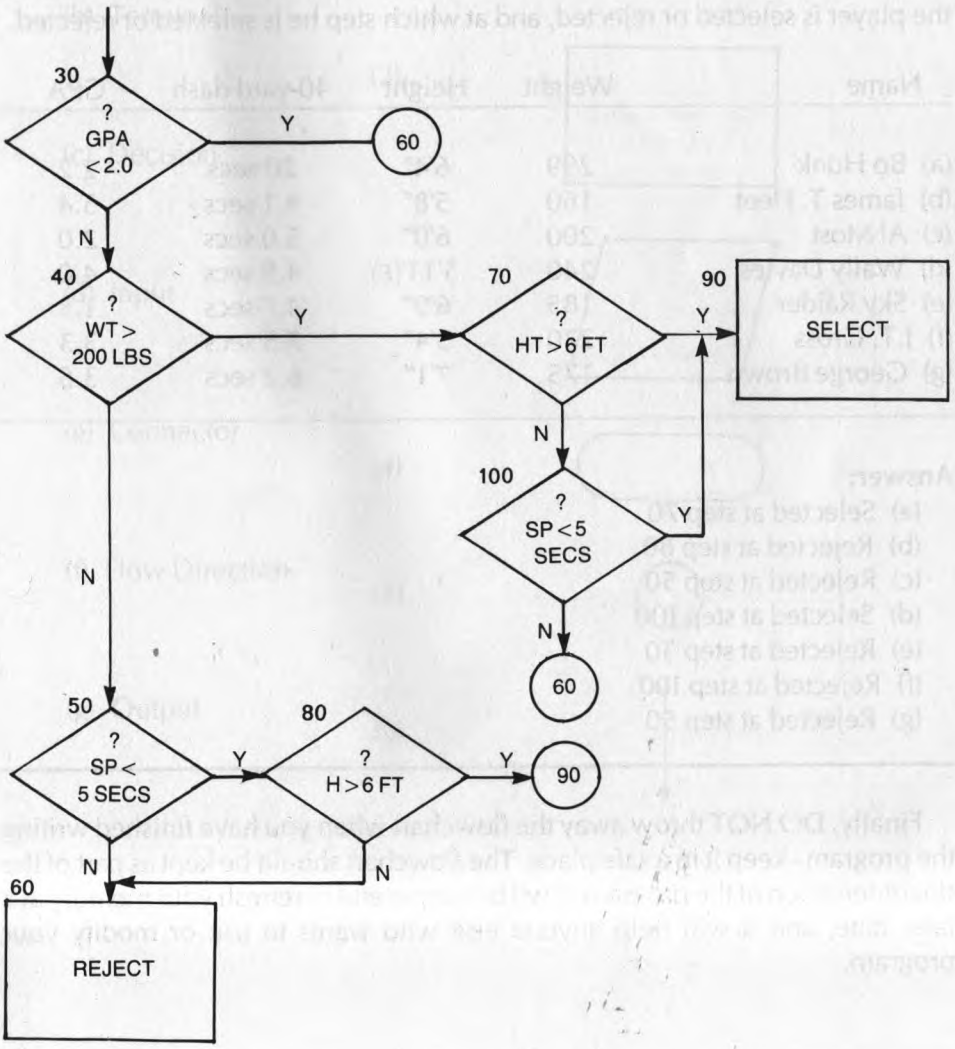


Now, the dean has said that players must have at least a C, 2.00, grade average to play on the team. Where would you add this decision point to the flowchart?

Answer: At the beginning - step 30.

Can you draw the addition?

Answer:



When we have finished the flow chart, we should take some test data and use this to test the flowchart, making the changes indicated at each step. If the end result is the one you predicted for the program, then the flowchart is correct and it can be translated into BASIC code. If the end result is not as predicted, then the flowchart has a flaw in its logic.

Questions & Answers

Here is a list of the statistics for some of the students who tried out for the football team. Use the flowchart we developed in frame 10 to decide whether the player is selected or rejected, and at which step he is selected or rejected.

Name	Weight	Height	40-yard dash	GPA
(a) Bo Hunk	299	6'4"	20 secs	2.2
(b) James T. Fleet	160	5'8"	4.1 secs	3.4
(c) Al Most	200	6'0"	5.0 secs	2.0
(d) Wally Davies	240	5'11"	4.9 secs	4.0
(e) Sky Raider	185	6'9"	4.7 secs	1.5
(f) J.T. Gross	270	5'4"	7.5 secs	3.3
(g) George Brown	175	7'1"	6.2 secs	3.6

Answer:

(a) Selected at step 70
(b) Rejected at step 80
(c) Rejected at step 50
(d) Selected at step 100
(e) Rejected at step 30
(f) Rejected at step 100
(g) Rejected at step 50

Finally, DO NOT throw away the flowchart when you have finished writing the program - keep it in a safe place. The flowchart should be kept as part of the documentation of the program. It wil be very useful to refresh your memory at a later date, and it will help anyone else who wants to use or modify your program.

SELF-CHECK

Work this check carefully. I am sure you are more than ready to get back to the keyboard, but this preparation will pay off in the long run - I promise!

1. Match the following:

(a) Process

(b) Terminal

(c) Decision

(d) Input

(e) Connector

(f) Flow Direction

(g) Output

(1)

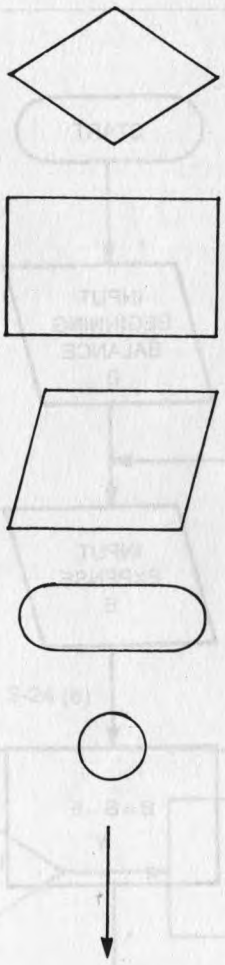
(2)

(3)

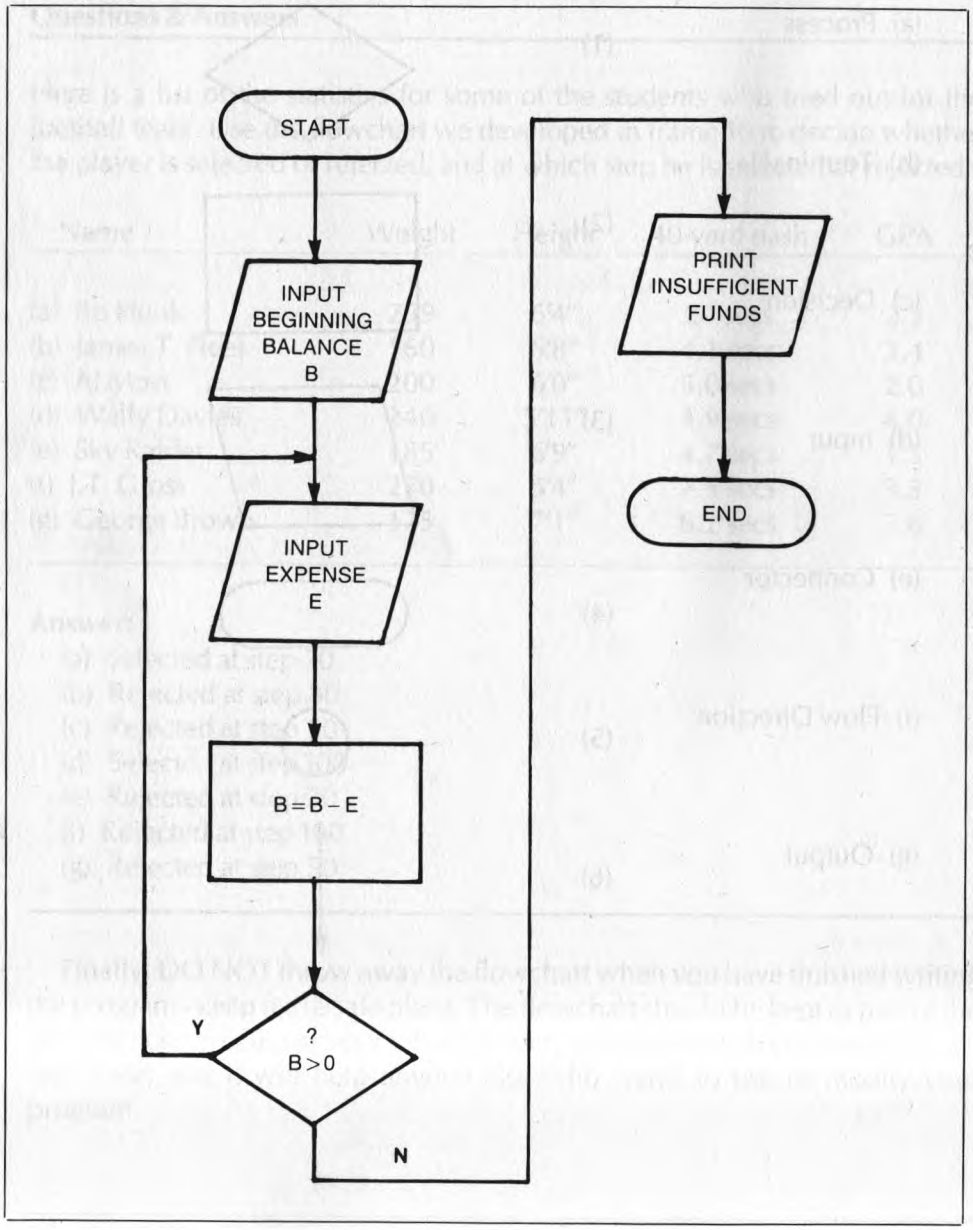
(4)

(5)

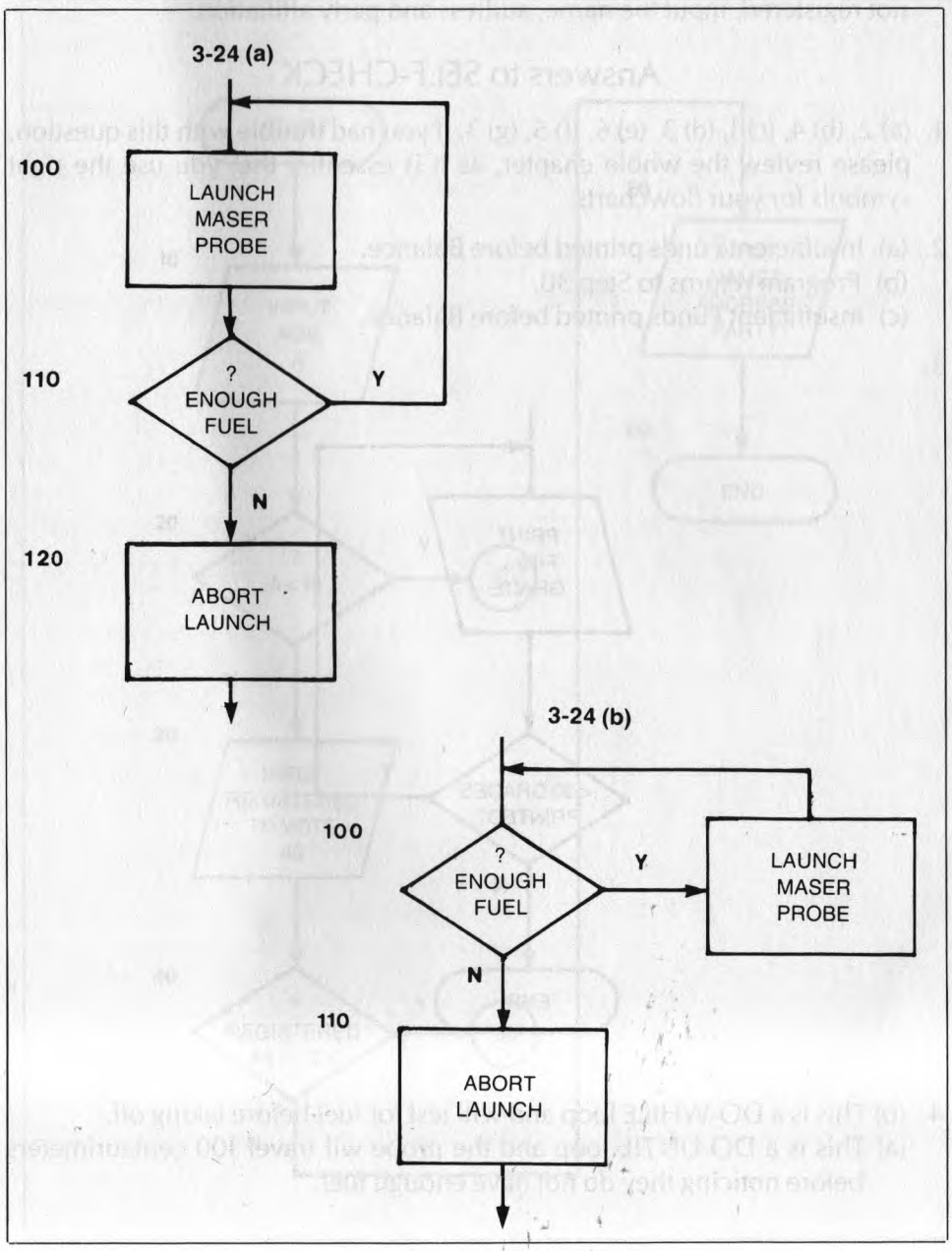
(6)



2. In this flowchart, which statement will be executed when?
   (a) B = 0
   (b) B = 0.99
   (c) B = −0.99

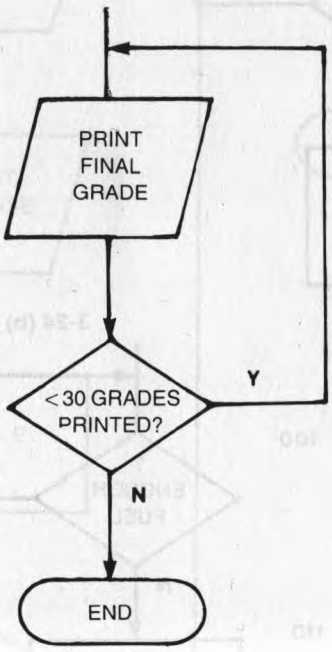


3. Draw a DO-UNTIL loop to print the final grades for a class of thirty students.
4. Which of the following flowcharts will prevent the Intergalactic forces from launching their laser-probe to the planet of Id without sufficient fuel?



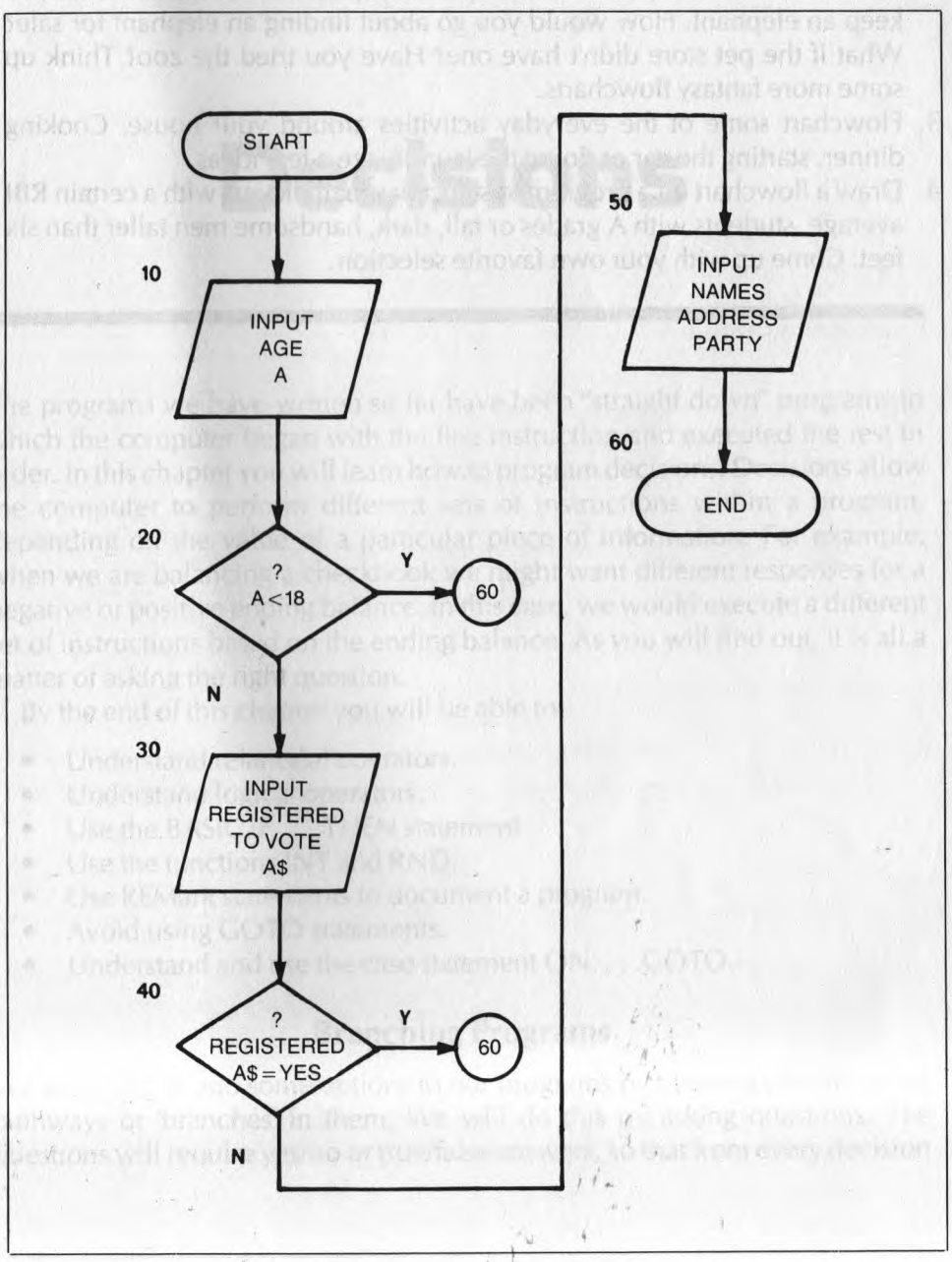
5. There are always _____ pathway(s) entering a decision symbol and _____ leaving.
6. Draw a flowchart to input a person's age and to check if he/she is old enough to vote. If so, find out if he/she has registered to vote. If the person is not registered, input the name, address and party affiliation.

Answers to SELF-CHECK

1. (a) 2, (b) 4, (c) 1, (d) 3, (e) 6, (f) 5, (g) 3. If you had trouble with this question, please review the whole chapter, as it is essential that you use the right symbols for your flowcharts.
2. (a) Insufficient Funds printed before Balance.
   (b) Program returns to Step 30.
   (c) Insufficient Funds printed before Balance.
3. 

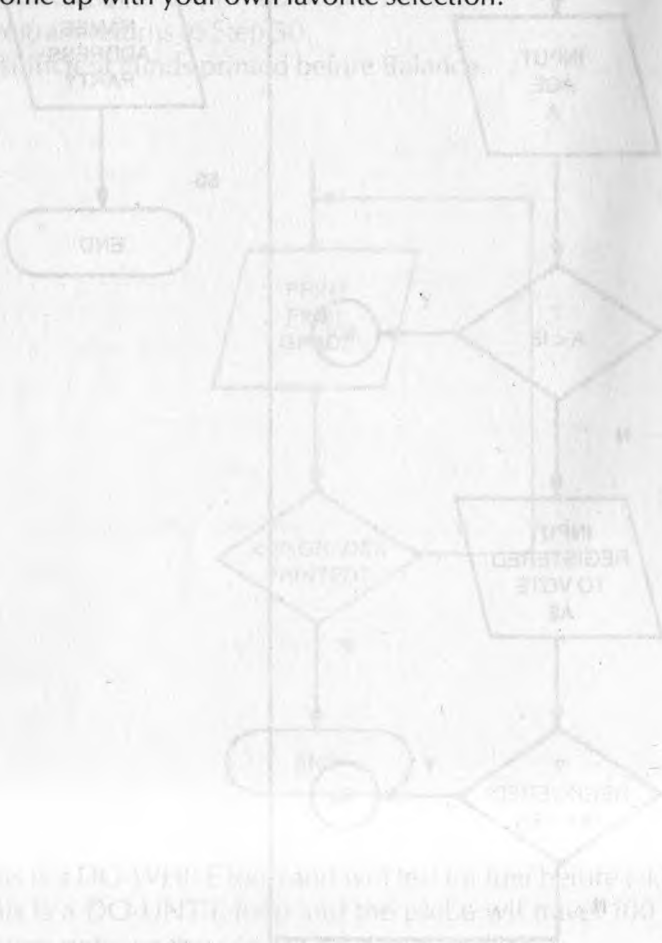
4. (b) This is a DO-WHILE loop and will test for fuel before taking off.
   (a) This is a DO-UNTIL loop and the probe wil travel 100 centaurimeters before noticing they do not have enough fuel.

5. One, two.
6. Here is my flowchart. Yours may not have the same statements, but the order and the flow should be the same.



Suggestions for More Flowchart Practice

1. Draw a flowchart for every program that you have written up until now - and start a program documentation file to be kept in a safe place!
2. Draw a flowchart to persuade your parents, spouse or roommate to let you keep an elephant. How would you go about finding an elephant for sale? What if the pet store didn't have one? Have you tried the zoo? Think up some more fantasy flowcharts.
3. Flowchart some of the everyday activities around your house. Cooking dinner, starting the car or doing the laundry are a few ideas.
4. Draw a flowchart for a program to select baseball players with a certain RBI average, students with A grades or tall, dark, handsome men taller than six feet. Come up with your own favorite selection.



CHAPTER 4

Decisions

The programs we have written so far have been "straight down" programs in which the computer began with the first instruction and executed the rest in order. In this chapter you will learn how to program decisions. Decisions allow the computer to perform different sets of instructions within a program, depending on the value of a particular piece of information. For example, when we are balancing a checkbook we might want different responses for a negative or positive ending balance. In this case, we would execute a different set of instructions based on the ending balance. As you will find out, it is all a matter of asking the right question.

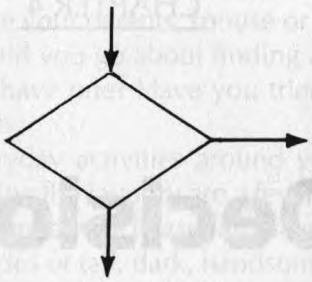
By the end of this chapter you will be able to:

- Understand relational operators.
- Understand logical operators.
- Use the BASIC IF. . . .THEN statement.
- Use the functions INT and RND.
- Use REMark statements to document a program.
- Avoid using GOTO statements.
- Understand and use the case statement ON. . . .GOTO.

Branching Programs

We are going to add some options to our programs by opening up alternative pathways or 'branches' in them. We will do this by asking questions. The questions will require yes/no or true/false answers, so that from every decision

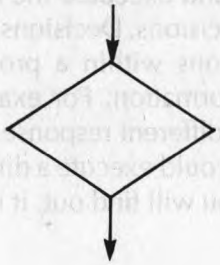
there will be two resultant pathways. If you remember from Chapter 3, the flowchart symbol for a decision looks like this:



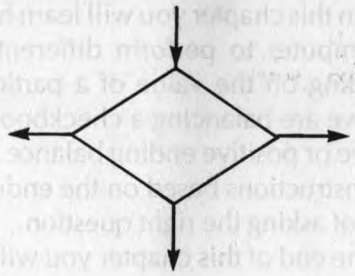
Questions & Answers

What is the problem with these flowchart symbols?

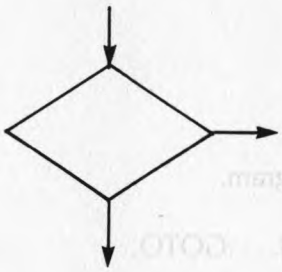
(a)



(b)



(c)



Answer:

(a) There is only one pathway leaving the symbol.
(b) There are three pathways leaving the symbol.
(c) Nothing - it is perfect, with two pathways leaving the symbol.

The questions tested in decision statements include relational and logical operators. The symbols for the relational operators are the same ones used in flowcharts.

= equal
< > not equal
< less than
> greater than
< = less than or equal
> = greater than or equal

They can be used in the following way in a BASIC statement:

<expression> <operator> <expression>

Here are some examples using numeric information:

C = A + B	Is C equal to A plus B?
HT > 6	Is height greater than 6 feet?
WT < 200	Is weight less than 200 pounds?
B > = 0	Is the balance greater than or equal to zero?

The relational operators, equal, not equal, greater than and less than, may also be used to compare string variables.

A$ = "YES"	Is the answer "yes"?
S$ = "F"	Is this student female?
FF$ < > "PIZZA"	The favorite food isn't pizza?

Questions & Answers

Which of these statements will check to see whether my cup of coffee has cooled to at least seventy degrees?

(a) T = 70
(b) T > 69
(c) T < 70
(d) T > = 69
(e) T < = 70

Answer: (e) This statement has the less than and equal to operators, so it will be true when the temperature is less than or equal to 70.

Logical operators are a little more complicated. The Commodore 64 uses these three logical operators:

AND
OR
NOT

Logical operators result in a true/false answer. This is a rather simplistic explanation for what actually happens. We will deal with logical operators in more detail in Chapter 7, but for now this explanation will allow us to practice using them. Let's look at them one at a time.

AND is used like this:

<expression>AND<expression>

For example:

A>B AND B>C

This statement will be true if A is greater than B AND B is greater than C. Remember that AND will be true only if both expressions are true.

OR is used this way:

<expression>OR<expression>

For example:

A>B OR B>C

This statement will be true when:

A is greater than B,
OR B is greater than C,
OR A is greater than B and B is greater than C.

An OR statement will be true if both expressions are true or if either one is true.

NOT is used like this:

NOT <expression>

For example:

NOT A>B

This statement will be true when A is not greater than B. This is perhaps the hardest one to understand, but it is very useful. Remember that NOT compliments the expression, which means that NOT will be true when the expression is false.

Questions & Answers

If A, the cost of bubble gum, is 27 cents, B, the cost of a candy bar, is 30 cents, and C, the cost of a soda, is 40 cents, which of the following expressions are true?

_____ (a) A < C and C > B
_____ (b) B > C OR A < C
_____ (c) NOT B > C

Answer:

All of them.

(a) True, A (27) is less than C (40), and C (40) is greater than B (30).
(b) True, B (30) is not greater than C (40), but A (27) is less than C (40). (Only one has to be true to make this statement true).
(c) True, B (30) is not greater than C (40).

Before we move on, let's try one more example. The captain of the Titanic has the passenger list coded by sex (S$ = M or F) and age (A). Write an expression that will allow him to select all females and all children under the age of twelve?

Answer: S$ = "F" OR A < = 12

IF Statements

Now let's put operators in a program. To do this we need the BASIC IF statement:

IF <expression> <operator> <expression> THEN <expression>

For example:

```
330 IF A < = 12 THEN T = T * 0.5
340 PRINT T
```

The test, or question, in statement 330 is whether the person's age, (A), is less than or equal to 12. If it is, then that person is a child and entitled to a half-fare ticket (T). Statement 340 prints the cost of the ticket.

Look at this example:

```
40 IF B < =0 THEN PRINT "INSUFFICIENT FUNDS"
50 PRINT B
```

Questions & Answers

What will be printed when the balance (B) is each of these amounts?

(a) 0
(b) 1525.98
(c) −23.69

Answer:

(a) INSUFFICIENT FUNDS
    0
(b) 1525.98
(c) INSUFFICIENT FUNDS
    −23.69

Frequently the expression following THEN is another statement number. This makes the program branch to that statement number. For example, after playing tic-tac-toe, the program should ask if the player wants another game. The statements might look like this:

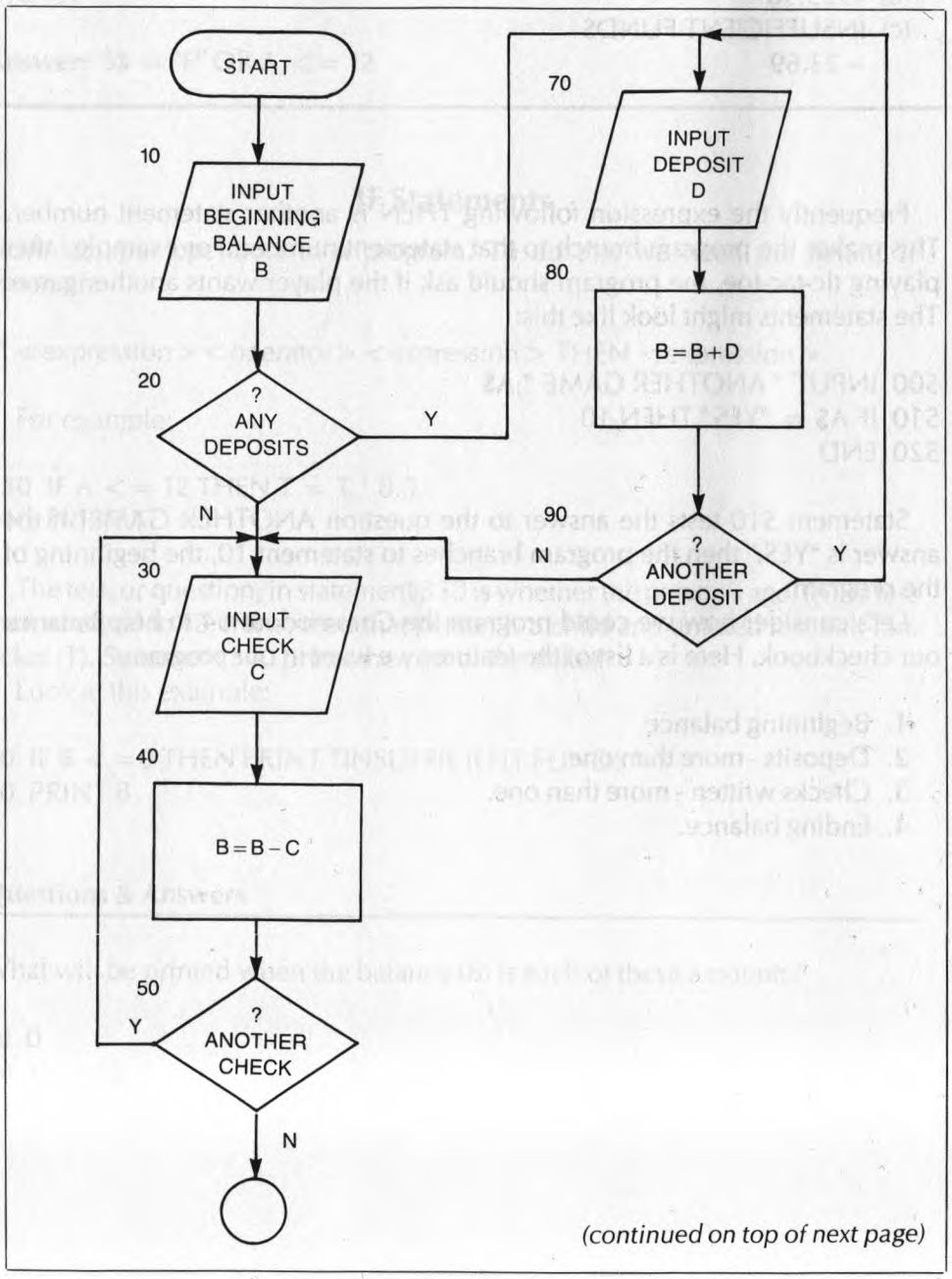
```
500 INPUT " ANOTHER GAME ";A$
510 IF A$ = "YES" THEN 10
520 END
```

Statement 510 tests the answer to the question ANOTHER GAME? If the answer is "YES," then the program branches to statement 10, the beginning of the program.

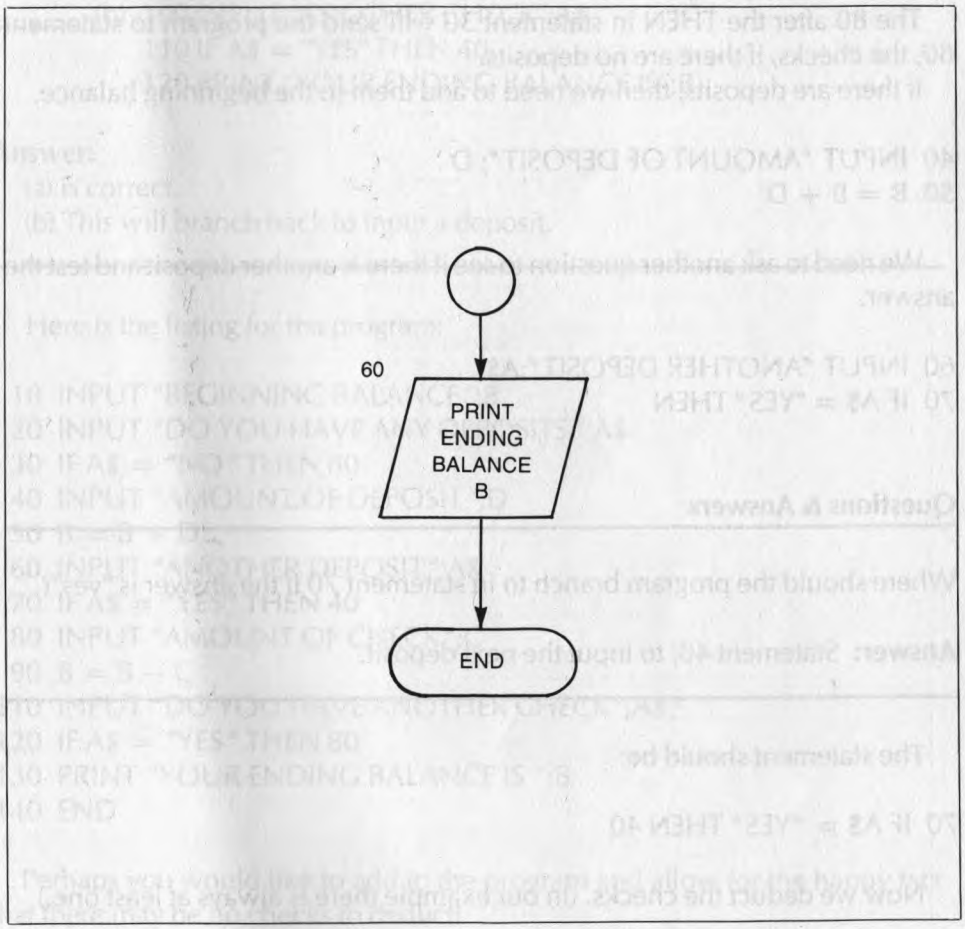
Let's consider how we could program the Commodore 64 to help balance our checkbook. Here is a list of the features we want in our program.

1. Beginning balance.
2. Deposits - more than one.
3. Checks written - more than one.
4. Ending balance.

The second and third items on our list require us to see if there is another entry. The flowchart will look like this:



(continued on top of next page)



We will program this step by step. The first step is entering the beginning balance.

```
10 INPUT "BEGINNING BALANCE ";B
```

Next we need to find out if there are any deposits. First we ask the question and then we test the answer.

```
20 INPUT "DO YOU HAVE ANY DEPOSITS ";A$
30 IF A$ = "NO" THEN 80
```

The 80 after the THEN in statement 30 will send the program to statement 80, the checks, if there are no deposits.

If there *are* deposits, then we need to add them to the beginning balance.

```
40 INPUT "AMOUNT OF DEPOSIT"; D
50 B = B + D
```

We need to ask another question to see if there is another deposit and test the answer.

```
60 INPUT "ANOTHER DEPOSIT";A$
70 IF A$ = "YES" THEN
```

Questions & Answers

Where should the program branch to in statement 70 if the answer is "yes"?

Answer: Statement 40, to input the next deposit.

The statement should be:

```
70 IF A$ = "YES" THEN 40
```

Now we deduct the checks. (In our example there is always at least one.)

```
80 INPUT "AMOUNT OF CHECK";C
90 B = B – C
```

Questions & Answers

Which of these statements will find out if there is another check, and branch to the correct place in the program?

_____ (a)
```
100 INPUT "ANOTHER CHECK";A$
110 IF A $ = "YES" THEN 80
120 PRINT "YOUR ENDING BALANCE IS";B
```

_____ (b)
```
100 INPUT "ANOTHER CHECK";A$
110 IF A$ = "YES" THEN 40
120 PRINT "YOUR ENDING BALANCE IS";B
```

Answer:

(a) is correct.

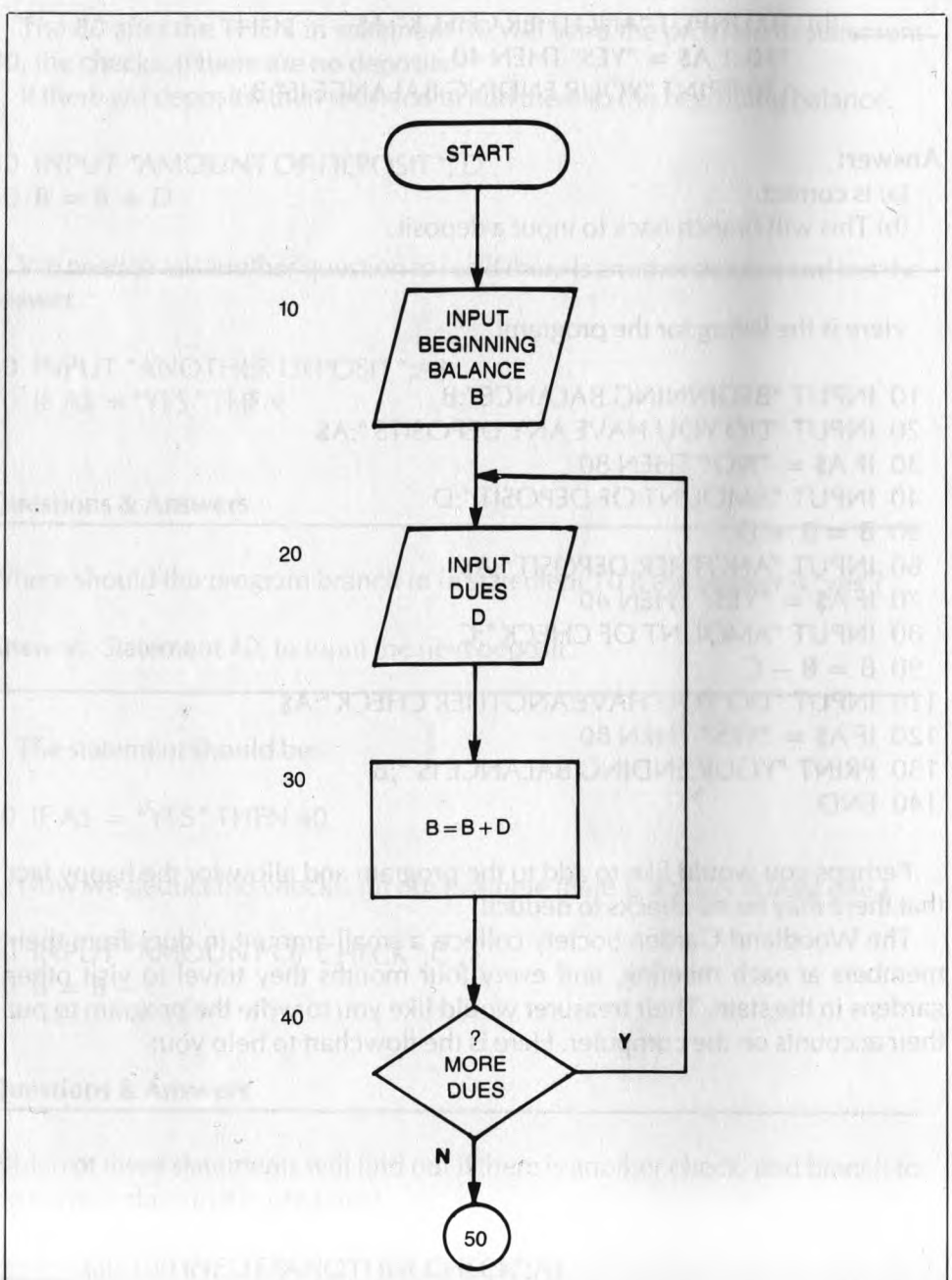
(b) This will branch back to input a deposit.

Here is the listing for the program:

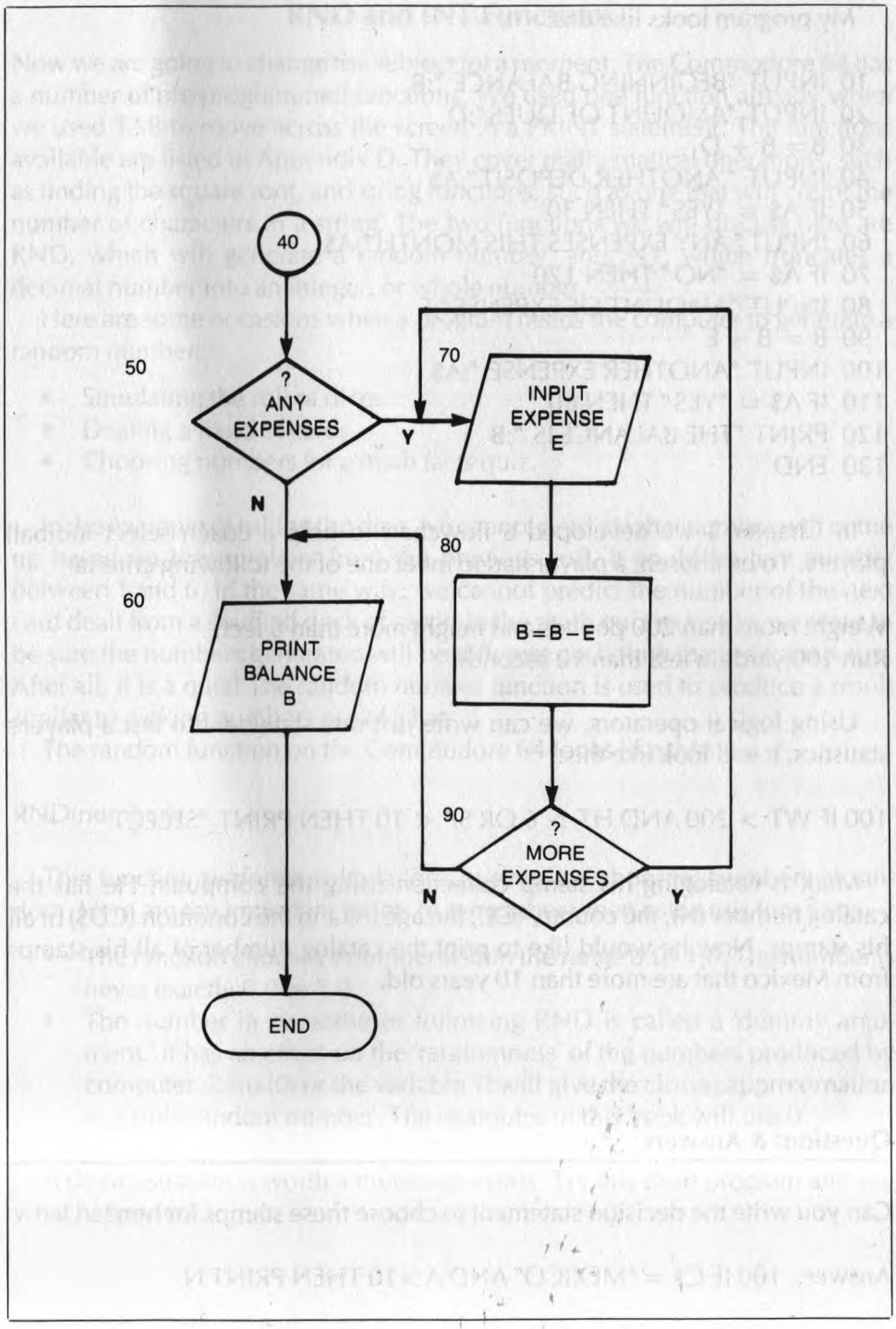
```
 10 INPUT "BEGINNING BALANCE";B
 20 INPUT "DO YOU HAVE ANY DEPOSITS ";A$
 30 IF A$ = "NO" THEN 80
 40 INPUT "AMOUNT OF DEPOSIT";D
 50 B = B + D
 60 INPUT "ANOTHER DEPOSIT";A$
 70 IF A$ = "YES" THEN 40
 80 INPUT "AMOUNT OF CHECK";C
 90 B = B - C
110 INPUT "DO YOU HAVE ANOTHER CHECK";A$
120 IF A$ = "YES" THEN 80
130 PRINT "YOUR ENDING BALANCE IS ";B
140 END
```

Perhaps you would like to add to the program and allow for the happy fact that there may be no checks to deduct!

The Woodland Garden Society collects a small amount in dues from their members at each meeting, and every four months they travel to visit other gardens in the state. Their treasurer would like you to write the program to put their accounts on the computer. Here is the flowchart to help you:


START
10
INPUT
BEGINNING
BALANCE
B
20
INPUT
DUES
D
30
B=B+D
40
?
MORE
DUES
Y
N
50

(continued)


40
50
?
ANY
EXPENSES
Y
70
INPUT
EXPENSE
E
N
80
60
PRINT
BALANCE
B
B=B-E
90
?
MORE
EXPENSES
N
Y
END

My program looks like this:

```
 10 INPUT "BEGINNING BALANCE ";B
 20 INPUT "AMOUNT OF DUES";D
 30 B= B + D
 40 INPUT " ANOTHER DEPOSIT";A$
 50 IF A$ = "YES" THEN 30
 60 INPUT" ANY EXPENSES THIS MONTH";A$
 70 IF A$ = "NO" THEN 120
 80 INPUT "AMOUNT OF EXPENSE";E
 90 B = B – E
100 INPUT "ANOTHER EXPENSE ";A$
110 IF A$ = "YES" THEN 80
120 PRINT "THE BALANCE IS ";B
130 END
```

In Chapter 3 we developed a flowchart to help a coach select football players. To be chosen, a player had to meet one of the following criteria:

Weight more than 200 pounds and height more than 6 feet.
Run 100 yards in less than 10 seconds.

Using logical operators, we can write just one statement to test a player's statistics. It will look like this:

```
100 IF WT > 200 AND HT > 6 OR SP < 10 THEN PRINT "SELECT"
```

Mark is cataloging his stamp collection using the computer. He has the catalog number (N), the country (C$), the age (A) and the condition (CD$) of all his stamps. Now he would like to print the catalog number of all his stamps from Mexico that are more than 10 years old.

Questions & Answers

Can you write the decision statement to choose these stamps for him?

Answer: 100 IF C$ = "MEXICO" AND A>10 THEN PRINT N

RND and INT Functions

Now we are going to change the subject for a moment. The Commodore 64 has a number of pre-programmed functions. We used one function already when we used TAB to move across the screen in a PRINT statement. The functions available are listed in Appendix D. They cover mathematical operations, such as finding the square root, and string functions, such as one that will count the number of characters in a string. The two functions we will discuss here are RND, which will generate a random number, and INT, which truncates a decimal number into an integer, or whole number.

Here are some occasions when a program needs the computer to generate a random number:

- Simulating the roll of dice.
- Dealing a hand of cards.
- Choosing numbers for a math facts quiz.

In the example of rolling the dice, we cannot predict what number will come up based on the numbers from the previous roll. It could be any number between 1 and 6. In the same way, we cannot predict the number of the next card dealt from a shuffled deck of cards. In the math quiz example, we want to be sure the numbers generated will be different each time the program is run. After all, it is a quiz! The random number function is used to produce a result similar to picking numbers out of a hat.

The random function on the Commodore 64 looks like this:

RND(number)

This function performs calculations to simulate choosing numbers at random. Here are the important points to remember when using this function:

- The function chooses a number within the range 0.0 - 1.0. The number is never exactly 0.0 or 1.0.
- The number in parentheses following RND is called a 'dummy argument.' It has an effect on the 'randomness' of the numbers produced by computer. Zero (0) or the variable TI will give the closest approximation to a truly random number. The examples in this book will use 0.

A demonstration is worth a thousand words. Try this short program and see what happens:

```
10 X = RND (0)
20 PRINT X
30 N = N + 1
40 IF N < 10 THEN 10
50 END
```

Run it several times and look at the numbers that are printed.

You will find more information about the random number function in the *Commodore 64 User's Guide*. You might like to find out more about it after working with it for a while.

Most of the time we do not need a number between 0 and 1. Here is an example of a random number statement to calculate a number between zero and ten. Again, the number will never be exactly zero or exactly ten.

```
10 X = RND(0) * 10
```

In this example, the random number function chooses a number between 0.0 and 1.0 and multiplies it by 10. Substitute this statement in the program and see what happens.

Questions & Answers

Write a statement to choose a random number between 0.0 and 50.00?

Answer: 10 X = RND (1) * 50

In most instances, including rolling dice or dealing playing cards, we need a random whole number, not a decimal number. The INT function will truncate the random number to a whole number. The INT function looks like this:

```
INT (<NUMBER>)
```

The number following the function is the decimal number, variable or expression to be truncated to an integer. For example, the INT function can be added to the random function in our program:

```
10 X = INT (RND(0)*10)
```

Now statement 10 will generate a whole number between 0 and 9.

When the random number is close to zero it will be truncated to zero, so the smallest value of X from this statement will be zero. When the random number is close to one, then the number to be truncated will be close to, but not quite 10. The highest value of X we will get will be nine. The highest value of X generated will always be one less than the multiplying number.

Try this new statement 10 in the previous program to generate random numbers. Run it a few times and look at the numbers printed.

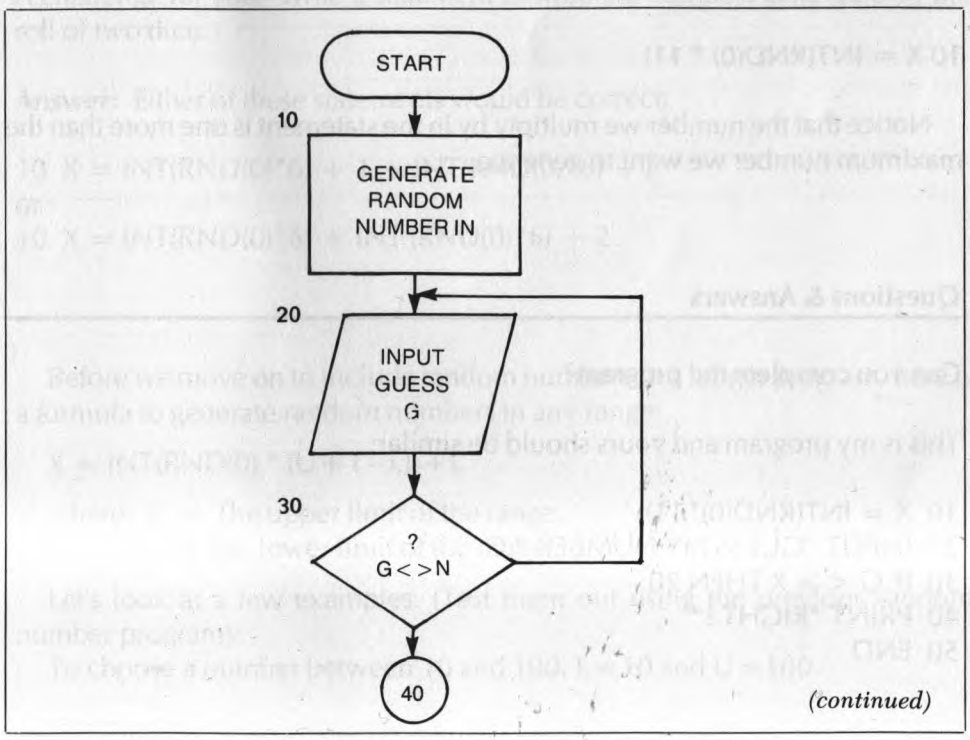
Questions & Answers

The INT function truncates a _____ number to an _____ or whole number.

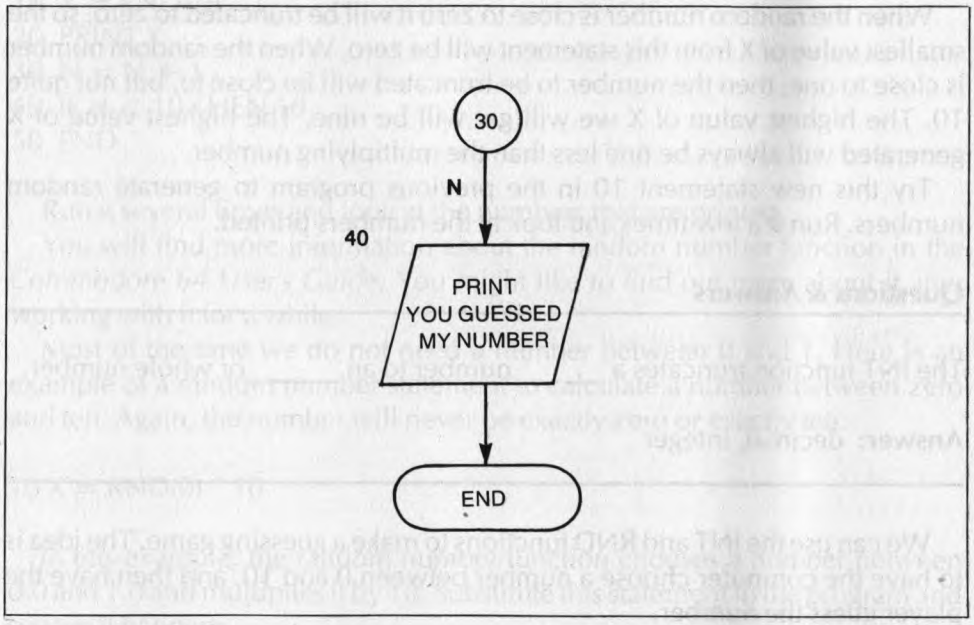
Answer: decimal, integer

We can use the INT and RND functions to make a guessing game. The idea is to have the computer choose a number between 0 and 10, and then have the player guess the number.

Here is the flowchart for the game:



(continued)



The statement to choose the random number will look like this:

```
10 X = INT(RND(0) * 11)
```

Notice that the number we multiply by in the statement is one more than the maximum number we want to generate.

Questions & Answers

Can you complete the program?

This is my program and yours should be similar:

```
10  X = INT(RND(0)*11)
20  INPUT "GUESS MY NUMBER ";G
30  IF G < > X THEN 20
40  PRINT "RIGHT!"
50  END
```

Now let's use these functions to program a more complicated simulation, such as the roll of a dice. We must generate a random integer number between 1 and 6. Look at this statement:

```
10 X = INT(RND(0) * 7)
```

The number generated in this statement will not be greater than 6, but it could be 0. While this would not generate an error in the program, it is a "real world" logical error because 0 does not appear on a dice.

This statement will solve the problem.

```
10  X = INT(RND(0)*6)+1
```

Now 1 is added to the random integer each time it is generated. When the random integer is 0 then the value of X is 1, and when the random integer is 1 then the value of X will be 6.

Questions & Answers

A challenge for you - write a statement to simulate numbers generated by the roll of two dice.

Answer: Either of these statements would be correct:

```
10  X = INT(RND(0)*6) + 1 + INT(RND(0)*6) +1
or
10  X = INT(RND(0)*6) + INT (RND(0)*6) + 2
```

Before we move on to include random numbers in a longer program, here is a formula to generate random numbers in any range:

```
X = INT(RND(0) * (U+1-L))+L
```

where: U = The upper limit of the range.
L = The lower limit of the range.

Let's look at a few examples. (Test them out using the previous random number program):

To choose a number between 10 and 100: L=10 and U=100

X = INT(RND(0)*100+1−10)+10

This could be shortened to:

X = INT(RND(0)*91)+10

To choose a number between 0 and 7: L=0 and U=7

X = INT(RND(0)*8)

To choose a number between 1 and 13: L=1 and U=13

X = INT(RND(0)*13)+1

Questions & Answers

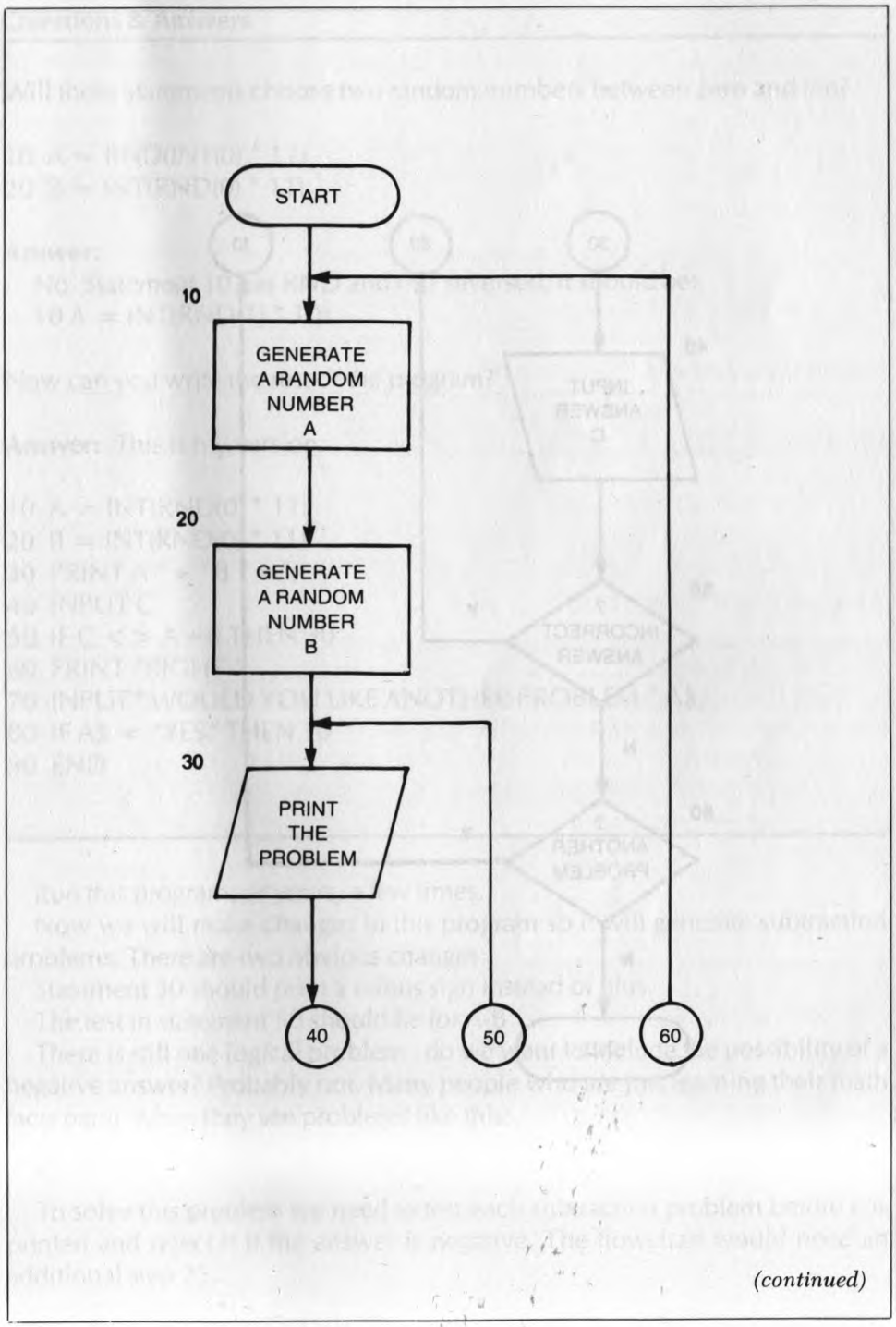
What are the upper and lower limits for the range of random numbers in this statement?

X = INT(RND(0)*52)+1

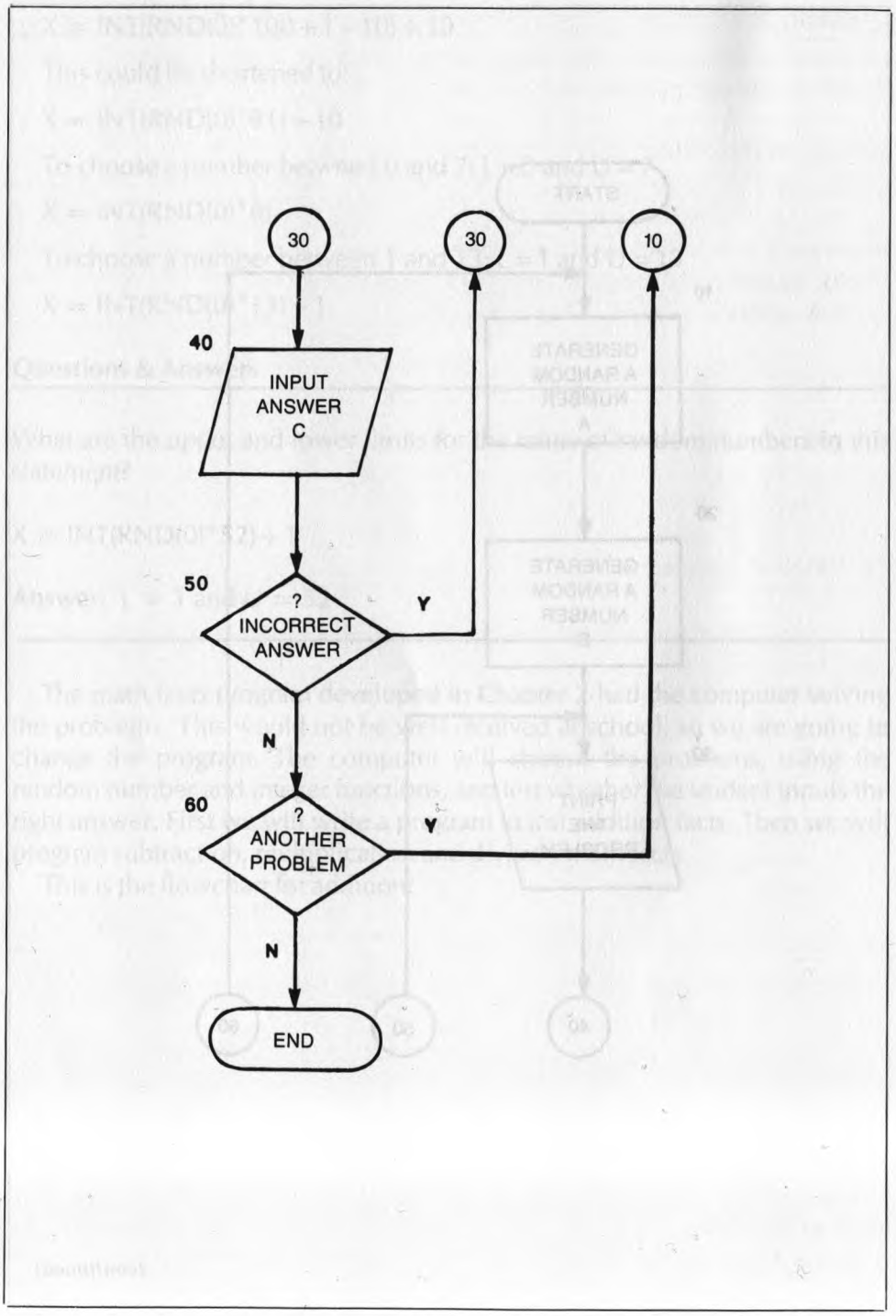
Answer: L = 1 and U = 52

The math facts program developed in Chapter 2 had the computer solving the problems. This would not be well received at school, so we are going to change the program. The computer will choose the problems, using the random number and integer functions, and test whether the student inputs the right answer. First we will write a program to test addition facts. Then we will program subtraction, multiplication and division math facts.

This is the flowchart for addition:


START
10
GENERATE A RANDOM NUMBER A
20
GENERATE A RANDOM NUMBER B
30
PRINT THE PROBLEM
40
50
60

(continued)


30
30
10
40
INPUT
ANSWER
C
50
?
INCORRECT
ANSWER
Y
N
60
?
ANOTHER
PROBLEM
Y
N
END

Questions & Answers

Will these statements choose two random numbers between zero and ten?

```
10 A = RND(INT(0) * 11)
20 B = INT(RND(0) * 11)
```

Answer:

No. Statement 10 has RND and INT reversed. It should be:

```
10 A = INT(RND(1) * 10)
```

Now can you write the rest of the program?

Answer: This is my version:

```
10 A = INT(RND(0) * 11)
20 B = INT(RND(0) * 11)
30 PRINT A "+" B "=";
40 INPUT C
50 IF C <> A+B THEN 30
60 PRINT "RIGHT "
70 INPUT" WOULD YOU LIKE ANOTHER PROBLEM ";A$
80 IF A$ = "YES" THEN 10
90 END
```

Run this program, or yours, a few times.

Now we will make changes in this program so it will generate subtraction problems. There are two obvious changes:

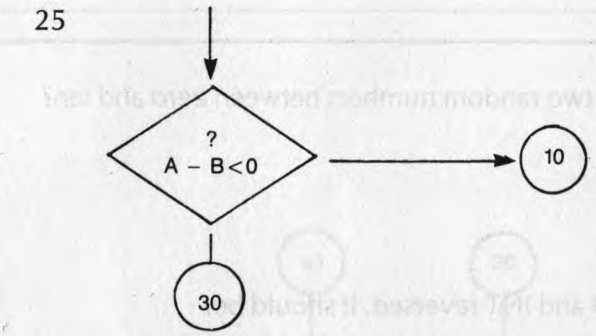
Statement 30 should print a minus sign instead of plus.

The test in statement 50 should be for A-B.

There is still one logical problem - do we want to include the possibility of a negative answer? Probably not. Many people who are just learning their math facts panic when they see problems like this:

2 – 6 =

To solve this problem we need to test each subtraction problem before it is printed and reject it if the answer is negative. The flowchart would need an additional step 25.



The program will work with the changes to statements 30 and 50, and the addition of statement 25 to test if A is less than B.

Questions & Answers

Can you write statement 25?

```
10 A = INT(RND(0) * 11)
20 B = INT(RND(0) * 11)
25
30 PRINT A "–" B "=";
40 INPUT C
50 IF C <> A – B THEN 30
60 PRINT "RIGHT"
70 INPUT " DO YOU WANT ANOTHER PROBLEM ";A$
80 IF A$ = "YES" THEN 10
90 END
```

Answer: 25 IF A–B < 0 THEN 10

Changing the addition facts program to a multiplication facts program should not be difficult because it will not matter if A is less than B.

Questions & Answers

Change the addition program to a multiplication program.

Answer:

```
10 A = INT(RND(0) * 11)
20 B = INT(RND(0) * 11)
30 PRINT A "X" B "=";
40 INPUT C
50 IF C <> A*B THEN 30
60 PRINT" RIGHT "
70 INPUT "DO YOU WANT ANOTHER PROBLEM ";A$
80 IF A$ = "YES" THEN 10
90 END
```

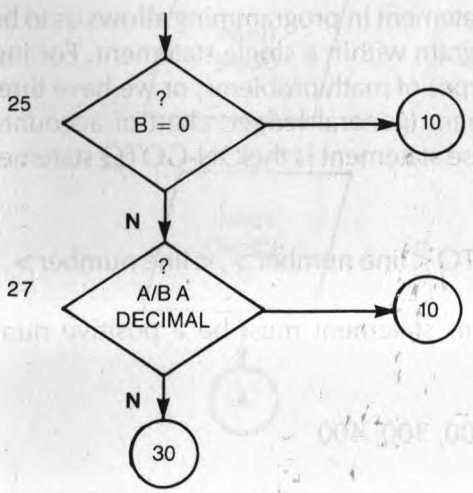
Division problems present two potential pitfalls. First, division by zero is always illegal. Second, since this program is for beginning students, we do not want the answer to be a decimal. For example, 4/2 (2) or 9/3 (3) would be acceptable but 5/2 (2.5) or 9/4 (2.25) would not.

The program will have to test for both of these conditions before the problem is printed.

Questions & Answers

This time, you draw the steps to add to the flowchart. We need step 25 to test for division by zero and step 27 to test for an answer that is an integer.

Answer: The program looks like this:



```
10 A = INT(RND(0) * 11)
20 B = INT(RND(0) * 11)
25 IF B = 0 THEN 20
27 IF A/B < > INT (A/B) THEN 10
30 PRINT A "/" B "=";
40 INPUT C
50 IF C < > A/B THEN 50
60 PRINT "RIGHT"
70 INPUT "DO YOU WANT ANOTHER PROBLEM ";A$
80 IF A$ = "YES" THEN 10
90 END
```

Statement 25 is straightforward: B is rejected if it is 0. Statement 27 uses the fact that the only time A/B equals INT(A/B) is when A/B is a whole number.

Can statements 25 and 27 be reversed?

Answer: No. This would give a division by 0 error in statement 27, if B is 0. We must test for division by 0 first.

ON-GOTO Statements

Now we have the programs written for the four different kinds of math facts problems. We can put the four programs together in one program using a different branching technique.

The ON-GOTO statement in programming allows us to branch to more than one place in the program within a single statement. For instance, we want to choose one of four types of math problems, or we have three different options in an accounts program (general ledger, chart of accounts up-date, balance sheet). The BASIC case statement is the ON-GOTO statement and it looks like this:

ON <variable> GOTO <line number>, <line number>, ...

The variable in this statement must be a positive numeric variable. For example:

ON C GOTO 100, 200, 300, 400

In this example, if the value of the variable, C, is 1, then the program branches to the statement 100; if C is 4, then the program will go to statement 400.

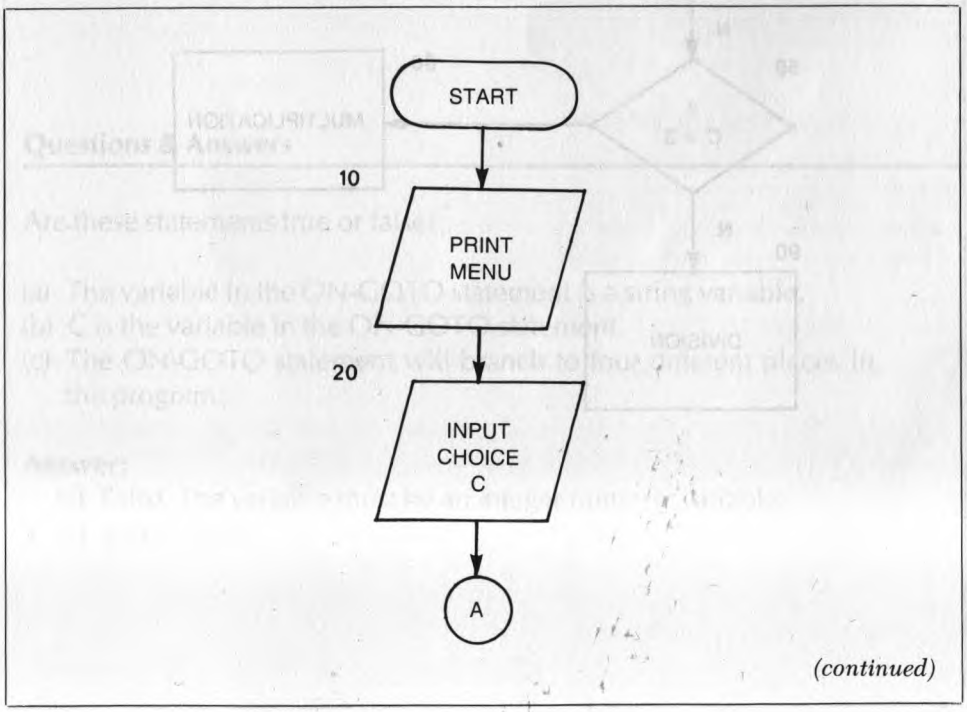
Questions & Answers

Where will the program branch when C is:

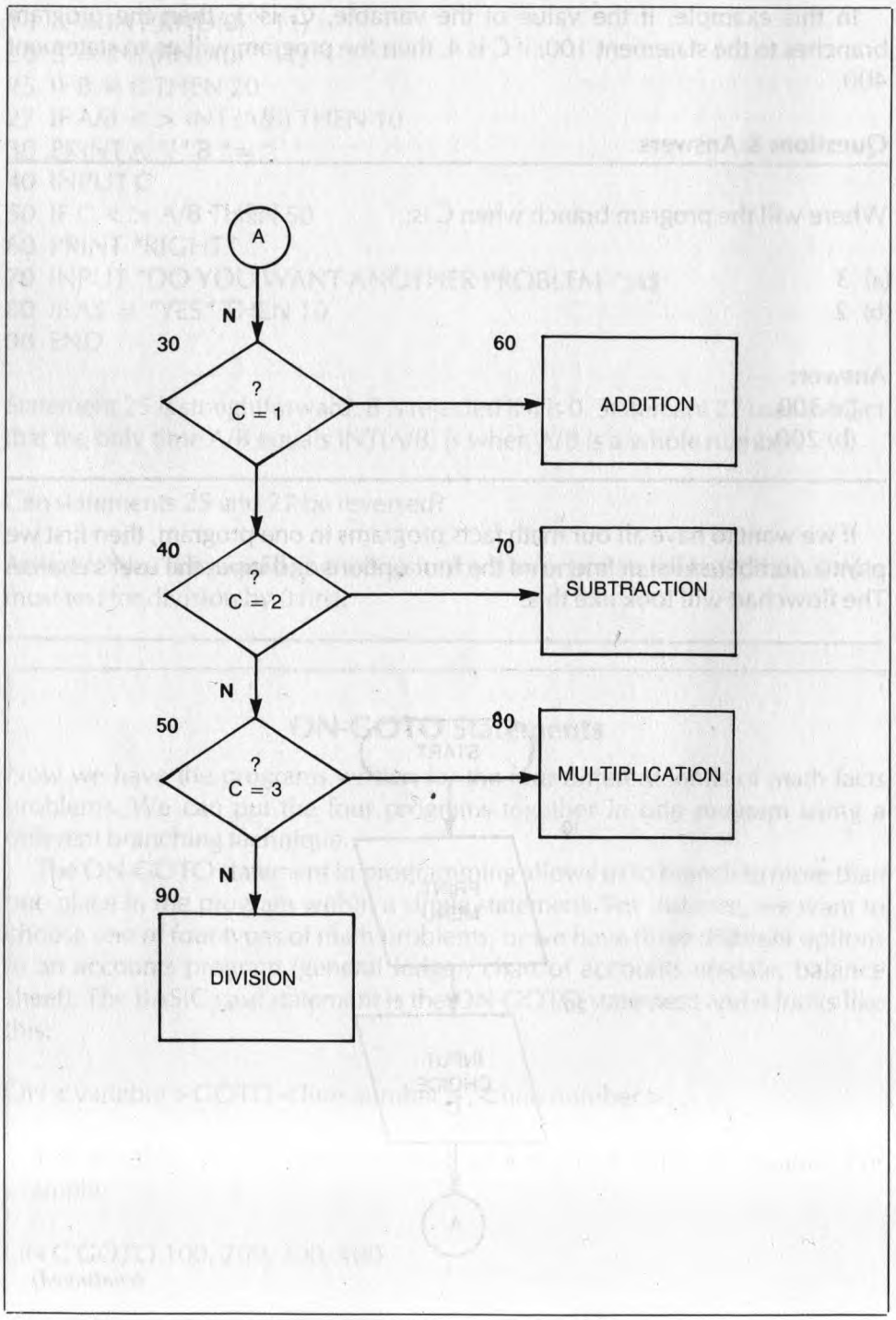
(a) 3
(b) 2

Answer:
(a) 300
(b) 200

If we want to have all our math facts programs in one program, then first we print a numbered list or 'menu' of the four options and input the user's choice. The flowchart will look like this:



(continued)


A
N
30
? C = 1
60
ADDITION
40
? C = 2
70
SUBTRACTION
N
50
? C = 3
80
MULTIPLICATION
N
90
DIVISION

The first part of the program to print the menu, will be easy, but there are some finishing touches we can add to make it easy to read. The screen should be cleared before the menu is printed, and the menu should be centered on the screen. Also, the color of the print can be changed. I have decided to print my output in white, by including the CTRL and color key as the first character in the title statement.

The menu looks like this:

```
10 PRINT"□"
20 PRINT TAB(10)"■MATH FACTS PROGRAM"
25 PRINT
30 PRINTTAB(5)"ON THE MENU TODAY WE HAVE:"
35 PRINT
40 PRINT "1 ADDITION PROBLEMS"
50 PRINT "2 SUBTRACTION PROBLEMS"
60 PRINT "3 MULTIPLICATION PROBLEMS"
70 PRINT "4 DIVISION PROBLEMS"
75 PRINT
80 INPUT " YOUR CHOICE ";C
90 ON C GOTO 100,200,300,400
```

Questions & Answers

Are these statements true or false?

(a) The variable in the ON-GOTO statement is a string variable.
(b) C is the variable in the ON-GOTO statement.
(c) The ON-GOTO statement will branch to four different places in the program.

Answer:

(a) False. The variable must be an integer numeric variable.
(b) and (c) True.

REM Statements

This program is going to be longer than the others we have written. It will be easier to read and debug if we add notes. To do this we can use a BASIC statement, REM, which stands for REMark. REM statements are for the user's benefit, not for execution by the computer. REM statements look like this:

REM <remark>

Here is a REM statement for the beginning of the sequence of statements for addition problems:

100 REM ADDITION PROBLEMS

Questions & Answers

Now, add the programs we have already written for addition, subtraction, multiplication and division with an appropriate remark statement at the start of each one.

Answer: Here is the listing for my completed program:

```
10 PRINT"□"
20 PRINT TAB(10)"■MATH FACTS PROGRAM"
25 PRINT
30 PRINTTAB(5)"ON THE MENU TODAY WE HAVE:"
35 PRINT
40 PRINT "1 ADDITION PROBLEMS"
50 PRINT "2 SUBTRACTION PROBLEMS"
60 PRINT "3 MULTIPLICATION PROBLEMS"
70 PRINT "4 DIVISION PROBLEMS"
75 PRINT
80 INPUT " YOUR CHOICE ";C
90 ON C GOTO 100,200,300,400
100 REM **** ADDITION FACTS ****
105 PRINT"□"
110 A = INT(RND(0) *11)
120 B = INT(RND(0) *11)
130 PRINT A " + " B " = ";
140 INPUT C
150 IF C <> A+B THEN 30
160 PRINT "RIGHT "
170 INPUT" WOULD YOU LIKE ANOTHER PROBLEM ";A$
180 IF A$="YES" THEN 110
190 END
200 REM **** SUBTRACTION FACTS ****
```

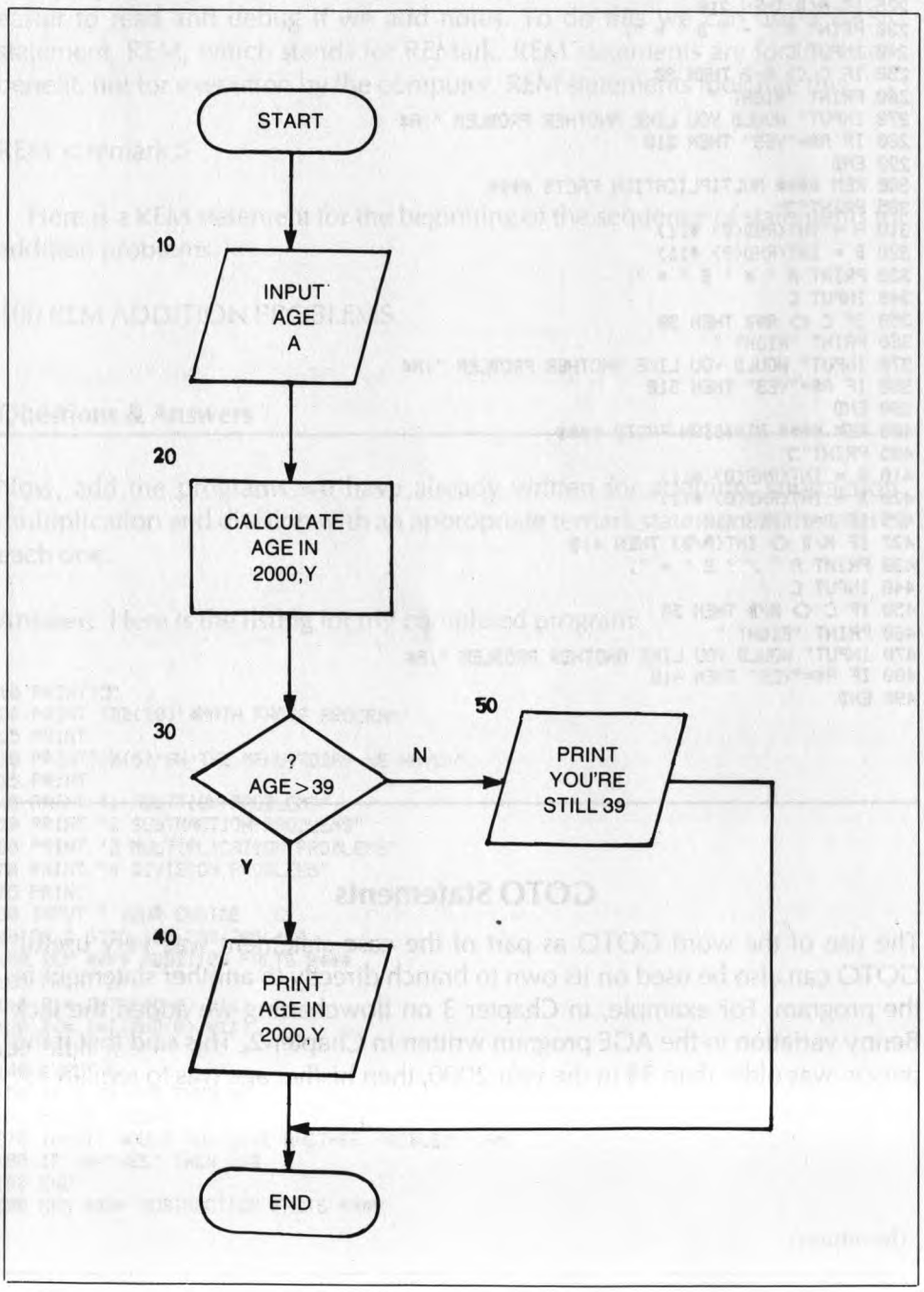
(continued)

```
205 PRINT"□"
210 A = INT(RND(0) *11)
220 B = INT(RND(0) *11)
225 IF A<B THEN 210
230 PRINT A " - " B " = ";
240 INPUT C
250 IF C <> A-B THEN 30
260 PRINT "RIGHT "
270 INPUT" WOULD YOU LIKE ANOTHER PROBLEM ";A$
280 IF A$="YES" THEN 210
290 END
300 REM **** MULTIPLICATION FACTS ****
305 PRINT"□"
310 A = INT(RND(0) *11)
320 B = INT(RND(0) *11)
330 PRINT A " * " B " = ";
340 INPUT C
350 IF C <> A*B THEN 30
360 PRINT "RIGHT "
370 INPUT" WOULD YOU LIKE ANOTHER PROBLEM ";A$
380 IF A$="YES" THEN 310
390 END
400 REM **** DIVISION FACTS ****
405 PRINT"□"
410 A = INT(RND(0) *11)
420 B = INT(RND(0) *11)
425 IF B=0 THEN 410
427 IF A/B <> INT(A/B) THEN 410
430 PRINT A " / " B " = ";
440 INPUT C
450 IF C <> A/B THEN 30
460 PRINT "RIGHT "
470 INPUT" WOULD YOU LIKE ANOTHER PROBLEM ";A$
480 IF A$="YES" THEN 410
490 END
```

GOTO Statements

The use of the word GOTO as part of the case statement was very useful. GOTO can also be used on its own to branch directly to another statement in the program. For example, in Chapter 3 on flowcharting we added the Jack Benny variation to the AGE program written in Chapter 2. This said that if the person was older than 39 in the year 2000, then his/her age was to remain 39.

The flowchart looked like this:



In this program we want to print one of two different messages as a result of the test in step 40, but we do not want to print BOTH of them. We can use a GOTO statement like this:

```
10 INPUT "YOUR NAME";N$
20 INPUT "YOUR AGE ";A
30 Y = A + 2000 - 1984
40 IF Y > 39 THEN 70
50 PRINT N$; " IN THE YEAR 2000 YOU WILL BE ";Y
60 GOTO 80
70 PRINT N$;" IN THE YEAR 2000 YOU WILL BE 39 AGAIN!"
80 END
```

If the calculated age, Y, is less than 39 then statement 50 will be executed and Y will be printed. Then statement 60 will send the program directly to the END in statement 80, bypassing statement 70.

CAUTION

Avoid GOTO statements whenever possible. Decisions and case statements are control structures within the program that are used to control the flow of the program. On the other hand, GOTOs are not dependent on the data, and they interrupt the flow of the program. Trying to debug a program with lots of GOTO statements is like trying to unravel a plate of spaghetti! It is a good programming habit to save the GOTO statement for the last resort.

Here are some suggestions for avoiding GOTOs. Sometimes reversing the operator in the IF statement will remove a GOTO. For example, if when X is 99, we want to end the program. Here is one set of statements to do this:

```
100 IF X = 99 THEN 120
110 GOTO 20
120 END
```

Here is the same thing without the GOTO.

```
100 IF X <> 99 THEN 20
120 STOP
```

Don't you think that is better?

Questions & Answers

Can you rewrite this example to remove the GOTO?

```
490 INPUT "DO YOU WANT TO PLAY AGAIN";A$
500 IF A$ = "NO" THEN 520
510 GOTO 10
520 END
```

Answer:

```
490 INPUT "DO YOU WANT TO PLAY AGAIN";A$
500 IF A$ = "YES" THEN 10
510 END
```

The overuse of GOTOs and muddled programming occurs when the problem is not stated clearly and the flowchart is drawn with a mass of lines crossing over each other.

Sam runs a small business making spaghetti sauce, and he has decided to offer a discount. He ran this advertisement in the local paper:

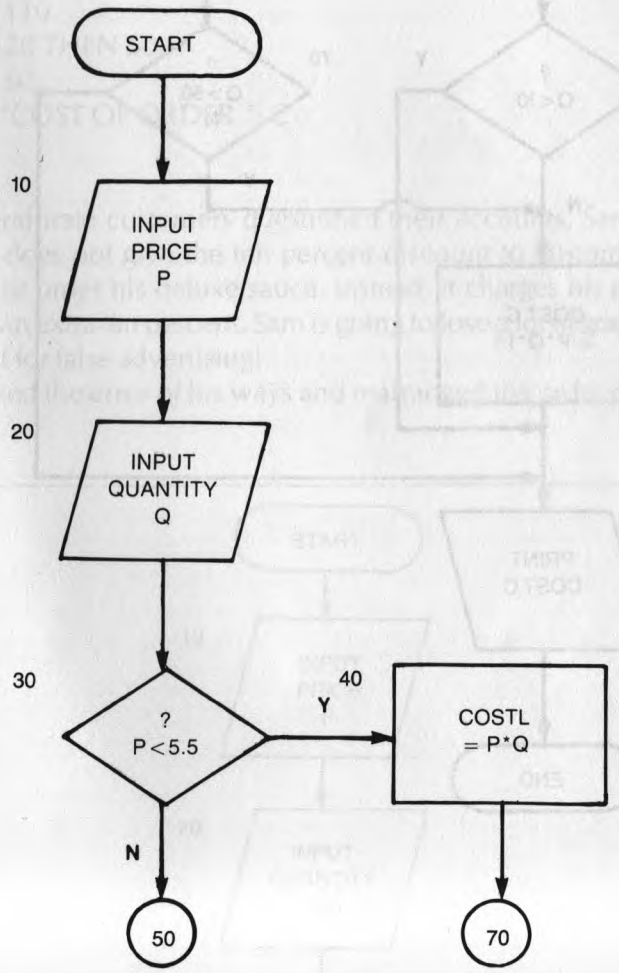
SAM'S SPECIAL

Deluxe Spaghetti Sauce $5.50 per case
10% discount on all orders of 10 cases or more.

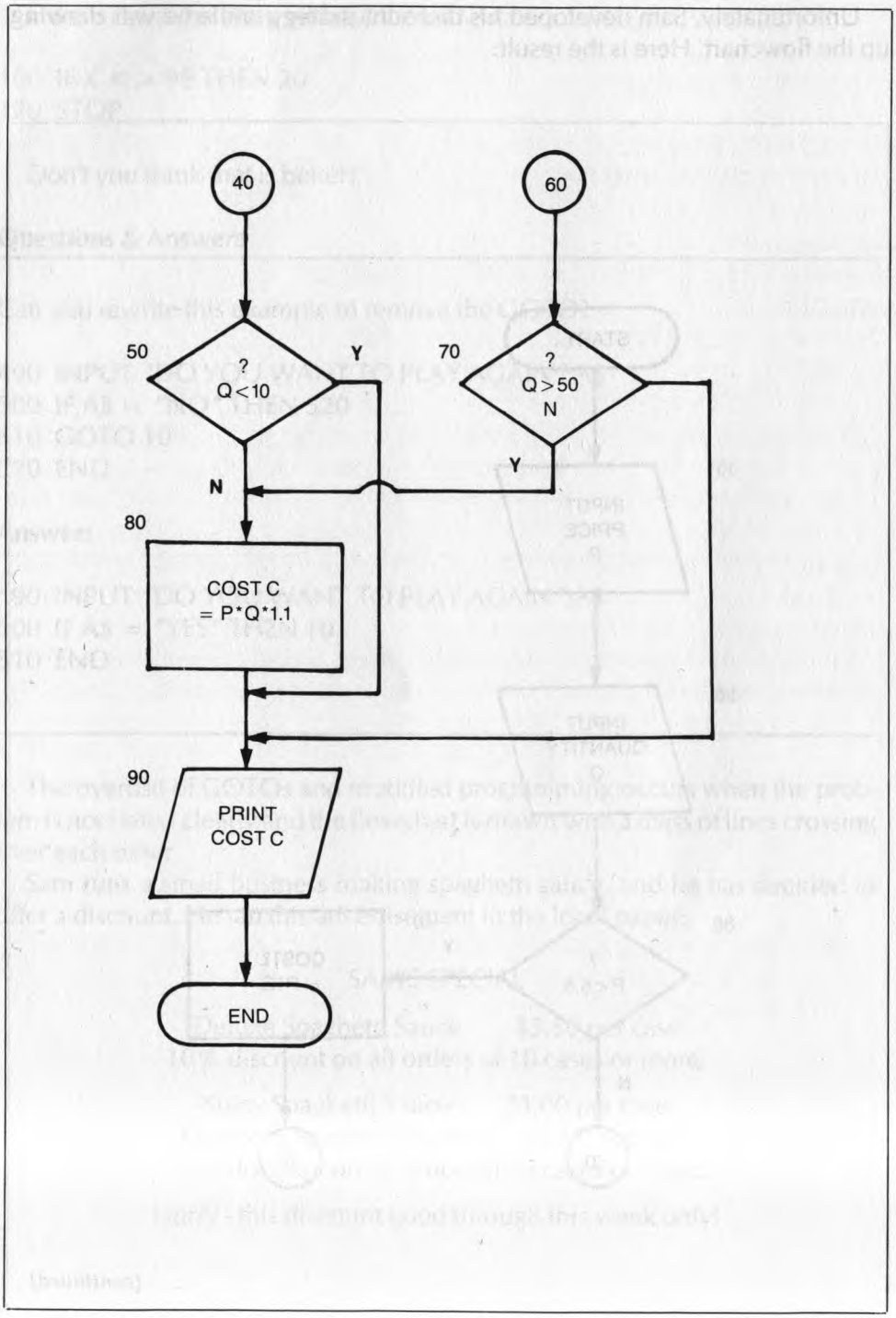
Spicy Spaghetti Sauce $3.00 per case
Economy Spaghetti Sauce $2.00 per case
10% discount on all orders of 20 cases or more.

Hurry - this discount good through this week only!

Unfortunately, Sam developed his discount strategy while he was drawing up the flowchart. Here is the result:



(continued)

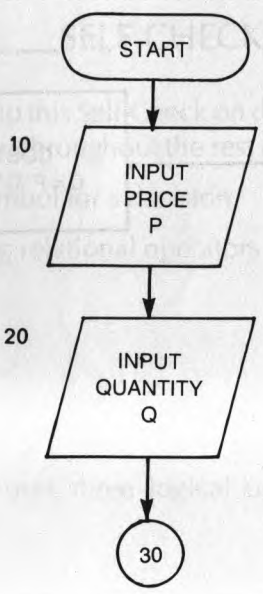

40
60
50
?
Q<10
Y
70
?
Q>50
N
Y
N
80
COST C
= P*Q*1.1
90
PRINT
COST C
END

Sam's first attempt to write his discount program looked like this:

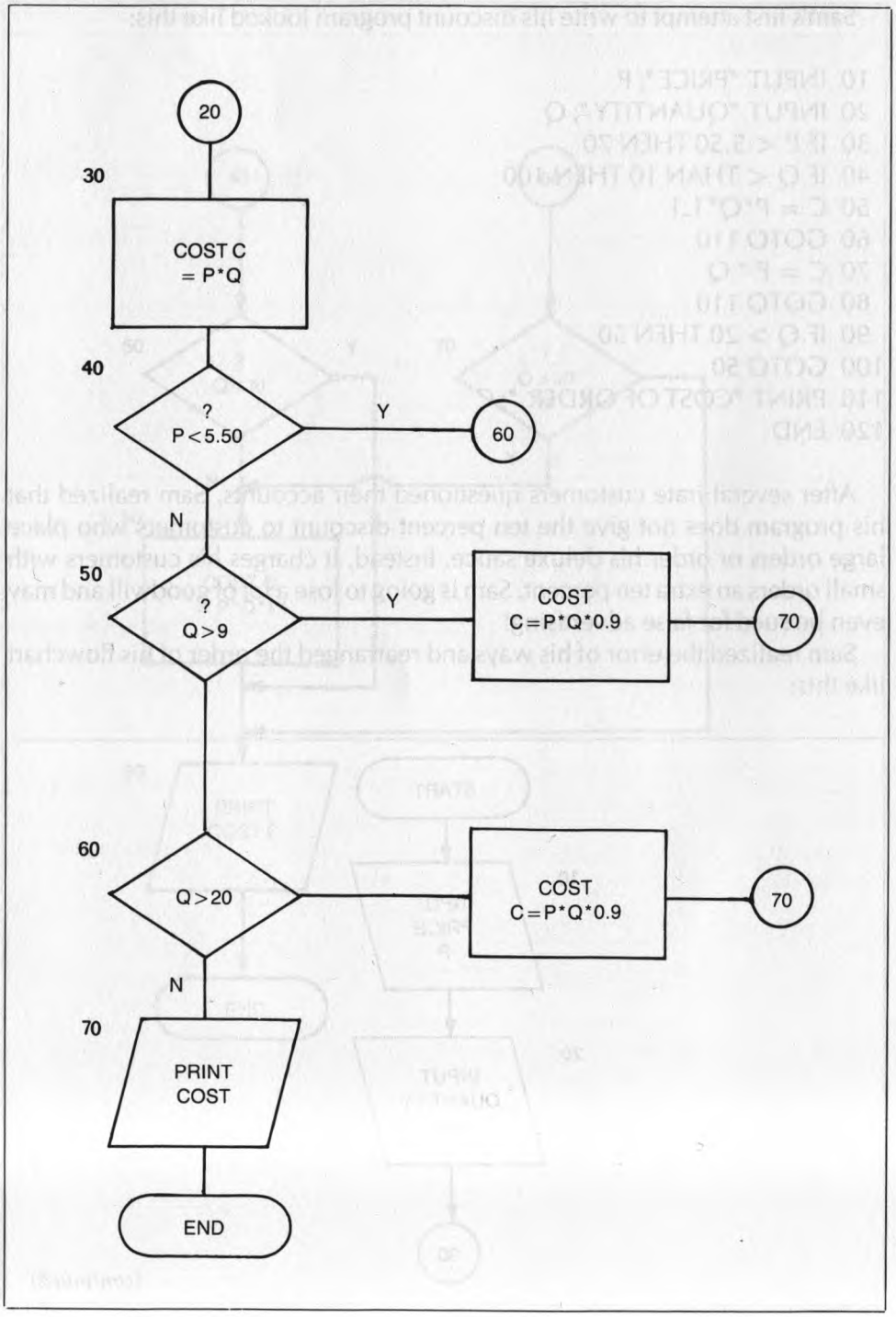
```
 10 INPUT "PRICE"; P
 20 INPUT "QUANTITY"; Q
 30 IF P < 5.50 THEN 70
 40 IF Q < THAN 10 THEN 100
 50 C = P*Q*1.1
 60 GOTO 110
 70 C = P * Q
 80 GOTO 110
 90 IF Q > 20 THEN 50
100 GOTO 50
110 PRINT "COST OF ORDER "; C
120 END
```

After several irate customers questioned their accounts, Sam realized that his program does not give the ten percent discount to customers who place large orders or order his deluxe sauce. Instead, it charges his customers with small orders an extra ten percent. Sam is going to lose a lot of goodwill and may even be sued for false advertising!

Sam realized the error of his ways and rearranged the order of his flowchart like this:



(continued)


20
30
COST C
= P*Q
40
?
P<5.50
Y
60
N
50
?
Q>9
Y
COST
C=P*Q*0.9
70
60
Q>20
COST
C=P*Q*0.9
70
N
70
PRINT
COST
END

Questions & Answers

Sam wrote a program without any GOTOs. Can you do it?

Answer: This is Sam's program:

```
10 INPUT "PRICE"; P
20 INPUT "QUANTITY";Q
30 C = P * Q
40 IF P<5.50 THEN 60
50 IF Q > 9 THEN C = P * Q*0.9
60 IF Q > 20 THEN C = P * Q*0.9
70 PRINT " TOTAL COST OF ORDER IS ";C
80 END
```

The moral of this story is: it does indeed pay to prepare your program before you start to write the BASIC code.

In the next chapter there will be some more thoughts on the GOTO statement, as well as useful information on how to make your program repeat. Before you move on, here is a short Self-Check on decisions.

SELF-CHECK

Please do not decide to skip this Self-Check on decisions! The material covered in this chapter will be used throughout the rest of the book.

1. Draw the flowchart symbol for a decision.
2. Which of the following relational operators will give the answer "true" if X = 5?
   (a) 0 < X < 5
   (b) 0 < X < 6
   (c) 4 < X < 6
   (d) 4 < X < 5
3. The Commodore 64 uses three logical operators _____, _____, and _____.

4. Write an IF statement to test whether A is less than B.

5. Look at these statements.

```
10 IF A > B OR NOT C = B THEN 100
20 PRINT A, B, C
```

Which will be executed when A, B and C have the following values:

(a) A = 5, B = 3, C = 1
(b) A = 5, B = 3, C = 3
(c) A = 3, B = 5, C = 5
(d) A = 5, B = 3, C = 5

6. Write a statement to choose a random whole number, X, between 0 and 8.

7. How would you write the statements to choose one card from each suit of a deck of cards?

8. Mrs. Stockard would like a spelling program for her class that would allow the students to choose between four levels of difficulty. Write the statements for the menu to appear at the beginning of the program and the case statement using the student's choice.

Answers to SELF-CHECK

1. If you couldn't remember, refer back to this chapter and Chapter 3.

2. (a) No. This will be true for numbers between 1 and 4.
   (b) No. This will be true for numbers between 1 and 5.
   (c) Right! This will be true for five only.
   (d) No. This will never be true.

3. AND, OR and NOT.

4. 10 IF A<B THEN 100

   It doesn't matter which statement numbers you used, but the rest of the statement must be exactly like mine.

5. In this question, the OR means that the whole statement will be true if both tests are true, or if one of the tests is true.

(a) True because NOT C = B is true.
(b) True because A > B is true.
(c) False because both tests are false.
(d) True because both tests are true.

6. 10 X = INT(RND(0) * 9)

7. This is a little more tricky than question 6. It will take four statements and the numbers should be between 1 and 13.

```
10 H = INT(RND(0)*13) +1
20 D = INT(RND(0)*13) +1
30 C = INT(RND(0)*13) +1
40 S = INT(RND(0)*13) +1
```

8. The answer to this problem should include the PRINT statements to display the choices on the screen, an INPUT statement to input the children's choice, and an ON-GOTO statement to branch to the four levels.

Here is my version:

```
10 PRINT " MRS.STOCKARD'S SUPER SPELLING SPECTACULAR"
20 PRINT " THERE ARE FOUR LEVELS FOR YOU TO TRY"
30 PRINT " 1 START"
40 PRINT " 2 SAIL"
50 PRINT " 3 SPRINT"
60 PRINT " 4 SOAR"
70 INPUT "YOUR CHOICE ";C
80 ON C GOTO 100,200,300,400
90 END
```

Suggestions for Further Decision-Making

1. Take a few minutes to draw a simple flowchart for your programs that involve decisions, to avoid muddled programs that end up like 'plates of spaghetti.'
2. Write a program using IF statements to select all the stamps in your collection that are American, in good condition, and printed before 1930.
3. This is a much harder problem. Use the integer and random functions to write a program to guess a person's age. Start from a fixed number such as 10 and ask the person to tell the computer if his/her guess is H, too high, or L, too low.

CHAPTER 5

Repetitions

Now we are going to find out how to program REPETITIONS. There are many instances when all or part of a program needs to be repeated. For example:

Entering all the checks for one month.
Making up a set of ten math problems.
Printing out the days of the week or the months of the year.
Taking turns playing a game such as tic-tac-toe.

We discussed how to flowchart the logic for repetitions in Chapter 4. Now we will learn how to program them.

Also in this chapter, more pre-programmed functions will be introduced. These will include mathematical functions such as the trigonometric functions and string functions.

Another new topic in this chapter will be an alternative way to input information into a program.

By the end of this chapter you will be able to:

- Write a loop sequence using the BASIC statements FOR-NEXT and STEP.
- Use the concept of a counter.
- Use punctuation at the end of a PRINT statement effectively.
- Understand the use of nested loops.
- Use BASIC functions SIN, COS, LEFT$, RIGHT$ and MID$.
- Input data into a program using READ and DATA statements.
- Understand the concept of subscripted variables or ARRAYS.

- Know when to use a dimension, DIM statement.
- Use arrays in programs.
- Write your own functions for long or complicated expressions which will be repeated in the program.
- Use the BASIC statements GOSUB and RETURN to create SUB-ROUTINES in a program.

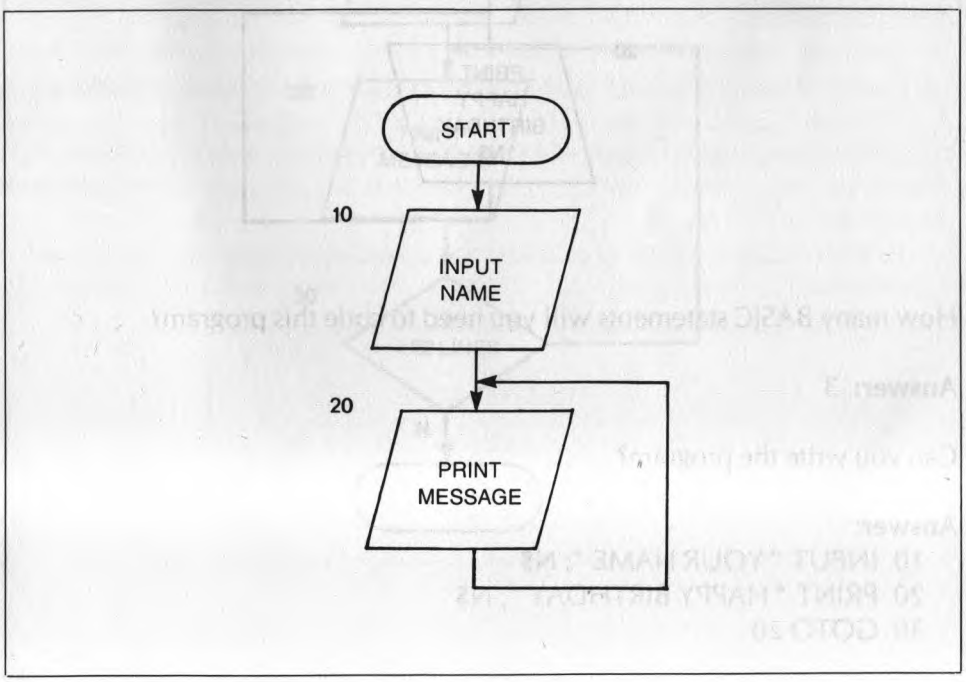
Programming Repetitions

We will start with the first program we wrote in Chapter 3, the NAME program. This program inputs a person's name and then prints "HELLO" and the person's name on the screen.

Here is the program:

```
10 INPUT "YOUR NAME "; N$
20 PRINT "HELLO "; N$
```

One way to make this program more interesting is to continue printing the message all the way down the screen. To do this, the control of the program needs to keep returning to statement 20. The flowchart would look like this:



The BASIC statement to make statement 20 repeat is the GOTO statement introduced in Chapter 4. Add this statement to the NAME program:

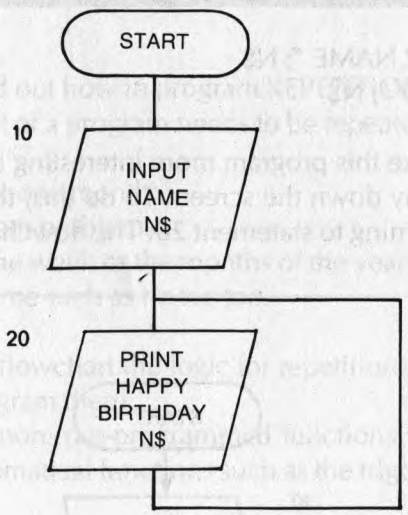
```
30 GOTO 20
```

Now RUN the program. The message does indeed print over and over again. When will it stop? Never, unless you use the RUN STOP key.

Questions & Answers

How would you draw a flowchart to input someone's name and have the computer continue printing "Happy Birthday" to the person?

Answer:



How many BASIC statements will you need to code this program?

Answer: 3

Can you write the program?

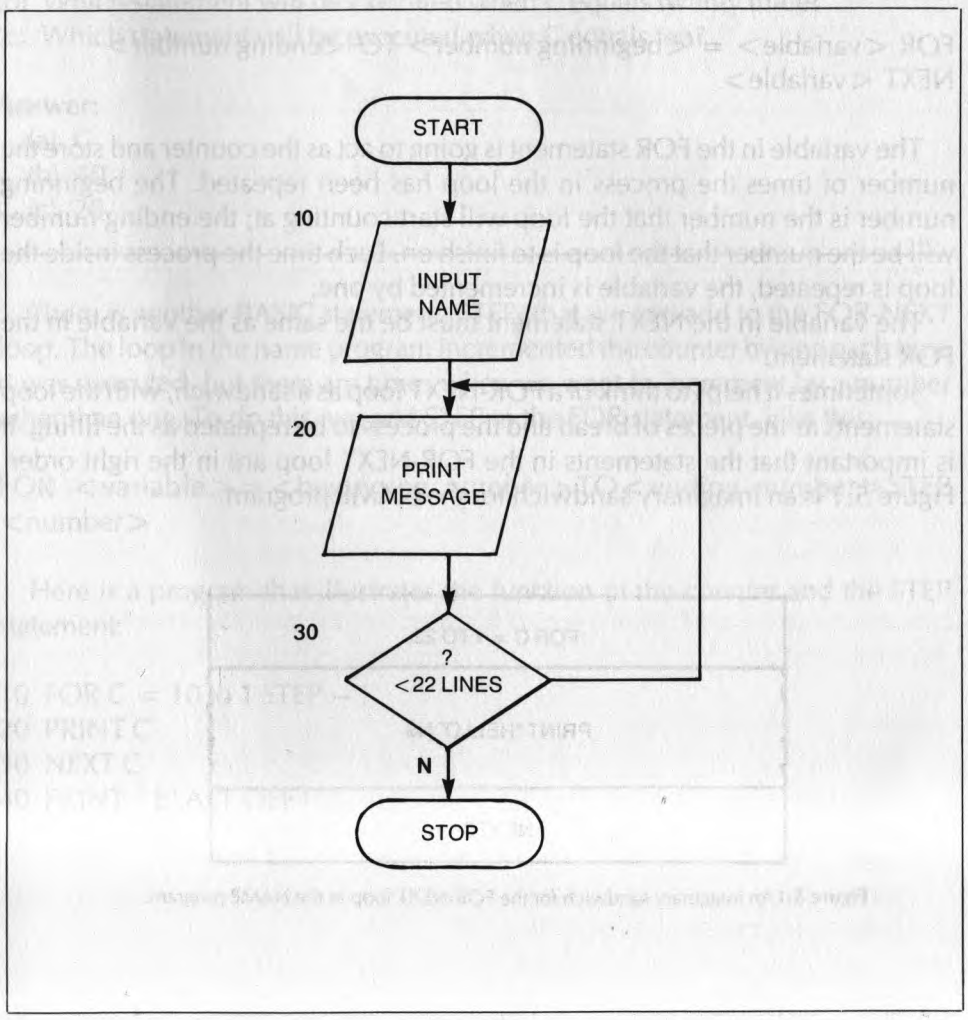
Answer:

```
10 INPUT " YOUR NAME "; N$
20 PRINT " HAPPY BIRTHDAY "; N$
30 GOTO 20
```

Using the GOTO statement this way creates 'endless' loops that will not end unless the user stops the program with the RUN STOP key. If an endless loop is buried in a longer program, then the program will loop around until it is stopped and will not complete its task.

There are alternatives to using GOTO to repeat program steps. Using the NAME program again, we will change it so that the message is listed down the screen and stops when it reaches the bottom. To do this the PRINT statement will be limited to twenty-two repetitions. We will need a logical DO-UNTIL loop that will stop printing after the twenty second repetition. You might want to refer back to Chapter 3 and try to flowchart the problem for yourself.

Here is my flowchart:



Notice that in step 30 the flowchart tests for the number of lines printed. This means we need a COUNTER in the program somewhere. The counter will keep track of the number of times the program has executed the loop. As soon as the counter exceeds twenty two, the flowchart comes out of the loop and ends.

FOR-NEXT Loops

We will need the BASIC statements FOR and NEXT to create this loop. The FOR statement goes at the beginning of the sequence of statements to be repeated, and the NEXT statement goes at the end. Both are necessary to complete the loop. The statements look like this:

FOR <variable> = <beginning number> TO <ending number>
NEXT <variable>

The variable in the FOR statement is going to act as the counter and store the number of times the process in the loop has been repeated. The beginning number is the number that the loop will start counting at; the ending number will be the number that the loop is to finish on. Each time the process inside the loop is repeated, the variable is incremented by one.

The variable in the NEXT statement must be the same as the variable in the FOR statement.

Sometimes it helps to think of a FOR-NEXT loop as a sandwich, with the loop statements as the pieces of bread and the process to be repeated as the filling. It is important that the statements in the FOR-NEXT loop are in the right order. Figure 5.1 is an imaginary sandwich for the NAME program.

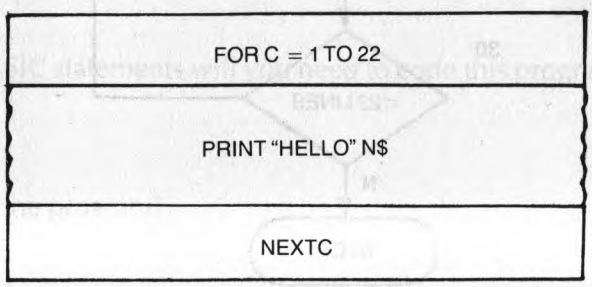


Figure 5.1 An imaginary sandwich for the FOR-NEXT loop in the NAME program.

The NAME program including the loop will look like this:

```
10 INPUT " YOUR NAME ";N$
15 FOR C = 1 TO 22
20 PRINT " HELLO "; N$
30 NEXT C
40 END
```

Questions & Answers

(a) What variable name was used in the FOR statement?
(b) Which statement will be executed when C equals twenty three?
(c) Which statement will be executed when C equals ten?

Answer:
(a) C
(b) 40
(c) 20

There is another BASIC statement; STEP, that we can add to the FOR-NEXT loop. The loop in the name program incremented the counter by one each time it was executed, but there are times when we want to increment by a number other than one. To do this, we add STEP to the FOR statement. Like this:

FOR <variable> = <beginning number> TO <ending number> STEP <number>

Here is a program that illustrates the function of the counter and the STEP statement:

```
10 FOR C = 10 to 1 STEP -1
20 PRINT C
30 NEXT C
40 PRINT " BLAST OFF !"
```

When this program is executed, the output illustrates the construction of the FOR-NEXT loop very well. The process inside the loop is to print the counter, C, and the first number printed is 10. Each time the loop is executed, the STEP statement increments the counter by minus one, resulting in a countdown on the screen.

Questions & Answers

How would you change the COUNTDOWN program to count down from 1000?

Answer:

Change the beginning number in the FOR statement to 1000.

```
10 FOR C = 1000 TO 1 STEP -1
```

If STEP is not used, the computer assumes that you want an increment of 1. When the FOR-NEXT loop was added to the NAME program, the program repeated the message on the next line on the screen. However, it is possible to write a program that will fill the screen by printing across the screen as well as down the screen. To do this, the punctuation at the end of the print statement is changed. At the moment, the program looks like this:

```
10 INPUT "YOUR NAME ";N$
15 FOR C = 1 TO 22
20 PRINT "HELLO "; N$
30 NEXT C
40 END
```

If a semicolon is added to the end of statement 20, it will change the output. Try this version:

```
20 PRINT "HELLO ";N$;
```

Remember that you can make changes to a program listed on the screen by using the cursor control keys. Do not forget to enter the changes by pressing RETURN.

The semicolon causes the next character to be printed in the space immediately following the last character printed.

There is one other print variation that can be made using punctuation. Change the semicolon at the end of statement 20 to a comma (,). This time when the program is executed, the message is printed in columns across the screen.

Any one line on the screen has four PRINT ZONES, each ten spaces wide. The comma at the end of the statement shifts the print position for the start of the next print statement to the next print zone. Of course, if the statement to be printed is longer than 10 characters, then there will not be four columns of print across the screen. For example, "HI JO" would be repeated in four columns, but " HELLO JOSEPHINE" would be repeated in only two. (Remember blanks count as characters).

Questions & Answers

Match the following printed effect with the punctuation that produces it:

_____ (a) Print next to last character printed.	(1) Comma	
	(2) Semicolon	
_____ (b) Print on next line.	(3) <RETURN>	
_____ (c) Print in next zone.		

Answer:

(a) 2
(b) 3
(c) 1

It is always a wise precaution to add an empty PRINT statement after a loop that has a semicolon or a comma at the end of a PRINT statement. This will ensure that the first character printed in the next PRINT statement starts on a new line. Try this program:

```
10 FOR C = 1 TO 20
20 PRINT "* ";
30 NEXT C
40 PRINT " IN MY EYES"
50 END
```

Questions & Answers

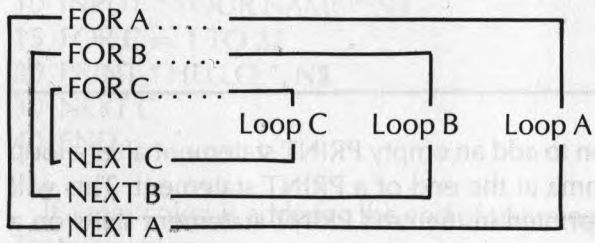
Can you add a statement 35 to make "IN MY EYES " print under the row of stars?

Answer: 35 PRINT

We can use more than one FOR-NEXT loop at the same time. When this happens they are called nested loops. If we want to print a screen full of stars, we can use a loop from 1 to 1040 to put a star in every space. We could also use loops to print 26 rows of 40 stars like this:

```
10 FOR C = 1 TO 26
20 FOR K = 1 TO 40
30 PRINT "*";
40 NEXT K
50 PRINT
60 NEXT C
```

Each loop must have its own unique variable as a counter and each loop must be truly nested inside another. Like this:



Questions & Answers

What is wrong with this program?

```
10 FOR M = 1 TO 12
20 FOR D = 1 TO 7
30 PRINT "GOOD MORNING WORLD"
40 NEXT M
50 NEXT D
```

Answer: The loops are not nested. Either statements 10 and 20 or statements 40 and 50 need to be switched.

FOR-NEXT loops are very useful and powerful programming strategies. The MATH FACTS program we have been working on could be improved by using FOR-NEXT loops.

As the program is currently written, students have to respond after each problem to get another one, which may break their concentration. We might speed up the learning process if the computer went through a set of ten problems before asking if the student wants to continue. To do this, add a FOR-NEXT loop around each set of statements that generate the problems. Here is the loop for the part of the program that presents addition problems:

```
100 REM **** ADDITION FACTS ****
105 PRINT"□"
107 FOR S = 1 TO 10
110 A = INT(RND(1) *10)
120 B = INT(RND(1) *10)
130 PRINT A;" + ";B;" = ";
140 INPUT C
150 IF C <> A+B THEN 130
160 PRINT "RIGHT "
180 NEXT S
190 END
```

Questions & Answers

Can you add FOR-NEXT loops to the sections for the subtraction, multiplication and division problems?

Test your program by running it and working one set of each kind of problem.

Answer: I hope you did not have any bugs. Here is a complete listing of the program:

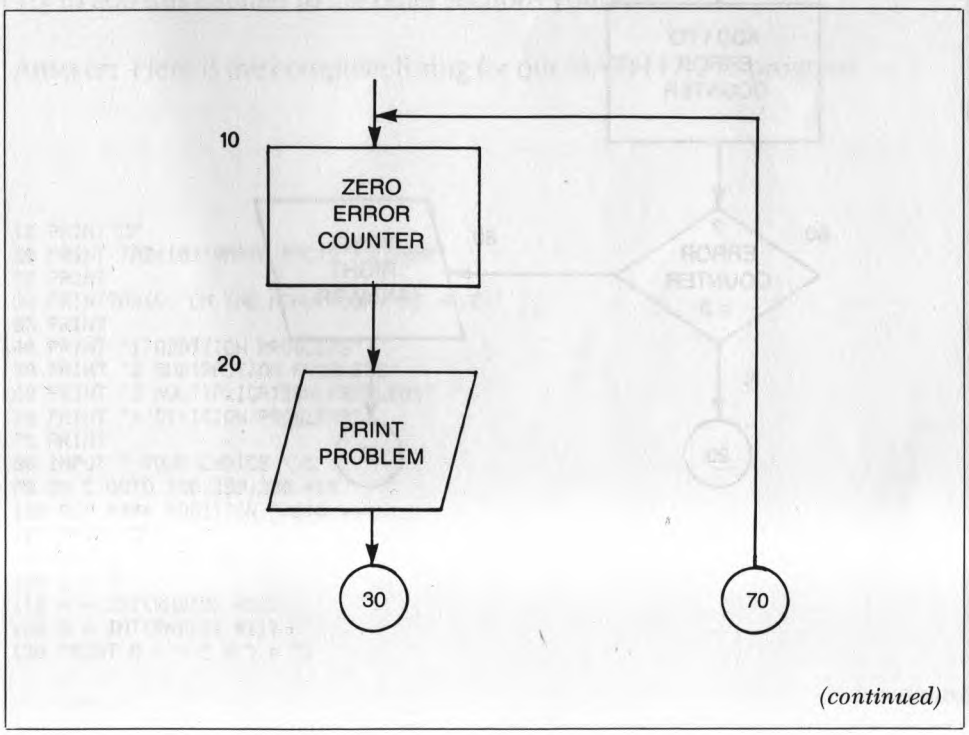
```
10 PRINT"□"
20 PRINT TAB(10)"■MATH FACTS PROGRAM"
25 PRINT
30 PRINTTAB(5)"ON THE MENU TODAY WE HAVE:"
35 PRINT
40 PRINT "1 ADDITION PROBLEMS"
50 PRINT "2 SUBTRACTION PROBLEMS"
60 PRINT "3 MULTIPLICATION PROBLEMS"
70 PRINT "4 DIVISION PROBLEMS"
75 PRINT
80 INPUT " YOUR CHOICE ";C
90 ON C GOTO 100,200,300,400
100 REM **** ADDITION FACTS ****
105 PRINT"□"
107 FOR S = 1 TO 10
110 A = INT(RND(1) *10)
120 B = INT(RND(1) *10)
130 PRINT A;" + ";B;" = ";
140 INPUT C
150 IF C <> A+B THEN 130
160 PRINT "RIGHT "
180 NEXT S
190 END
200 REM **** SUBTRACTION FACTS ****
205 PRINT"□"
207 FOR S = 1 TO 10
210 A = INT(RND(1) *10)
220 B = INT(RND(1) *10)
225 IF A<B THEN 210
230 PRINT A;" - ";B;" = ";
240 INPUT C
250 IF C <> A-B THEN 230
260 PRINT "RIGHT "
280 NEXT S
290 END
300 REM **** MULTIPLICATION FACTS ****
305 PRINT"□"
307 FOR S = 1 TO 10
310 A = INT(RND(1) *10)
320 B = INT(RND(1) *10)
330 PRINT A;" * ";B;" = ";
340 INPUT C
350 IF C <> A*B THEN 330
360 PRINT "RIGHT "
380 NEXT S
390 END
400 REM **** DIVISION FACTS ****
405 PRINT"□"
407 FOR S = 1 TO 10
410 A = INT(RND(1) *10)
420 B = INT(RND(1) *10)
```

(continued)

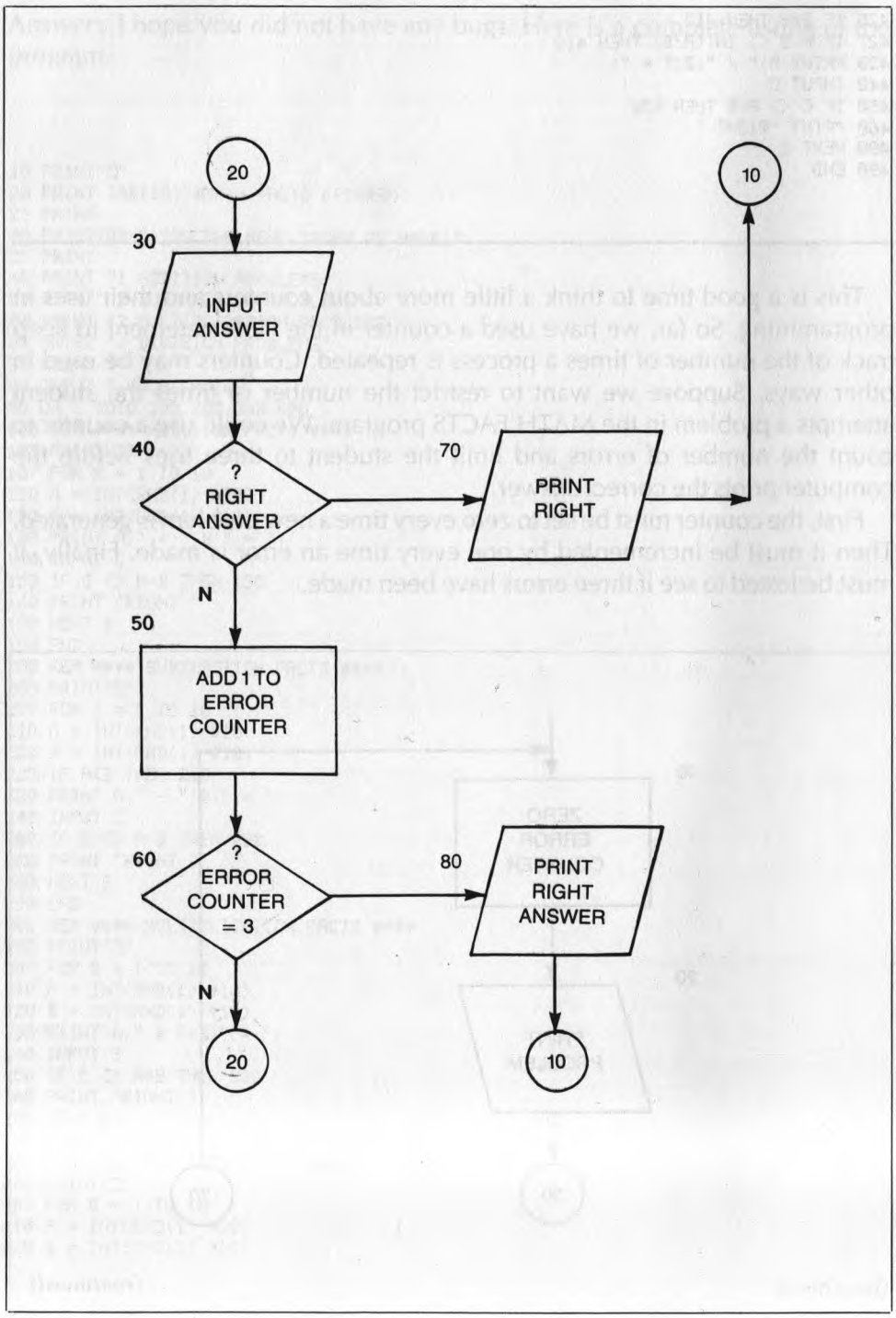
```
425 IF B=0 THEN 410
427 IF A/B <> INT(A/B) THEN 410
430 PRINT A;" / ";B;" = ";
440 INPUT C
450 IF C <> A/B THEN 430
460 PRINT "RIGHT "
480 NEXT S
490 END
```

This is a good time to think a little more about counters and their uses in programming. So far, we have used a counter in the FOR statement to keep track of the number of times a process is repeated. Counters may be used in other ways. Suppose we want to restrict the number of times the student attempts a problem in the MATH FACTS program. We could use a counter to count the number of errors and limit the student to three tries before the computer prints the correct answer.

First, the counter must be set to zero every time a new problem is generated. Then it must be incremented by one every time an error is made. Finally, it must be tested to see if three errors have been made.



(continued)


20
30
INPUT
ANSWER
40
?
RIGHT
ANSWER
70
PRINT
RIGHT
10
N
50
ADD 1 TO
ERROR
COUNTER
60
?
ERROR
COUNTER
= 3
80
PRINT
RIGHT
ANSWER
N
20
10

Here is the addition section of the program with the counter:

```
100 REM **** ADDITION FACTS ****
105 PRINT"□"
107 FOR S = 1 TO 10
109 E = 0
110 A = INT(RND(0) *11)
120 B = INT(RND(0) *11)
130 PRINT A " + " B " = ";
140 INPUT C
150 IF C =  A+B THEN 160
152 E = E +1
154 IF E <3 THEN 130
156 PRINT " THE ANSWER IS " A " + " B " = " A + B
157 GOTO 180
160 PRINT "RIGHT "
180 NEXT S
190 END
```

Questions & Answers

Try to add this counter to the other sections yourself.

Answer: Here is the complete listing for our MATH FACTS program.

```
10 PRINT"□"
20 PRINT TAB(10)"■MATH FACTS PROGRAM"
25 PRINT
30 PRINTTAB(5)"ON THE MENU TODAY WE HAVE:"
35 PRINT
40 PRINT "1 ADDITION PROBLEMS"
50 PRINT "2 SUBTRACTION PROBLEMS"
60 PRINT "3 MULTIPLICATION PROBLEMS"
70 PRINT "4 DIVISION PROBLEMS"
75 PRINT
80 INPUT " YOUR CHOICE ";C
90 ON C GOTO 100,200,300,400
100 REM **** ADDITION FACTS ****
105 PRINT"□"
107 FOR S = 1 TO 10
109 E = 0
110 A = INT(RND(0) *11)
120 B = INT(RND(0) *11)
130 PRINT A " + " B " = ";
```

(continued)

```
140 INPUT C
150 IF C =  A+B THEN 160
152 E = E +1
154 IF E <3 THEN 130
156 PRINT " THE ANSWER IS " A " + " B " = " A + B
157 GOTO 180
160 PRINT "RIGHT "
180 NEXT S
190 END
200 REM **** SUBTRACTION FACTS ****
205 PRINT"□"
207 FOR S = 1 TO 10
209 E = 0
210 A = INT(RND(0) *11)
220 B = INT(RND(0) *11)
225 IF A<B THEN 210
230 PRINT A " - " B " = ";
240 INPUT C
250 IF C =  A-B THEN 260
252 E = E +1
254 IF E <3 THEN 230
256 PRINT " THE ANSWER IS " A " - " B " = " A - B
257 GOTO 280
260 PRINT "RIGHT "
280 NEXT S
290 END
300 REM **** MULTIPLICATION FACTS ****
305 PRINT"□"
307 FOR S = 1 TO 10
309 E = 0
310 A = INT(RND(0) *11)
320 B = INT(RND(0) *11)
330 PRINT A " * " B " = ";
340 INPUT C
350 IF C =  A*B THEN 360
352 E = E +1
354 IF E <3 THEN 330
356 PRINT " THE ANSWER IS " A " * " B " = " A * B
357 GOTO 380
360 PRINT "RIGHT "
380 NEXT S
390 END
400 REM **** DIVISION FACTS ****
405 PRINT"□"
407 FOR S = 1 TO 10
409 E = 0
410 A = INT(RND(0) *11)
420 B = INT(RND(0) *11)
425 IF B=0 THEN 410
427 IF A/B <> INT(A/B) THEN 410
430 PRINT A " / " B " = ";
440 INPUT C
450 IF C =  A/B THEN 460
```

(continued)

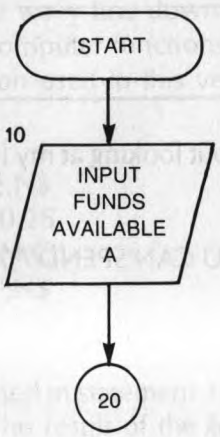
```
452 E = E +1
454 IF E <3 THEN 430
456 PRINT " THE ANSWER IS " A " / " B " = " A / B
457 GOTO 480
460 PRINT "RIGHT "
480 NEXT S
490 END
```

DO-WHILE Loops

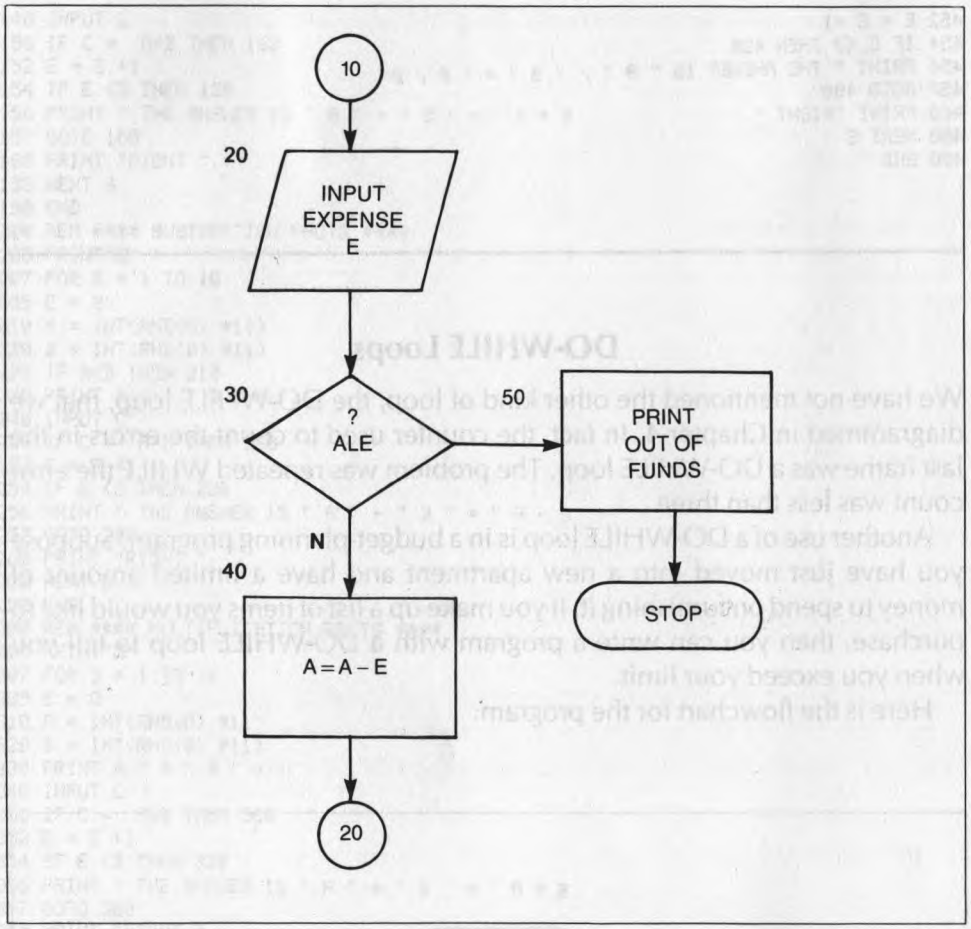
We have not mentioned the other kind of loop, the DO-WHILE loop, that we diagrammed in Chapter 4. In fact, the counter used to count the errors in the last frame was a DO-WHILE loop. The problem was repeated WHILE the error count was less than three.

Another use of a DO-WHILE loop is in a budget-planning program. Suppose you have just moved into a new apartment and have a limited amount of money to spend on furnishing it. If you make up a list of items you would like to purchase, then you can write a program with a DO-WHILE loop to tell you when you exceed your limit.

Here is the flowchart for the program:



(continued)



Questions & Answers

Can you write a program without looking at my listing?

Answer:

```
10 INPUT " AMOUNT YOU CAN SPEND ";A
20 INPUT " EXPENSE ";E
30 IF A <E THEN 60
40 A = A – E
50 GOTO 20
60 PRINT "OUT OF FUNDS "
80 END
```

BASIC on the Commodore 64 does not use any special statements to program DO-WHILE loops. This is not the case with all microcomputers, and it is something you should check if you are adapting a program written for another computer. We will find DO-WHILE loops more useful in Chapter 8 when we discuss Datafiles.

Questions & Answers

Are these statements true or false?

_____ (a) A FOR-NEXT loop uses a counter.
_____ (b) The DO-WHILE loop repeats at least once.
_____ (c) The FOR-NEXT loop repeats at least once.
_____ (d) Every FOR must have a NEXT.

Answer:
(a) True
(b) False, the DO-WHILE loop need not repeat at all.
(c) True
(d) True

The counter used in the FOR statement may be used within the loop. We saw this in the countdown program when the counter C was included in a PRINT statement to print the countdown from ten to one. The counter can also be used in calculations within the loop. The next program prints the output from the name program in a wavy line down the screen. The wave effect is made by using one of the computer functions already programmed into the Commodore 64. The function used in this version is the trigonometric sine function, SIN.

```
10 INPUT "YOUR NAME ";.N$
20 FOR C = 0 TO 40 STEP 0.25
30 T = INT( 15 + 12.5 * SIN(C))
40 PRINT TAB(T) "HELLO ";N$
50 NEXT C
```

The sine function SIN is used in statement 30 to calculate T, the number for the TAB in statement 40. The result of the SIN function will be a decimal

number, so the function INT truncates the decimal output from SIN to an integer.

There are two other trigonometric functions: cosine, COS, and tangent, TAN. You might like to substitute these for the SIN function in the program.

Look at the STEP part of the FOR statement in this program. This time the STEP value is positive, but it is not equal to the default value of 1.

Questions & Answers

What is the STEP increment in this program?

_____ (a) 1
_____ (b) 40
_____ (c) 0.25
_____ (d) 0

Answer: (c) Statement 20 includes STEP 0.25, which increments the counter by this amount each time it is executed.

We will explore some other preprogrammed functions that can be used with string variables. Here are four of them:

LEFT$(A$,x) returns x number of characters of A$, starting at the left.

For example:
PRINT LEFT$("ANN",1) will print 'A'
PRINT LEFT $("ANN",2) will print 'AN'

RIGHT$(A$,x) returns x number of characters of A$, starting at the right.

For example:
PRINT RIGHT$("HARRIS",1) will print 'S'
PRINT RIGHT$("HARRIS",2) will print 'IS'

MID$(A$,y,x) returns x number of characters of A$, starting at the y character in the string.

For example:
PRINT MID$("HARRIS",3,2) will print 'RR'.
PRINT MID$("HARRIS",2,3) will print 'ARR'

MID$(A$,y) returns the characters of A$, starting at the y character.

For example:
PRINT MID$("HARRIS",2) will print 'ARRIS'
PRINT MID$("HARRIS",4) will print 'RRIS'

Questions & Answers

Try this program. It includes the LEFT$ function in a FOR-NEXT loop.

```
10 A$ = "HAPPY BIRTHDAY"
20 FOR C = 1 TO 14
30 PRINT LEFT$(A$,C)
40 NEXT C
```

(a) What happens if you substitute RIGHT$(A$,C) for LEFT$(A$,C)?
(b) What happens when you use MID$(A$,C,1) instead of LEFT$(A$,C)?

Answer:

(a)
```
Y
AY
DAY
HDAY
THDAY
RTHDAY
IRTHDAY
BIRTHDAY
 BIRTHDAY
Y BIRTHDAY
PY BIRTHDAY
PPY BIRTHDAY
APPY BIRTHDAY
HAPPY BIRTHDAY
```

(b)
```
H
A
P
P
Y

B
I
R
T
H
D
A
Y
```

Many computer users are slow typists, so this next sequence of statements allows us to enter just the first letter of a response, instead of the whole word.

```
110 PRINT " DO YOU WANT TO PLAY AGAIN ";
120 INPUT A$
130 IF LEFT$(A$,1) = "Y" THEN 10
140 END
```

The program will branch to 10 if I type 'Y' or 'YES.' This sequence is very useful, but is has its problems.

Questions & Answers

Which of these responses will send me to play again?

_____ (a) No
_____ (b) Yuck - no thank you!
_____ (c) Y
_____ (d) Naturally
_____ (e) You bet your sweet bippy!

Answer: (b), (c), and (e). Statement 130 tests the first left hand character of A$.

DATA Statements

We are now moving to a new topic that is introduced here because it is frequently used in conjunction with a loop, and it involves yet another kind of counter.

Sometimes a program requires the same information every time it is run. For most of us, it is tedious and unnecessary to type the same things in over and over again. We need a way to include this information in the program and let the computer retrieve it when it is needed.

We can include information in our program by using a DATA statement. It looks like this:

DATA <item>,<item>,....

For instance, we might want our output labelled with the days of the week or the months of the year. In this case, the names of the days of the week or the months of the year are included in the program in a DATA statement:

```
50 DATA SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
   FRIDAY, SATURDAY
```

The list of data items following the DATA statement can continue for more than one line, but you may lose items if you continue over to a third line. Not all the data items need to be in one statement, so if you have a long list you can split it up into several DATA statements. We will see this happen when we start to write music programs.

READ Statements

We can input the information from data statements for use in the program by using a READ statement. It looks like this:

```
READ <variable>,<variable>,.....
```

The READ statement is always followed by the variable name for the information to be read. So, for the days of the week example we would use:

```
20 READ D$
```

D$ is the variable name for the days of the week. The whole program to read in the days of the week and print them on the screen looks like this:

```
10 FOR C = 1 TO 7
20 READ D$
30 PRINT D$
40 NEXT C
50 DATA MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
   SATURDAY, SUNDAY
```

There may be more than one variable name following the READ statement.

Questions & Answers

Can you work out what will be printed the second time through this loop?

```
 60 FOR C = 1 TO 3
 70 READ N$,A
 80 PRINT N$;A
100 DATA O. OYL, 21, M.MOUSE, 6, W.CHURCHILL,67
```

Answer: M. MOUSE 6

The DATA statements may be anywhere in the program, but it is a good practice to put them at the end of the program. Then they will not confuse you when you are trying to follow the flow of the logic in the program.

It is important that the information in the DATA statements is in the order in which you want to READ it into the program. The Commodore 64 uses a special kind of counter, or pointer, to keep track of the items in the DATA statements. When the program is started, the pointer points to the first item in the first DATA statement in the program. After each READ statement is executed, the pointer shifts to the next item in the list. This is an important, but difficult concept to understand. It may help to think of the items in a DATA statement as a stack of records on the stereo. Each one is played, or READ, in the order it is stacked.

Look carefully at this program:

```
10 READ D$
20 INPUT HR
30 READ R
40 S = HR * R
50 READ N$,A
70 END
80 DATA ENGLISH, 15.00, ADAMS, 23
```

When this program is started, the pointer is pointing at ENGLISH, the first item in the data list.

Questions & Answers

Which of the following statements is true when statement 30 is executed?

_____ (a) The pointer is pointing at the second item in the DATA list.
_____ (b) The pointer is pointing at ADAMS.
_____ (c) The pointer is pointing at ENGLISH.
_____ (d) The pointer is pointing at 15.00.

Answer: (a) and (d) are true. The pointer has moved to the second item in the data list, 15.00.

RESTORE Statements

There is another BASIC statement associated with the READ and DATA statements that will reset the pointer to the first item in the DATA statement. The statement is called RESTORE, and it is used by itself this way:

```
80 RESTORE
```

The Davis Demon Dominoes club is playing a round robin tournament. The secretary wants a chart drawn up with the players' names printed across the top of the page and down the left-hand margin.

This should solve the problem:

```
 10 FOR C = 1 TO 4
 20 READ P$
 30 PRINT P$; " ";
 40 NEXT C
 50 PRINT
 60 RESTORE
 70 FOR C = 1 TO 4
 80 READ P$
 90 PRINT P$
100 NEXT C
110 DATA KING, STAR, TOPP, KLUTZ
```

Run this program and see if it fulfills the requirements.

The first loop, lines 10 through 40, will read all the names in the data statement. But we want to read the lines again for the second loop, lines 70 through 100. So line 60, RESTORE, will put the pointer back to the first item in the data list, KING.

Questions & Answers

Fill in the blanks:

The _____ statement inputs the information listed in a _____ statement. The _____ statement resets the pointer to the _____ item in the list.

Answer: READ, DATA, RESTORE, first

Now that we have covered FOR-NEXT loops, READ statements and DATA statements, we can discuss variables in a little more depth.

We have used variables frequently in our programs. As you remember, the definition of a variable is a piece of information that may change each time the program is run.

Examples of variables we have used include:

Names
Ages
Weights
Grades

Questions & Answers

What is the opposite of a variable?

Answer: A constant.

Although we have made use of many variables, we have not considered how to use variables in situations such as these:

- Storing the names of all the students in Mrs. Stockard's class.
- Keeping all twelve months in our program without using twelve variables.

Arrays

These problems can be solved if in each case we treat a variable as a list of values, not just one value. In computer programming, a list is called a subscripted variable or an ARRAY. Arrays look like this:

```
N(4)
```

In this example, N is the variable name and the number in parentheses is the subscript. In this case, the subscript refers to the fourth position in the list.

Mrs. Stockard has six children in her top reading group:

Bret
Buffie
Lisa
Mark
Max
Wendy

To make this list into an array, we choose a variable name, T, for top. The list contains alphabetic information, so it is a string variable, T$. Finally, each name has a position in the array, indicated by the subscript. The array will look like this:

```
T$(0) = "Bret"
T$(1) = "Buffie"
T$(2) = "Lisa"
T$(3) = "Mark"
T$(4) = "Max"
T$(5) = "Wendy"
```

If we wanted to print Bret's name we could use this statement:

```
120 PRINT T$(0)
```

Questions & Answers

(a) How could we print Mark's name?
(b) Whose name will be printed if we use T$(5)?

Answer:

(a) 120 PRINT T$(3)
(b) Wendy's

Notice that the first name in the array is stored at T$(0), not T$(1). An array always has a zero subscript, but many people are not comfortable with this. In everyday life, most of the numbered lists we see start with the number one. There are times when we will find subscript zero very useful, but for most of our arrays we will use subscript one as the first subscript in an array.

Here is a program that will read and print five of Sam's spaghetti sauces:

```
10 FOR C = 1 TO 5
20 READ S$(C)
30 PRINT S$(C),
40 NEXT C
50 DATA DE LUXE, SUPER, SPICY, REGULAR, ECONOMY
```

Questions & Answers

Using this program can you fill in the items in the S$ array?

S$(1) _____
S$(2) _____
S$(3) _____
S$(4) _____
S$(5) _____

Answer:

S$(1) DE LUXE
S$(2) SUPER
S$(3) SPICY
S$(4) REGULAR
S$(5) ECONOMY

When the Commodore 64 encounters a subscripted variable in a program for the first time, it sets aside eleven spaces for all the items in that array. This will not be enough spaces to store the twelve months in the years. If we want

---

the computer to set aside space for arrays with more than eleven items, (ten if we are not using subscript zero), then we use a dimension statement like this:

DIM <variable name> (<maximum subscript # >)

For example, we might use this statement to set aside room for the months of the year:

```
10 DIM M$(12)
```

We could use it in a program like this:

```
10 DIM M$(12)
20 FOR C= 1 TO 12
30 READ M$(C)
40 PRINT M$(C)
50 NEXT C
60 DATA JANUARY,FEBRUARY,MARCH,APRIL,MAY
70 DATA JUNE,JULY,AUGUST,SEPTEMBER,OCTOBER
80 DATA NOVEMBER,DECEMBER
```

Questions & Answers

True or false?

_____ (a) An array must have a dimension statement.
_____ (b) An array with more than 11 elements have a dimension statement.
_____ (c) This program needs a DIM statement.

Answer:
(b) and (c) are true.
(a) is false because only arrays larger than 11 need to be dimensioned.

Dimension statements are usually placed at the beginning of the program. The dimension statement may contain information for more than one array.

Questions & Answers

10 DIM L(100), A$(50)

(a) What kind of information is contained in the array L?
(b) How many elements does the L$ array contain?
(c) How many elements does the A$ array contain?

Answer:
(a) Numeric.
(b) None - there isn't an L$ array in the statement.
(c) 51, including A$(0).

WARNING

Do not dimension an array more than once. If you do, your program will stop and you will see an error message.

A list is a one dimensional array and the variable name is followed by one subscript. If we wanted to store a TABLE of information, we need another subscript. We can think of the first subscript as the number of rows in the table, and the second subscript as the number of columns. Here is a table, A(4,3), with all the elements labelled to illustrate the rows and columns and their relationship to the two subscripts.

	Column 1	Column 2	Column 3
Row 1	A(1,1)	A(1,2)	A(1,3)
Row 2	A(2,1)	A(2,2)	A(2,3)
Row 3	A(3,1)	A(3,2)	A(3,3)
Row 4	A(4,2)	A(4,2)	A(4,3)

Questions & Answers

How many rows are there in these arrays?

_____ (a) F$(6,7)
_____ (b) Y(10,80)
_____ (c) N$(2,10)

Answer: (a) 6, (b) 10, (c) 2.

How many columns are there in these arrays?

_____ (a) F$(4,3)
_____ (b) Y(10,80)
_____ (c) N$(2,10)

Answer: (a) 3, (b) 80, (c) 10.

Let's use the row and column idea with some real data. The Brown family wants to keep a close watch on their household expenses this year. For comparison, they have a table of utility, grocery and gas expenses for last year broken down by month. Here is the table:

Month	Utility	Grocery	Gas
Jan	110.00	245.00	40.00
Feb	120.00	240.00	35.00
Mar	115.00	250.00	42.00
Apr	100.00	263.00	20.00
May	75.00	235.00	29.00
Jun	60.00	248.00	60.00
Jul	80.00	270.00	32.00
Aug	95.00	244.00	95.00
Sept	70.00	252.00	43.00
Oct	87.00	238.00	48.00
Nov	95.00	290.00	56.00
Dec	100.00	310.00	61.00

The table has twelve rows and three columns, so we will call it T(12,3). It has a subscript larger than ten, so it will need a dimension statement.

```
10 DIM T(12,3)
```

To make the information in the table available to the program, we must first read into the array, T. This requires nested loops, as shown in these statements.

```
20 FOR R = 1 TO 12
30     FOR C = 1 TO 3
40     READ T(R,C)
50     NEXT C
60 NEXT R
```

Questions & Answers

(a) What is the counter R counting?
(b) What is the counter C counting?

Answer: (a) rows. (b) columns.

We will put the data statements at the end of the program, starting at line 200. Here they are:

```
200 DATA 110.00,245.00,40.00,120.00,240.00,35.00
210 DATA 115.00,250.00,35.00,100.00,263.00,20.00
220 DATA 75.00,235.00,29.00,60.00,248.00,60.00
230 DATA 80.00,270.00,32.00,95.00,244.00,95.00
240 DATA 70.00,252.00,43.00,87.00,238.00,48.00
250 DATA 95.00,290.00,56.00,100.00,310.00,61.00
```

As we are reading the data row by row, we must be sure that it is in the correct order in the data statements. Notice that each data statement contains six entries, one for each for utilities, groceries and gas for two months.

To check that we have entered our table of data correctly, we will add another nested loop to print out the table.

```
 70 FOR R = 1 TO 12
 80     FOR C = 1 TO 3
 90     PRINT T(R,C)
100     NEXT C
110 PRINT
120 NEXT R
130 END
```

Now run the program and see if the output matches the table.

Here is another small table. It contains the player's number, GPA, weight, height in inches, and speed for four football players.

Questions & Answers

Write a program to read this information into memory.

Player #	GPA	WT	HT	Speed
23	3.1	205	73	4.7
76	3.5	215	75	4.9
44	2.2	230	72	5.1
17	2.9	195	70	4.6

Answer: This is my version of the program; yours should be similar. I have put the information for each player in a separate data statement.

```
10 FOR R = 1 TO 4
20     FOR C = 1 TO 5
30     READ F(R,C)
40     NEXT C
50 NEXT R
60 DATA 23,3.1,205,73,4.7
70 DATA 76,3.5,215,73,4.7
80 DATA 44,2.2,215,75,4.9
90 DATA 17,2.9,195,70,4.6
```

Now that we know how to create one and two dimensional arrays, let's think about some of the more frequent ways in which they are used.

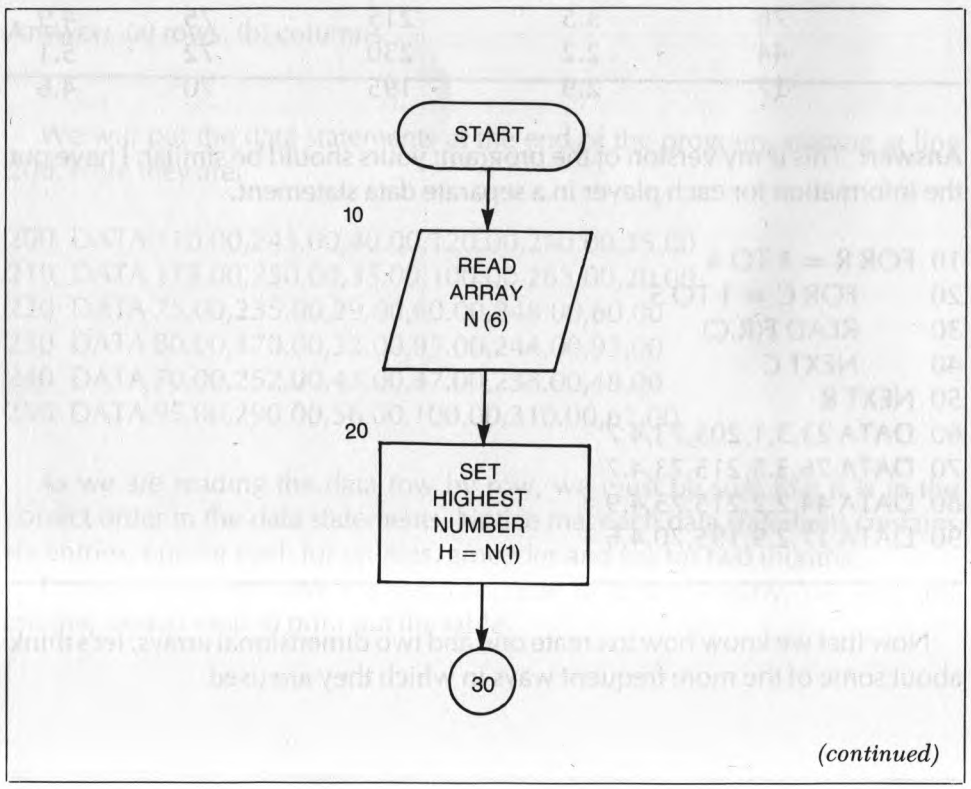
One operation that occurs quite often is to find the largest or smallest number in an array. To do this we will use an array of eleven numbers. Here it is:

27,43,78,69,33,18,91,54,82,5

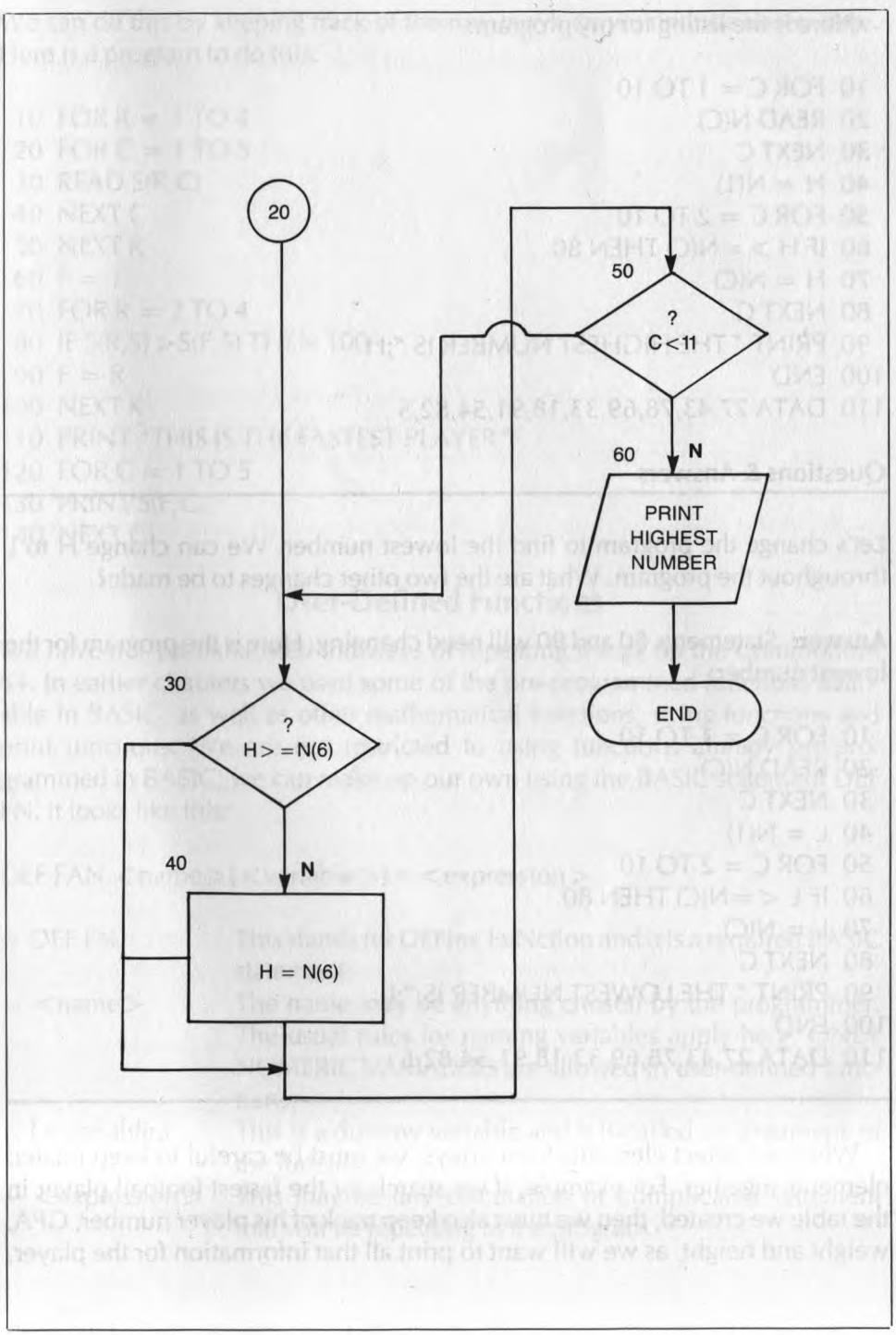
This is the strategy we will use to find the biggest number in the array:

- Read the number into the array, N.
- Make the first number in the array, N(1), the highest number and store it in a variable called H.
- Use a FOR-NEXT loop to compare H with each number in the array.
- If a number in the array is larger than the number in H, put it in H.
- When the loop is completed, H will contain the largest number in the list.

Look at this flowchart, and then write the program for yourself before you check mine.



(continued)


20
50
?
C<11
60
N
PRINT
HIGHEST
NUMBER
END
30
?
H>=N(6)
40
N
H = N(6)

Here is the listing for my program:

```
 10 FOR C = 1 TO 10
 20 READ N(C)
 30 NEXT C
 40 H = N(1)
 50 FOR C = 2 TO 10
 60 IF H >=N(C) THEN 80
 70 H = N(C)
 80 NEXT C
 90 PRINT " THE HIGHEST NUMBER IS ";H
100 END
110 DATA 27,43,78,69,33,18,91,54,82,5
```

Questions & Answers

Let's change the program to find the lowest number. We can change H to L throughout the program. What are the two other changes to be made?

Answer: Statements 60 and 90 will need changing. Here is the program for the lowest number:

```
 10 FOR C = 1 TO 10
 20 READ N(C)
 30 NEXT C
 40 L = N(1)
 50 FOR C = 2 TO 10
 60 IF L <=N(C) THEN 80
 70 L = N(C)
 80 NEXT C
 90 PRINT " THE LOWEST NUMBER IS ";L
100 END
110 DATA 27,43,78,69,33,18,91,54,82,5
```

When we select elements from arrays, we must be careful to keep related elements together. For example, if we search for the fastest football player in the table we created, then we must also keep track of his player number, GPA, weight and height, as we will want to print all that information for the player.

We can do this by keeping track of the row in which we find the fastest time. Here is a program to do this:

```
 10 FOR R = 1 TO 4
 20 FOR C = 1 TO 5
 30 READ S(R,C)
 40 NEXT C
 50 NEXT R
 60 F = 1
 70 FOR R = 2 TO 4
 80 IF S(R,5) > S(F,5) THEN 100
 90 F = R
100 NEXT R
110 PRINT "THIS IS THE FASTEST PLAYER"
120 FOR C = 1 TO 5
130 PRINT S(F,C);
140 NEXT C
```

User-Defined Functions

We have not yet exhausted the ways of repeating things on the Commodore 64. In earlier chapters we used some of the pre-programmed functions available in BASIC, as well as other mathematical functions, string functions and print functions. We are not restricted to using functions already pre-programmed in BASIC; we can make up our own using the BASIC statement DEF FN. It looks like this:

DEF FAN <name>(<variable>)=<expression>

– DEF FN	This stands for DEFine FuNction and it is a required BASIC statement.
– <name>	The name may be anything chosen by the programmer. The usual rules for naming variables apply here. ONLY NUMERIC VARIABLES are allowed in user-defined functions.
– (<variable.)	This is a dummy variable and it is called an 'argument' of the function.
– <expression>	This may be any calculation or complicated statement that will be repeating in the program.

When we want to use a programmer-defined function in the program we write:

FN<name>(<variable>

- FN stands for FuNction and is required BASIC.
- <name> is the name we used in the DEF FN statement.
- (<variable>) is the variable we want to use in the calculation at this point in the program.

User-defined functions are complicated to use, but some examples will help make it clear.

Earlier, we drew a flowchart and wrote a program to convert measurements from one scale to another. One example was converting temperatures from degrees Fahrenheit to degrees Celsius. The conversion factor was:

```
C = (F-32)*5/9
```

We can make this conversion factor a user-defined function at the beginning of the program:

```
10 DEF FNC(F) = (F - 32)*5/9
```

The rest of the program will now look like this:

```
20 INPUT "TEMPERATURE IN FAHRENHEIT";F
30 C = FNC(F)
40 PRINT F,C
```

Another example is formatting decimal numbers in a financial calculation so that they are printed to the nearest cent. The calculation looks like this:

```
D = INT(A*100+0.5)/100
```

Questions & Answers

Can you write the DEF FN statement for this?

Answer: 10 DEF FND(A) = INT(A*100+0.5)/100

Now can you write the rest of the program to read the following numbers and print them out as dollars and cents?

10.999
3.471
0.76543

Answer:

```
20 FOR C = 1 TO 3
30 READ N
40 PRINT FND(N)
50 NEXT C
60 DATA 10.999,4.471,0.76543
```

The advantage of a user-defined function is that it only needs to be written out once at the beginning of a program. Then it can be called as many times as it is needed during the program, and the dummy variable, or argument, can be used for a different variable each time.

The function you write can contain only one expression. However, one user-defined function may contain another user-defined statement. For example, look at these two functions:

```
10 DEF FNA(R) = TT*R*R
20 DEF FNV(H) = FNA*H
```

Statement 10 defines a function to calculate the area of a circle with a radius R. Statement 20 defines a statement to calculate the volume of a cylinder, and it includes the function to calculate the area of a circle.

Questions & Answers

Are these statements true or false?

(a) A user-defined function is defined at the beginning of a program.
(b) User-defined functions can be more than one expression.
(c) A user-defined function may contain another user-defined function.
(d) A user-defined function may contain only numeric variables.

Answer: (a) and (c) are true.

Subroutines

There are times when we want to repeat a section of a program in more than one place. For example:

- Printing the headings at the top of each page of a report.
- Calculating depreciation on certain business expenditures.
- Printing the board in a tic-tac-toe game.

We can do this by using a SUBROUTINE. A subroutine is a small program within a program.

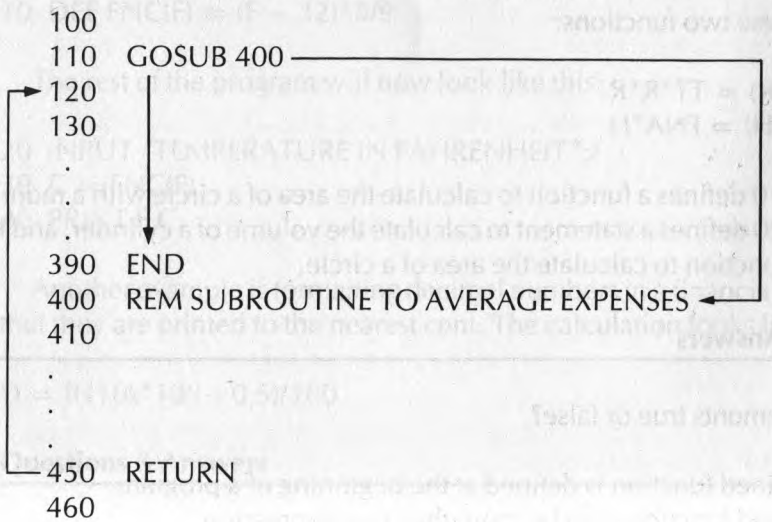
To branch to or 'call' a subroutine, use the BASIC statement:

GOSUB <statement number>

When the subroutine is completed, the final statement is:

RETURN

The control of the program then returns to the statement immediately following the GOSUB statement.



It is a good practice to make the first statement in a subroutine a remark (REM) statement. This clearly identifies the subroutine in the program listing,

and is especially helpful in developing the program. It can always be removed when the program is finished and debugged.

```
100  INPUT 'YOUR NAME'; N$
110   GOSUB 300
120   FOR C = 1 TO 1O
130   PRINT 2 * C
.
.
.
200   NEXT C
210   END
.
.
.
300  REM SUBROUTINE TO PRINT INSTRUCTIONS
310   PRINT 'WELCOME TO MULTIPLICATION MAGIC'
320
.
.
.
370   RETURN
380   REM
390
.
.
.
```

Questions & Answers

In the diagram of a program listing:

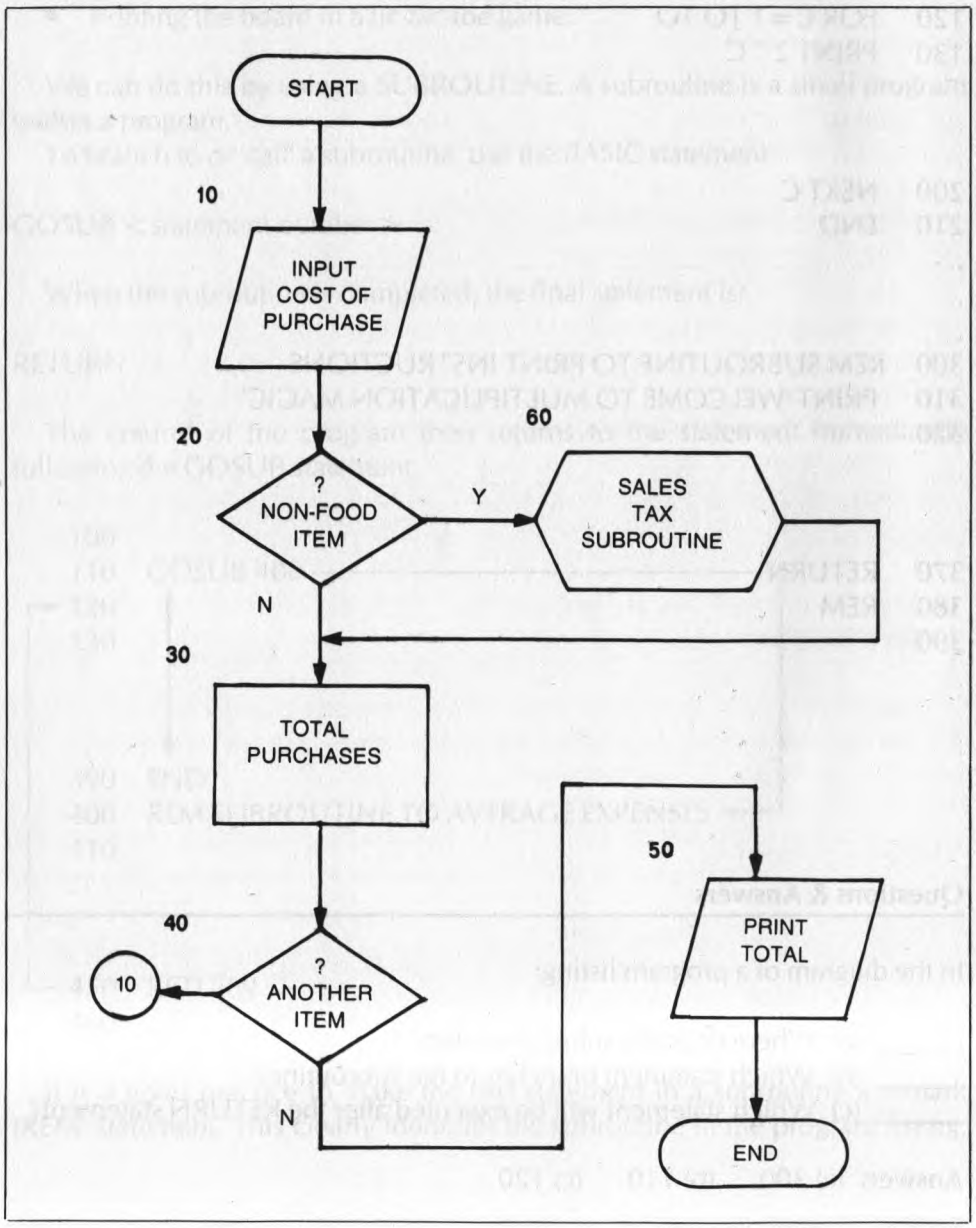
_____ (a) Where does the subroutine start?
_____ (b) Which statement branches to the subroutine?
_____ (c) Which statement will be executed after the RETURN statement?

Answer: (a) 300 (b) 110 (c) 120

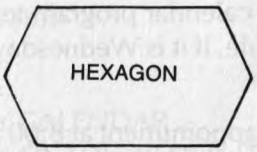
First we will practice using a subroutine in a short program.

We will write a program for the MOM AND POP CORNER STORE to calculate 6% sales tax on all non-food items. The sales tax calculation will be a subroutine.

First we draw the flowchart.



Notice that the symbol for a subroutine is a hexagon.



Here is the listing. We write the MAIN PROGRAM first and then the subroutine. There must be an END statement at the end of the main program and before the subroutine.

```
10   REM CASH REGISTER PROGRAM
20   INPUT "COST OF PURCHASE";C
30   INPUT "NON-FOOD ITEM"; A$
40   IF A$ = "NO" THEN 60
50   GOSUB 110
60   T = T + C
70   INPUT "ANOTHER ITEM";A$
80   IF A$ = "YES" THEN 20
90   PRINT "TOTAL COST IS"; T
100   END
110   REM SUBROUTINE TO CALCULATE SALES TAX
120   C = C * 1.06
130   RETURN
```

Questions & Answers

What are the final statements for:

_____ (a) The subroutine?
_____ (b) The main program?

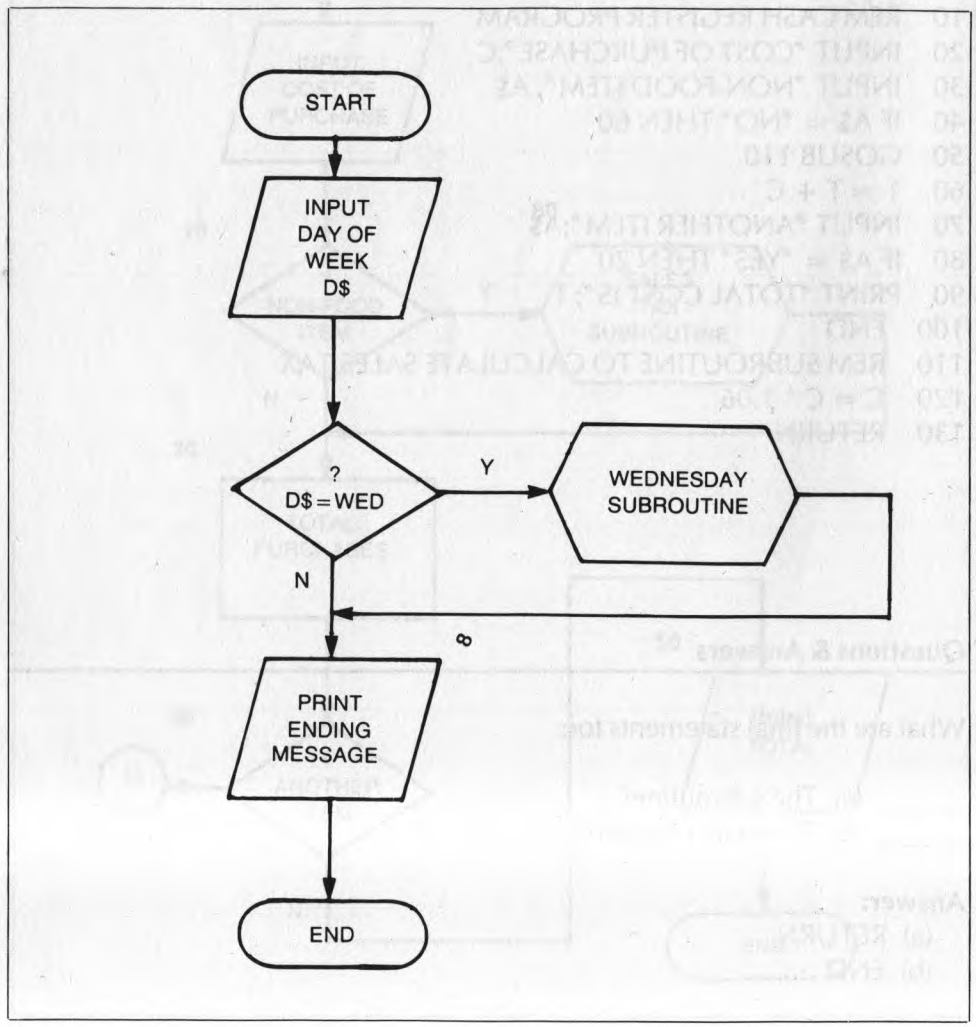
Answer:
(a) RETURN
(b) END

Here is another short example:

The Jones family is writing a calendar program to remind the members of the family of their weekly schedule. If it is Wednesday:

- Mr. Jones has a dental appointment at 8:00 a.m.
- Sally has a piano lesson at 4:00 p.m.
- Tim has soccer practice at 3:00 p.m.

We will make the appointments a subroutine:



Questions & Answers

Now can you write the program?

Answer: Here is my version:

```
10  REM JONES' FAMILY CALENDAR
20  INPUT "WHAT DAY OF THE WEEK IS IT";D$
30  IF D$ = "WEDNESDAY" THEN GOSUB 100
40  PRINT "HAVE A GOOD DAY"
50  END
100 REM WEDNESDAY SUBROUTINE
110 PRINT "MR. JONES DENTAL APPOINTMENT 8 AM"
120 PRINT "SALLY PIANO LESSON 4 PM"
130 PRINT "TIM SOCCER PRACTICE 3 PM"
140 RETURN
```

The Math Facts program could be changed so that each kind of problem, addition, subtraction, multiplication and division, is a subroutine. To do this we need to change the case statement, line 90 from ON-GOTO to ON-GOSUB like this:

```
90 ON C GOSUB 100,200,300,400
```

When the problem set chosen is completed, the program will return to statement 100. However, this is also the beginning of the addition subroutine. We will have to add some more statements to prevent this. We could put an END statement here, or we could find out if the user wants another problem set. In this case, we would ask a question and either branch to the beginning of the main menu or end the program. The new statements will look like this:

```
92 INPUT "WOULD YOU LIKE ANOTHER SET ";A$
94 IF LEFT$(A$,1) = "Y" THEN 10
96 END
```

Questions & Answers

Now check through the program carefully and make the necessary changes to convert the different problem sets to subroutines. Remember to put a RETURN statement at the end of each subroutine.

Answer: Here is my program with the changes underlined:

```
10 PRINT"□"
20 PRINT TAB(10)"■MATH FACTS PROGRAM"
25 PRINT
30 PRINTTAB(5)"ON THE MENU TODAY WE HAVE:"
35 PRINT
40 PRINT "1 ADDITION PROBLEMS"
50 PRINT "2 SUBTRACTION PROBLEMS"
60 PRINT "3 MULTIPLICATION PROBLEMS"
70 PRINT "4 DIVISION PROBLEMS"
75 PRINT
80 INPUT " YOUR CHOICE ";C
90 ON C GOSUB 100,200,300,400
92 INPUT "WOULD YOU LIKE ANOTHER SET ";A$
94 IF LEFT$(A$,1) = "Y" THEN 10
96 END
100 REM **** ADDITION FACTS ****
102 PRINT"□"
105 FOR S = 1 TO 10
107 E = 0
110 A = INT(RND(0) *11)
120 B = INT(RND(0) *11)
130 PRINT A " + " B " = ";
140 INPUT C
150 IF C =  A+B THEN 160
152 E = E +1
154 IF E <3 THEN 130
156 PRINT " THE ANSWER IS " A " + " B " = " A + B
157 GOTO 180
160 PRINT "RIGHT "
180 NEXT S
190 RETURN
200 REM **** SUBTRACTION FACTS ****
202 PRINT"□"
205 FOR S = 1 TO 10
207 E = 0
210 A = INT(RND(0) *11)
220 B = INT(RND(0) *11)
225 IF A<B THEN 210
230 PRINT A " - " B " = ";
240 INPUT C
250 IF C =  A-B THEN 260
252 E = E +1
254 IF E <3 THEN 230
256 PRINT " THE ANSWER IS " A " - " B " = " A - B
257 GOTO 280
260 PRINT "RIGHT "
280 NEXT S
290 RETURN
300 REM **** MULTIPLICATION FACTS ****
302 PRINT"□"
305 FOR S = 1 TO 10
307 E = 0
```

(continued)

```
310 A = INT(RND(0) *11)
320 B = INT(RND(0) *11)
330 PRINT A " * " B " = ";
340 INPUT C
350 IF C =  A*B THEN 360
352 E = E +1
354 IF E <3 THEN 330
356 PRINT " THE ANSWER IS " A " * " B " = " A * B
357 GOTO 380
360 PRINT "RIGHT "
380 NEXT S
390 RETURN
400 REM **** DIVISION FACTS ****
402 PRINT"□"
405 FOR S = 1 TO 10
407 E = 0
410 A = INT(RND(0) *11)
420 B = INT(RND(0) *11)
425 IF B=0 THEN 410
427 IF A/B <> INT(A/B) THEN 410
430 PRINT A " / " B " = ";
440 INPUT C
450 IF C =  A/B THEN 460
452 E = E +1
454 IF E <3 THEN 430
456 PRINT " THE ANSWER IS " A " / " B " = " A / B
457 GOTO 480
460 PRINT "RIGHT "
480 NEXT S
490 RETURN
```

Now let's look at using subroutines in a new problem. We are going to write a program to play tic-tac-toe. It will be an unusual game because the computer will not use any strategy, and will choose squares only at random. The player and the computer will choose squares by picking a number between 1 and 9. The board will look like this.

1	2	3
4	5	6
7	8	9

The usual rules for winning will apply: three in a row, horizontally, vertically or diagonally. Of course, the game may end in a draw.

Questions & Answers

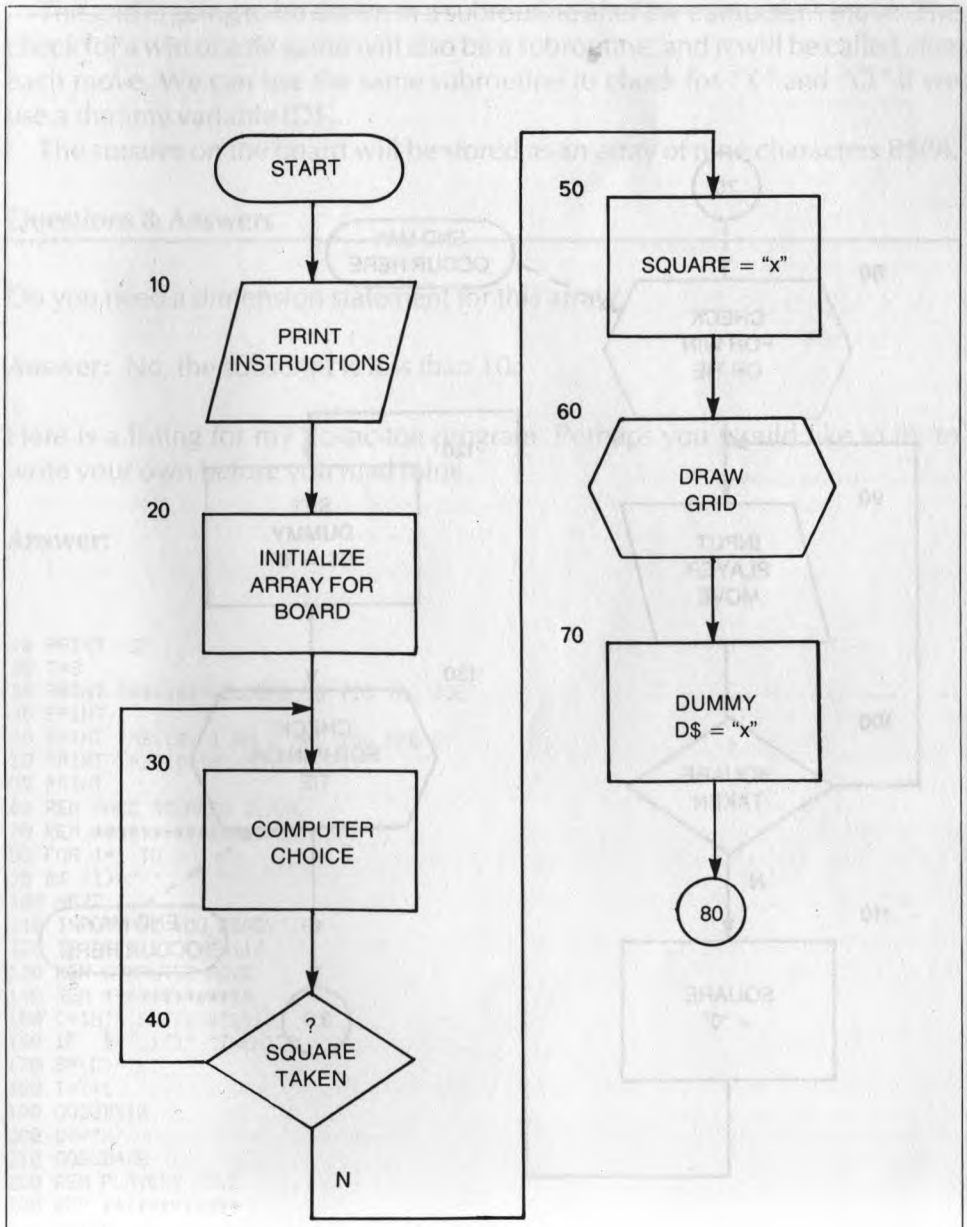
Take a minute to think of all the steps you would take if you were playing tic-tac-toe with a friend. Make a list of the steps and then compare it with mine.

Answer:

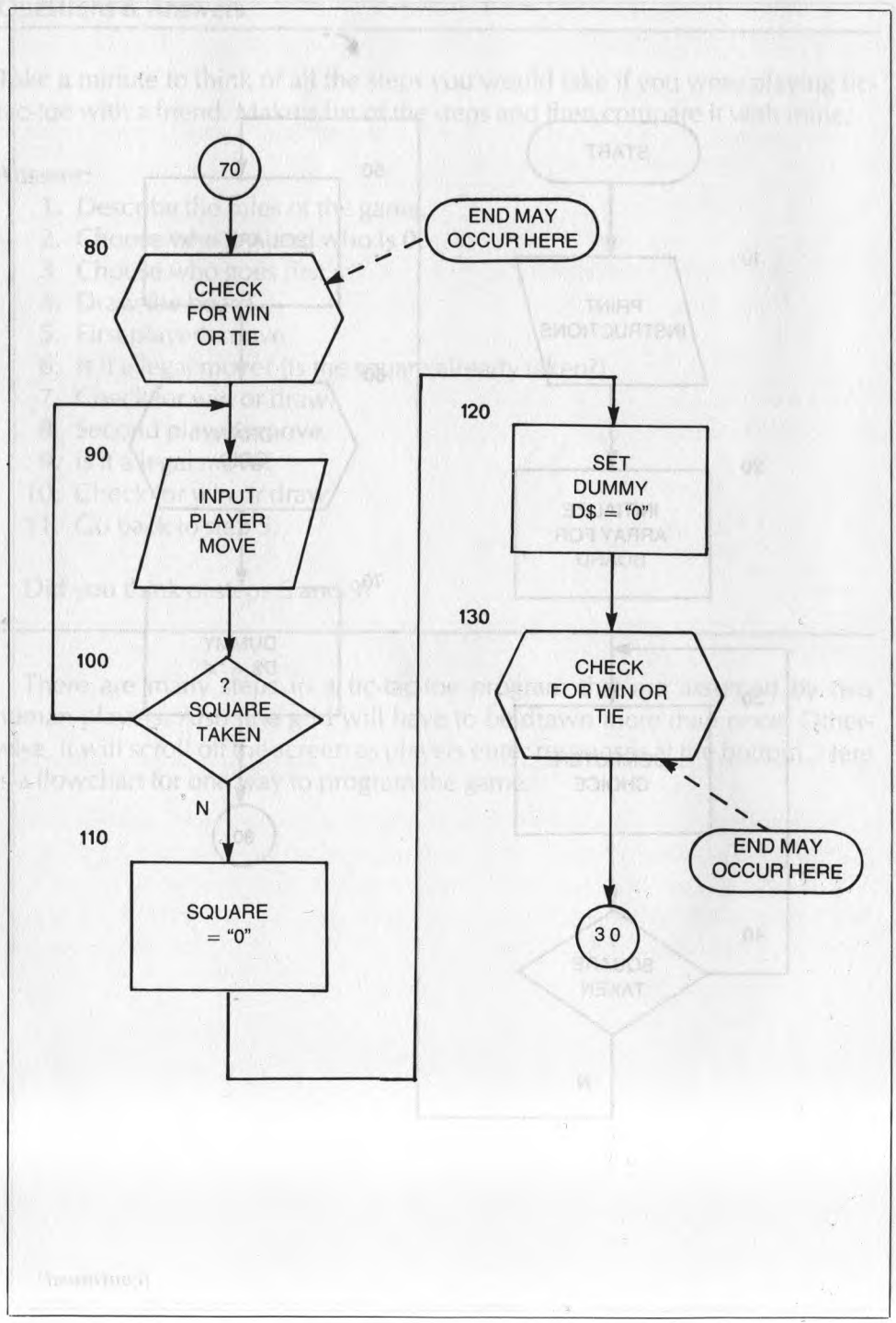
1. Describe the rules of the game.
2. Choose who is X and who is 0.
3. Choose who goes first.
4. Draw the board.
5. First player's move.
6. Is it a legal move? (Is the square already taken?)
7. Check for win or draw.
8. Second player's move.
9. Is it a legal move?
10. Check for win or draw.
11. Go back to step 5.

Did you think of steps 6 and 9?

There are many steps in a tic-tac-toe program that are assumed by two human players. Also, the grid will have to be drawn more than once. Otherwise, it will scroll off the screen as players enter responses at the bottom. Here is a flowchart for one way to program the game:


START
10
PRINT INSTRUCTIONS
20
INITIALIZE ARRAY FOR BOARD
30
COMPUTER CHOICE
40
? SQUARE TAKEN
N
50
SQUARE = "x"
60
DRAW GRID
70
DUMMY D$ = "x"
80

(continued)


70
END MAY OCCUR HERE
80
CHECK FOR WIN OR TIE
120
90
SET DUMMY D$ = "0"
INPUT PLAYER MOVE
130
100
CHECK FOR WIN OR TIE
? SQUARE TAKEN
N
110
END MAY OCCUR HERE
SQUARE = "0"
30

The grid is going to be drawn in a subroutine after the computer's move. The check for a win or a tie game will also be a subroutine, and it will be called after each move. We can use the same subroutine to check for "X" and "O" if we use a dummy variable (D$).

The squares on the board will be stored as an array of nine characters B$(9).

Questions & Answers

Do you need a dimension statement for this array?

Answer: No, the subscript is less than 10.

Here is a listing for my tic-tac-toe program. Perhaps you would like to try to write your own before you read mine.

Answer:

```
10 PRINT "□"
20 T=0
30 PRINT TAB(10)"WELCOME TO TIC TAC TOE"
35 PRINT
40 PRINT TAB(10)"I AM X AND YOU ARE O"
50 PRINT TAB(10)"MY MOVE FIRST"
55 PRINT
60 REM MAKE SQUARES BLANK
70 REM ******************
80 FOR I=1 TO 9
90 B$ (I)=" "
100 NEXT I
110 INPUT"ARE YOU READY";A$
120 IF A$<>"Y" THEN110
130 REM COMPUTER MOVE
140 REM **************
150 C=INT((RND(1)*9)+1)
160 IF  B$(C)<>" "THEN150
170 B$(C)="X"
180 T=T+1
190 GOSUB310
200 D$="X"
210 GOSUB470
220 REM PLAYERS MOVE
230 REM *************
235 PRINT
240 INPUT "YOUR MOVE (1 TO 9)";M
250 IF B$(M)<>" "THEN240
260 B$(M)="O"
270 T=T+1
```

(continued)

```
280 D$="O"
290 GOSUB470
300 GOTO130
310 REM DRAW THE GRID
320 REM *************
330 PRINT "□"
340 PRINT:PRINT:PRINT
350 PRINT TAB(5)"   |   |"
360 PRINT TAB(6) B$(1);" | ";B$(2);" | ";B$(3)
370 PRINT TAB(5)"   |   |"
380 PRINT TAB(5)"-----------"
390 PRINT TAB(5)"   |   |"
400 PRINT TAB(6) B$(4);" | ";B$(5);" | ";B$(6)
410 PRINT TAB(5)"   |   |"
420 PRINT TAB(5)"-----------"
430 PRINT TAB(5)"   |   |"
440 PRINT TAB(6) B$(7);" | ";B$(8);" | ";B$(9)
450 PRINT TAB(5)"   |   |"
460 RETURN
470 REM CHECK FOR WIN
480 REM *************
490 IF B$(1)=D$ AND B$(2) = D$ AND B$(3)=D$ THEN630
500 IF B$(4)=D$ AND B$(5) = D$ AND B$(6)=D$ THEN630
510 IF B$(7)=D$ AND B$(8) = D$ AND B$(9)=D$ THEN630
520 IF B$(1)=D$ AND B$(4) = D$ AND B$(7)=D$ THEN630
530 IF B$(2)=D$ AND B$(5) = D$ AND B$(8)=D$ THEN630
540 IF B$(3)=D$ AND B$(6) = D$ AND B$(9)=D$ THEN630
550 IF B$(3)=D$ AND B$(5) = D$ AND B$(7)=D$ THEN630
560 IF B$(1)=D$ AND B$(5) = D$ AND B$(9)=D$ THEN630
570 REM CHECK FOR TIE
580 REM *************
590 IF T=9 THEN660
600 RETURN
610 REM END MESSAGES
620 REM *************
630 GOSUB310
635 PRINT
640 PRINT D$;" WINS"
650 GOTO670
660 PRINT "IT'S A TIE"
670 INPUT "ANOTHER GAME";A$
680 IF LEFT$(A$,1)="Y"THEN10
690 END
```

Statements 10 through 120 explain the game and ask if the player is ready.

Statements 130 through 180 initialize the array to hold the moves and the counter to count the number of moves. The counter is then used to test for a tie game.

Statements 190 through 270 make the computer's move. Notice that statement 250 calls a subroutine to draw the grid.

Statement 260 sets the dummy variable D$ = "X," the computer's symbol, before the program branches to check for a win or a tie in statement 270.

Statements 290 through 360 are for the player's move.

Questions & Answers

How many subroutines are called in the player's section of the program?

Answer: 1

What does the subroutine do?

Answer: Checks for a win or a tie.

What happens in statement 340?

Answer:

The dummy variable is set to "0," the player's symbol.

The two subroutines could be written using different strategies. I chose these:

- The subroutine starting at line 500 prints the grid. It uses PRINT statements, and each time it is executed prints the current contents of the array B$ in the grid.
- The subroutine to check for a win or a tie first checks for a win by checking every possible winning combination. If all the tests fail, then the program tests to see if nine moves have been made. This test is in statement 80.

If the game is won or tied, the appropiate message is printed and the game ends.

What happens if less than nine moves have been made?

Answer: The subroutine ends and control returns to the main program.

Before we leave arrays and subroutines, let's look at the problem of programming a game of "BATTLESHIP." Both the player and the computer have a board, or grid, with 24 squares. Each can hide five battleships on their board. They then try to sink each other's battleships by guessing the numbers of the squares in which the opponent has hidden them. The winner is the first one to sink the opponent's five ships.

This is different than the tic-tac-toe game because here both the player and the computer are guessing squares at random.

Figure 5.2 shows the screen while the game is in progress.

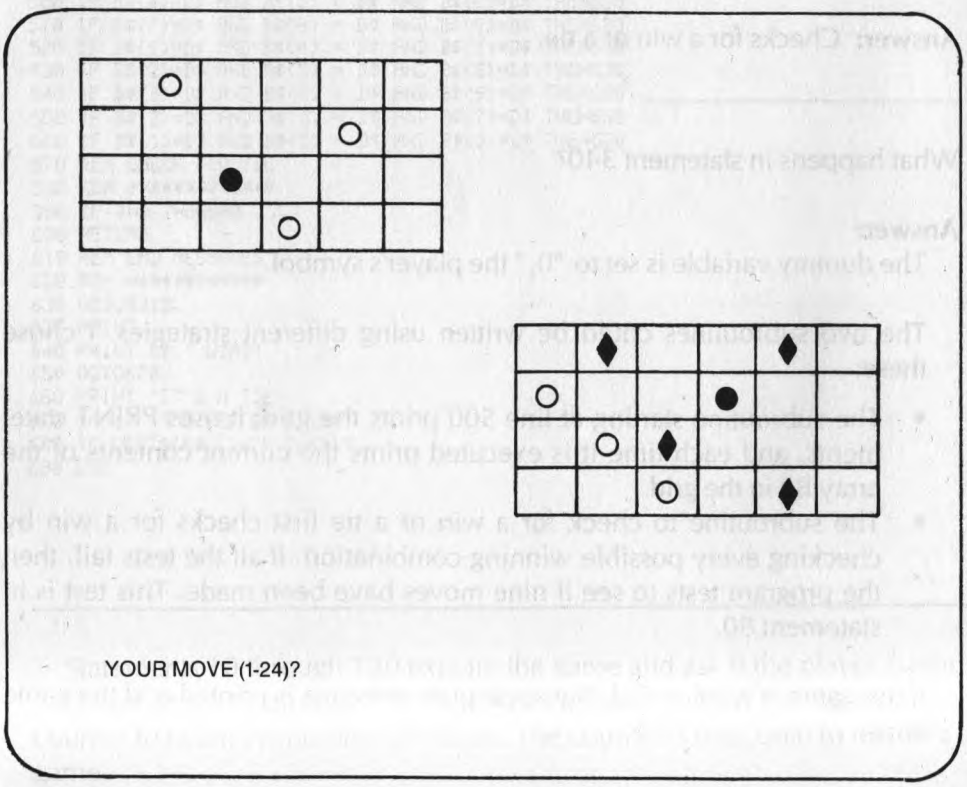
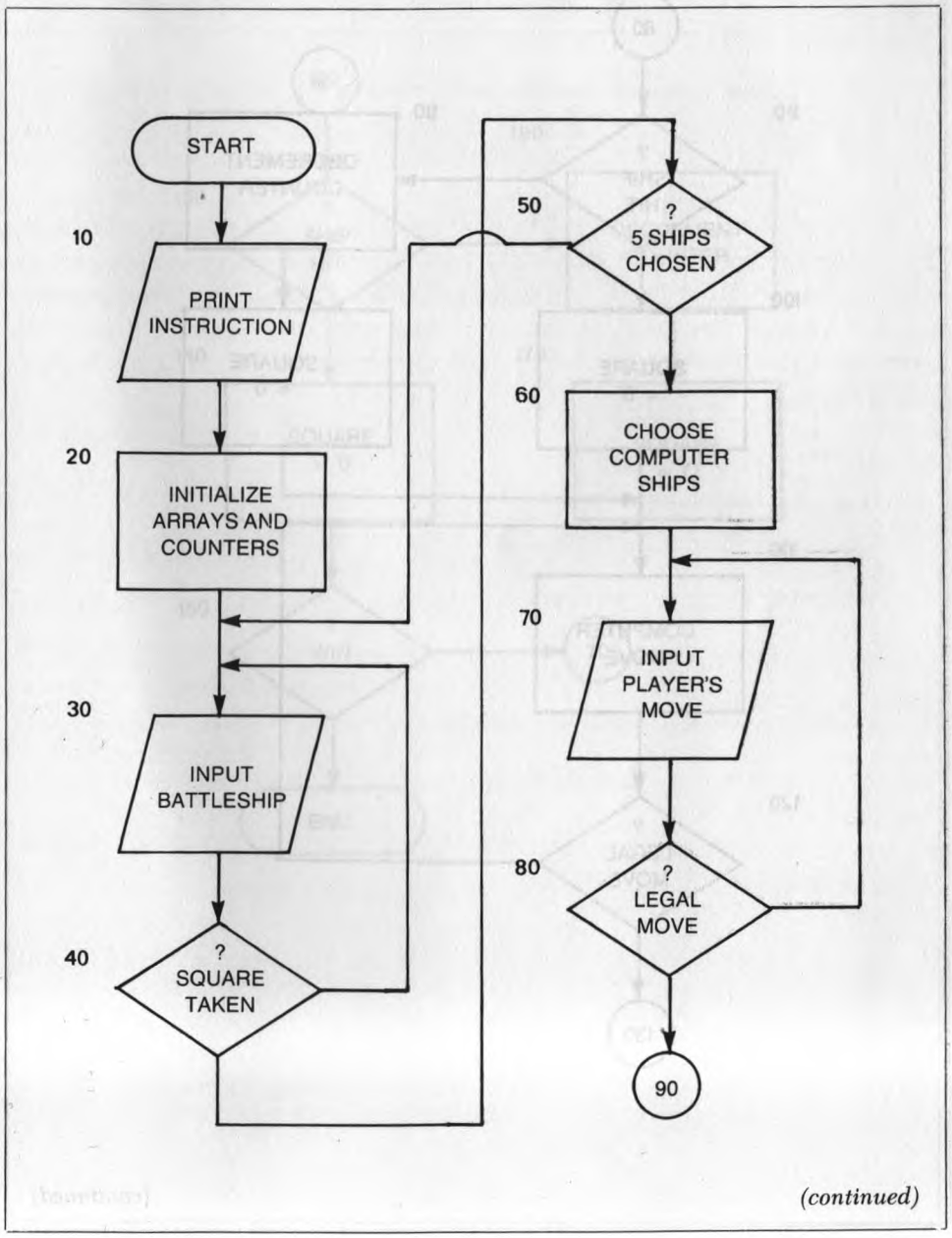


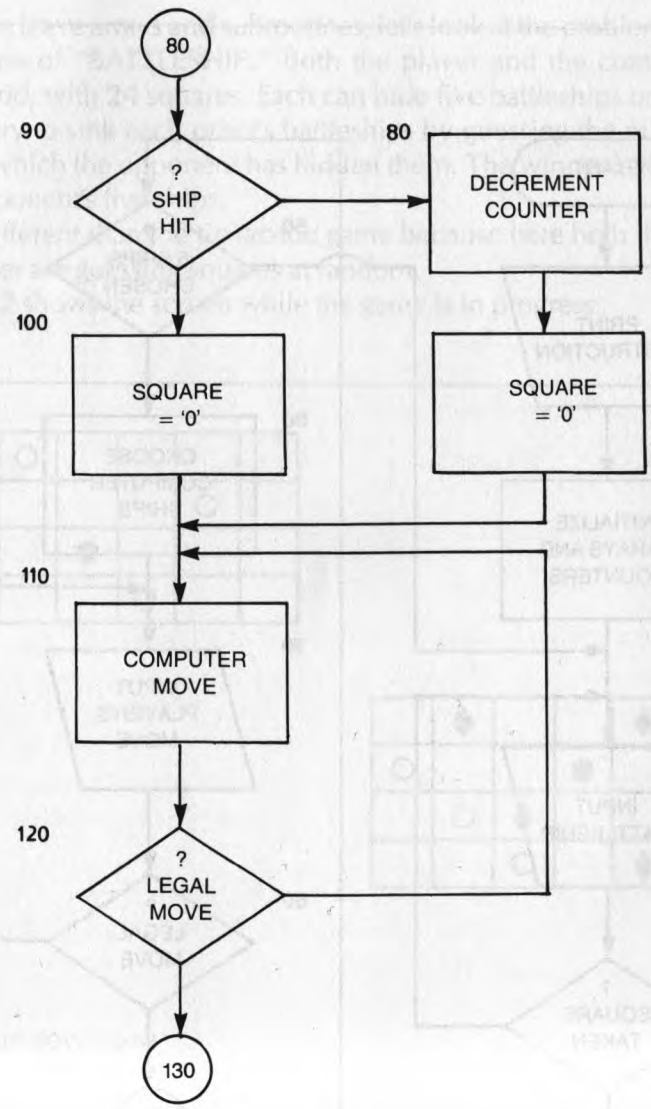
Figure 5.2 Battleship screen.

The top board is the computer's and the bottom board is the player's. The diamonds are the player's ships, the open circles are "misses" and the solid circles are "hits."

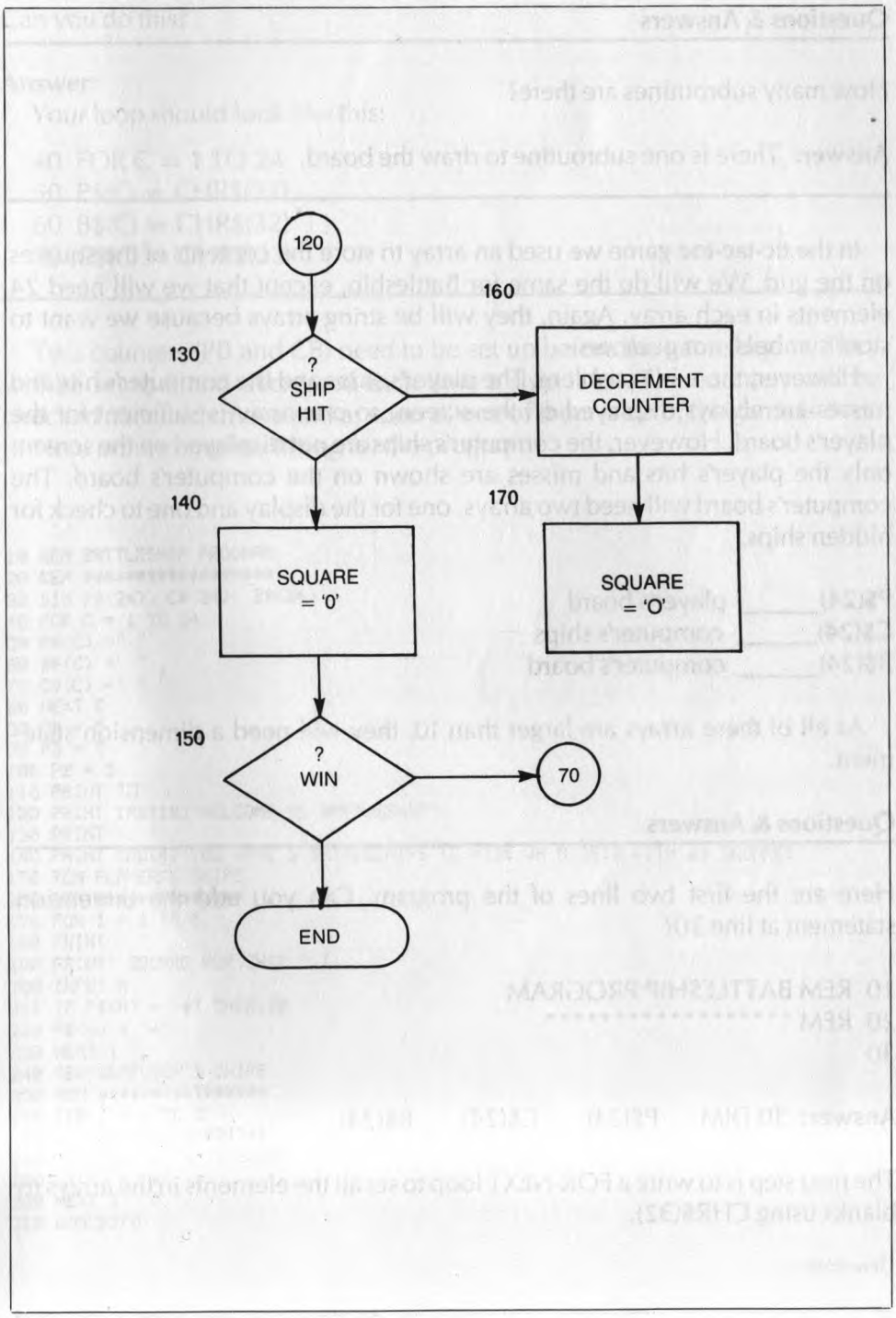
Here is the flowchart for the program.



(continued)


80
90
?
SHIP
HIT
80
DECREMENT
COUNTER
100
SQUARE
= '0'
SQUARE
= '0'
110
COMPUTER
MOVE
120
?
LEGAL
MOVE
130

(continued)


120
130
?
SHIP
HIT
160
DECREMENT
COUNTER
140
SQUARE
= '0'
170
SQUARE
= 'O'
150
?
WIN
70
END

Questions & Answers

How many subroutines are there?

Answer: There is one subroutine to draw the board.

In the tic-tac-toe game we used an array to store the contents of the squares on the grid. We will do the same for Battleship, except that we will need 24 elements in each array. Again, they will be string arrays because we want to store symbols, not numbers.

However, there is a problem. The player's ships and the computer's hits and misses are always displayed on the screen, so one array is sufficient for the player's board. However, the computer's ships are not displayed on the screen: only the player's hits and misses are shown on the computer's board. The computer's board will need two arrays, one for the display and one to check for hidden ships.

P$(24)_____ player's board
C$(24)_____ computer's ships
B$(24)_____ computer's board

As all of these arrays are larger than 10, they will need a dimension statement.

Questions & Answers

Here are the first two lines of the program. Can you add the dimension statement at line 30?

```
10 REM BATTLESHIP PROGRAM
20 REM ******************
30
```

Answer: 30 DIM P$(24), C$(24), B$(24)

The next step is to write a FOR-NEXT loop to set all the elements in the arrays to blanks using CHR$(32).

Can you do this?

Answer:

Your loop should look like this:

```
40 FOR C = 1 TO 24
50 P$(C) = CHR$(32)
60 B$(C) = CHR$(32)
70 C$(C) = CHR$(32)
```

Two counters (PB and CB) need to be set up before the game begins. They will be set to 5 and decreased each time a ship is hit. The counters will be checked after each move and as soon as one of them is zero, the game is over.

Here is the complete listing for the program:

```
10 REM BATTLESHIP PROGRAM
20 REM ******************
30 DIM P$(24), C$(24), B$(24)
40 FOR C = 1 TO 24
50 P$(C) =" "
60 B$(C) =" "
70 C$(C) =" "
80 NEXT C
90 CB = 5
95 PB = 5
100 PB = 5
110 PRINT "□"
120 PRINT TAB(10)"WELCOME TO BATTLESHIP"
130 PRINT
140 PRINT TAB(4)"YOU HAVE 5 BATTLESHIPS TO HIDE ON A GRID WITH 24 SQUARES"
150 REM PLAYER'S SHIPS
160 REM **************
170 FOR I = 1 TO 5
180 PRINT
190 PRINT" SQUARE FOR SHIP ";I;
200 INPUT N
210 IF P$(N) = "♦" THEN190
220 P$(N) = "♦"
230 NEXT I
240 REM COMPUTER'S SHIPS
250 REM ****************
260 FOR I = 1 TO 5
270 N = INT(RND(0)*24)+1
280 IF C$(N) = "♦" THEN270
290 C$(N) = "♦"
300 NEXT I
310 GOSUB570
```

(continued)

```
320 REM PLAYER'S MOVE
330 REM *************
340 INPUT "YOUR MOVE (1-24)";M
350 IF B$(M) = "●" OR B$(M) = "○" THEN340
360 IF C$(M) <>"♦" THEN390
370 CB = CB-1
380 B$(M) = "●"
385 GOTO 400
390 B$(M) = "○"
400 REM COMPUTER MOVE
410 REM *************
420 M = INT(RND(0)*24)+1
430 IF P$(M) ="●" OR P$(M) = "○" THEN420
440 IF P$(M)<> "♦" THEN470
450 PB = PB-1
460 P$(M) = "●"
465 GOTO 480
470 P$(M)= "○"
480 GOSUB570
490 IF PB = 0 OR CB = 0 THEN510
500 GOTO320
510 IF PB = 0 THEN550
520 PRINT
530 PRINTTAB(10) "YOU WIN "
540 GOTO560
550 PRINT TAB(10) " I WIN"
560 END
570 REM SUBROUTINE TO DRAW BOARDS
580 REM **************************
590 PRINT"□"
600 N=0
610 PRINTTAB(5);
620 FOR X = 1 TO 13
630 PRINT"-";
640 NEXT X
650 PRINT
660 PRINTTAB(5);
670 FOR I = 1 TO 4
675 PRINT TAB(5);
680 FOR J = 1 TO 6
690 N = N + 1
700 PRINT"|";B$(N);
710 NEXT J
720 PRINT"|"
730 PRINT TAB(5);
740 FOR X = 1 TO 13
750 PRINT"-";
760 NEXT X
770 PRINT
780 NEXT I
790 N=0
800 PRINT
810 PRINTTAB(18);
820 FOR X = 1 TO 13
830 PRINT"-";
840 NEXT X
```

(continued)

```
850 PRINT
860 PRINTTAB(18);
870 FOR I = 1 TO 4
880 PRINT TAB(18);
890 FOR J = 1 TO 6
900 N = N + 1
910 PRINT"|";P$(N);
920 NEXT J
930 PRINT"|"
940 PRINTTAB(18);
950 FOR X = 1 TO 13
960 PRINT"-";
970 NEXT X
980 PRINT
990 NEXT I
1000 PRINT
1010 RETURN
```

The remark statements identify the sections of the program, and you can see that the main program comes before the subroutine.

Questions & Answers

Why isn't the subroutine written in the program where it is called at statement 250?

Answer: Because it interrupts the flow of the program.

SELF-CHECK

At the risk of repeating myself, please do not skip this self-check on repetition!

1. Is the FOR-NEXT loop a DO-UNTIL loop or a DO-WHILE loop?
2. Every FOR must have a _____.
3. (a) Draw a flowchart for a loop to print the numbers from 1 to 100.
   (b) Write the program to print the numbers from 1 to 100.
4. Use nested loops to write ten rows of the numbers 1 to 10.

5. Look at this program.

```
10 W$ = "COMMODORE"
20 FOR C = 9 TO 1 STEP -1
30 PRINT MID$ (W$,C,1)
40 NEXT C
```

   (a) How many times will this loop be repeated?
   (b) What will the output look like?

6. Here is one of my favorite programs. I call it MOMPOWER.

```
10 X = INT(RND(0)*5) +1
20 FOR C = 1 TO X
30 READ D$
40 NEXT C
50 PRINT D$
60 END
70 DATA CLEAN YOUR ROOM, TAKE OUT THE GARBAGE, DO THE
   DISHES, SET THE TABLE, WALK THE DOG
```

   (a) What is happening in statement 10?
   (b) What will happen when X = 2?
   (c) What will happen when X = 5?

7. Are these statements true or false?

   The RESTORE statement:

   _____ (a) resets the loop counter.
   _____ (b) reads the next item in the DATA statement.
   _____ (c) resets the READ statement.
   _____ (d) resets the pointer to the first item in the first DATA statement.

8. Write a dimension statement for a string array with 120 elements.

9. In the array T(10,5):

   (a) how many rows are there?
   (b) how many columns are there?

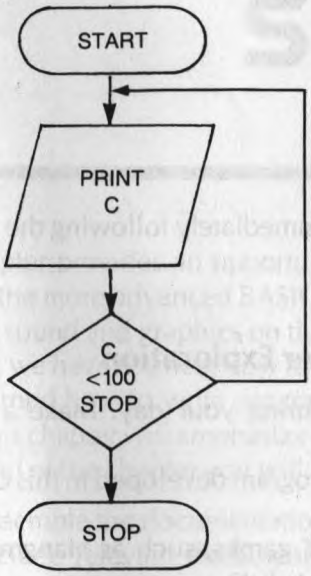
10. The statement DEF FNA(X) would be used to:

   (a) execute a user-defined function in a program.
   (b) define a user-defined function in a program.

11. Are these statements true or false?
    (a) Subroutines end with a RETURN statement.
    (b) A program may have more than one subroutine.
    (c) Subroutines are normally placed after the main program.
    (d) GOSUB is used to branch to a subroutine.
    (e) A subroutine returns to the statement that called it.

Answers to SELF-CHECK

1. DO-UNTIL.
2. NEXT
3. (a)



    (b)
```
10 FOR C = 1 TO 100
20 PRINT C
30 NEXT C
```

4.
```
10 FOR C = 1 TO 10
20 FOR D = 1 TO 10
30 PRINT D
40 NEXT D
50 NEXT C
```

5. (a) 9
   (b) ERODOMMOC

6. (a) X will be a random number between 1 and 5.
   (b) The program will print TAKE OUT THE GARBAGE.
   (c) The program will print WALK THE DOG.
7. (a) False.
   (b) False.
   (c) False.
   (d) True.
8. 10 DIM N$(120)
9. (a) 10
   (b) 5
10. (b)
11. (a) True.
    (b) True.
    (c) True.
    (d) True.
    (e) False. It returns to the statement immediately following the GOSUB.

Suggestions for Further Exploration

1. Identify the tasks that you repeat during your day. Make a habit of identifying repeated processes.
2. Try expanding the check-balancing program developed in this chapter to suit your own needs.
3. Program some other 'pen and pencil' games, such as Hangman. How about using computer terms for the data list?

CHAPTER 6

Programming Style

This chapter provides an opportunity to review and regroup before learning some of the more advanced BASIC concepts and venturing into the wonderful world of sound and graphics on the Commodore 64.

So far, we have learned, how to analyze problems using flowcharts, and we have learned how to write programs in BASIC that use repetitions and decisions. This chapter will emphasize organizing these programming skills, so that by the end of the chapter you will be able to:

- Assemble the documentation for a program.
- Include relevant documentation in your program using remark statements.
- Understand and use some debugging techniques.
- Understand the concept of TOP-DOWN design.
- Break an idea into small blocks or modules for programming.
- Streamline a working program by writing multiple statements.
- Use abbreviations for BASIC keywords.

Documentation

We will start by considering how to make our programs more 'readable' and easier to understand. The documentation of a program is one important aid in understanding its logic. In Chapter 3 we mentioned briefly that it was important to keep the documentation for a program in a safe place. In that chapter, documentation meant all the paperwork we had generated in analyzing the problem and drawing the flowchart. Now we can add some more items to the documentation. The documentation for a program may include any of the following:

- A statement of the program's purpose.
- A flowchart.
- The most current program listing.
- As the program is revised, copies of the earlier versions.

These kinds of documents are important for several reasons:

1. They are very useful as aids in explaining the program to someone else.
2. They are invaluable in debugging any errors that may show up at a later date.
3. They will refresh your memory if you have not used the program for a while.

Many programmers feel that keeping the documentation up to date is a tedious chore - which it is - but do it anyway. For there may come that awful day when you wipe out a disk or tape and need the documentation to reconstruct the program.

Questions & Answers

Which of the following would be considered part of the documentation of the program?

_____ (a) Flowchart.
_____ (b) Program listing update 7/8/83.
_____ (c) Aunt Sue's pickle recipe.
_____ (d) Local Commodore dealer's phone number.
_____ (e) Notes from family discussion on proposed household budget.

Answer: (a), (b), and (e). (I would suggest that (d) be taped to your phone!)

Backup Copies

There is another solution for recovering programs when a tape or disk is wiped out.

WARNING

ALWAYS MAKE A BACK-UP COPY OF YOUR PROGRAMS.

You might want to make a note to remind yourself of this often forgotten piece of wisdom and tape it to your storage device.

If you use tapes, this means saving your program on a spare tape. Programs on disk can be saved singly onto a new disk, or may be copied to a new disk using a commercially available copy program that copies the entire disk.

When you have made a back-up copy of a program, keep it in a safe place. In an earlier chapter we discussed environmental hazards involved in storing tapes and disks.

Questions & Answers

Three environmental hazards to tapes or disk are _____, _____, and _____.

Answer: Take your pick from this list: sunlight, heat, magnetic forces, dust, grease, scratches and folding.

Remark Statements

There is another form of documentation that is very useful to the programmer: REM statements. We used these remark statements in the Math Facts program in Chapter 4 and again in Chapter 5 when we discussed subroutines.

Here is an example of a REM statement to be included before a print subroutine:

```
50 REM THIS SUBROUTINE PRINTS THE RESULT
```

Notes like this help to break up the listing and to identify what is happening in the program at each step. Remark statements can also be used to include the title, author and data written at the beginning of a program.

```
2 REM MISSION TO MARS
4 REM WRITTEN BY 1. N. VADER
6 REM MARCH 1984 AD
```

This example tells us the name of the program, the author and when it was written. Suppose, for example, that after playing MISSION TO MARS for three months, it is expanded to include Intergalactic forces. Then we would add two additional remark statements:

```
8 REM REVISED JUNE 1984 AD
10 REM TO INCLUDE INTERGALACTIC FORCES
```

Here is another example:

Belinda Brown has written a program to keep track of the family budget. In the first week of November she finds out that Bob, "Bucky" to his friends, needs $2600 worth of orthodonture. Belinda has to revise the family budget.

Questions & Answers

How would you change the following statements?

```
2 REM BROWN'S BASIC BUDGET PROGRAM
4 REM WRITTEN BY THE TALENTED AND PENNY-PINCHING
6 REM BELINDA 'BOOTS' BROWN
8 REM 23 AUGUST 1983
```

Answer:

Add a statement similar to this one:

```
8 REM REVISED BY B. B. NOVEMBER 1983
```

Indentation

Another method frequently used to clarify programs is indentation, which sets apart sections of the program. It is very useful in situations using nested loops where the FOR and NEXT statements need to be matched. A colon at the

beginning of a statement will allow you to indent the statement. Notice how easy it is to check that each FOR has a matching NEXT.

```
10 FOR C = 1 TO 12
20 :  FOR K = 1 TO 10
30 :   PRINT C*K
40 :  NEXT K
50 NEXT C
```

Questions & Answers

Indent this listing and check the nested loops. Can you make the necessary corrections?

```
10 FOR I = 1 TO 5
20 FOR J = 10 TO 0 STEP -2
30 FOR K = 0 TO 3*J
40 PRINT K
50 NEXT J
60 PRINT
70 NEXT I
```

Answer:

The K loop is missing the NEXT statement. Add 45 NEXT K

```
10 FOR I = 1 TO 5
20 :  FOR J = 10 TO 0 STEP -2
30 :   FOR K = 0 TO 3*J
40 :    PRINT K
45 :   NEXT K
50 :  NEXT J
60 : PRINT
70 NEXT I
```

Spacing

As I expect you have found by now, the Commodore 64 is very forgiving about the way you space your program. The computer is able to separate the various

BASIC statements, variable names, operators, etc., in a program line. For example, this line:

```
10IFX=22THENPRINT" THAT'S ALL"
```

will be interpreted by the computer to mean this:

```
10 IF X = 22 THEN PRINT " THAT'S ALL"
```

Or, if you put extra spaces in the line, the computer will take care of that, too.

There are two occasions, however, when the spacing *is* critical. One is with the use of parentheses. You may not leave a space between the function and a left parenthesis. For example:

```
RND(1)      not   RND (1)
```

Second, the spacing is critical when something is written between quotation marks. Anything written between quotation marks will be printed on the screen exactly as it appears. The statement:

```
60  PRINT "THE COMMODORE64ISAGREATMACHINE"
```

will not print:

```
THE COMMODORE 64 IS A GREAT MACHINE
```

Questions & Answers

Which of the following statements is correct?

```
(a)  10 PRINT TAB (5) N$,A$
(b)  10 PRINT "FARE THEE WELL ";N$
(c)  10 PRINT SPC(10) "REPORT GRADE 2 SCORES"
(d)  10 X = INT( RND(1) * 10)
(e)  10 PRINT"HELLO";N$
```

Answer: (b), (c), and (d). In (a), there is a space between TAB and the left parenthesis. In (e), no space will be printed between HELLO and the string variable N$.

While we are discussing typing skills and the Commodore 64's ability to forgive some of our mistakes by inserting or deleting extra spaces - THE COMMODORE 64 WILL NOT CORRECT YOUR SPELLING!

It is always worth taking a few minutes to proof-read your program before you run it.

Debugging

Even with the best preparation in the world, a program will have some software 'bugs' the first time it is run.

There are some ways of speeding up the debugging process. The first step is to recognize the type of error that caused the problem. There are three main categories of errors:

- syntax errors,
- procedural errors, and
- analytical errors.

We will look at each kind of error separately and discuss some ways of debugging them.

Syntax Errors

The first kind of error is a syntax error. Syntax errors occur when the computer does not recognize a command or a specific arrangement of commands as an executable program statement. The main causes of syntax errors are typing errors, incorrect use of punctuation, and use of the letter O instead of the number 0.

Syntax errors generate this error message:

```
?SYNTAX ERROR IN LINE xxx
```

Syntax error messages are helpful because they include the line number in the program where the error occurred. You can correct a syntax error by either retyping the statement or editing the statement on the screen using the cursor keys.

Procedural Errors

The second type of error is a procedural error. Procedural errors occur when the program attempts the impossible or the illegal. For example, the following statement is written in correct BASIC.

```
20 IF A <18 THEN 60
```

However, it will generate a procedural error if the program does not contain a statement number 60. Other common procedural errors are:

OUT-OF-DATA	Trying to read more data items than are listed in data statements or forgetting the RESTORE statement.
NEXT WITHOUT FOR	Omitting the NEXT statement or using a variable in the NEXT statement that does not match the variable in the FOR statement. Also caused by improperly nested loops.
UNDEF'D STATEMENT	The program is trying to branch to a statement that does not exist in the program.
DIVISION BY ZERO	The program is evaluating an expression that includes division by zero, which is not allowed.

The complete listing of procedural errors is included in Appendix A.

There are several techniques you can use to debug procedural errors in a program. Here are some that I find helpful.

After the program has been run, the variables in memory are not cleared. This means that you can use a direct command to PRINT the content of a variable name. Try this program:

```
10 A = 10
20 B = 5
30 C = A + B
40 PRINT A B C
```

Now check the variables by using a direct command like this:

```
PRINT A
```

The computer will respond with the number 10.

Questions & Answers

What will happen when you use PRINT C?

Answer: The computer will print 15.

This technique can be very useful when a program 'crashes.' That is, it stops working before it gets to the end. You can use it to check the contents of variables and use this information to see how far the program executed before it crashed. For example, let's use the following program:

```
10 FOR N = 1 TO 10
20 READ N$,A
30 NEXT N
40 PRINT N$,A
50 END
60 DATA TOM,12,DICK,11,JANE,11,HARRY,12
70 DATA  MARY,12,BETH,12,JOHN,12,JASON,11
```

When this program is run, it generates an 'OUT OF DATA ERROR.' We can check the value N by using a direct PRINT command.

Questions & Answers

(a) What is N?
(b) Using the value of N, can you debug the program?

Answer:

(a) N is 8.
(b) Statement 10 sets the loop to repeat ten times, but there are only 8 pairs of variables, N$ and A, in the DATA statements.

We need to change statement 10 so that the loop repeats eight times.

```
10 FOR N = 1 TO 8
```

Sometimes it helps to include a PRINT statement in the program, as a test. Look at this program, for calculating a 6% sales tax.

```
10 INPUT " PRICE OF ITEM "; P1
20 INPUT " PRICE OF ITEM "; P2
30 C = P1 – P2
40 PRINT C
50 T = C * 6 / 100
60 PRINT "TOTAL COST INCLUDING SALES TAX IS $";T
```

Statement 40 will print C and allow us to check its value before the sales tax calculation is performed in statement 50. If P1 and P2 are prices, then the print statement should alert us to the fact that there is an error in statement 30 - the prices are being subtracted, not added, to get the total cost. Once the program is debugged, statement 40 can be deleted. To delete a statement, we enter the statement number and press RETURN.

Analytical Errors

Analytical errors are more difficult to identify than syntactical or procedural errors. When an analytical error occurs, there is no error message, and the program runs, but the results are not as expected. The output may be incorrect, or the program may execute but nothing appears at all as output, or there is an endless loop and execution never stops - until you press RUN/STOP.

One way to find the cause of an endless loop is to use STOP statements. Endless loops occur when there is an analytical error in a FOR-NEXT loop or at a branch in the program, such as an IF-THEN statement. GOTO statements are also the frequent cause of endless loops, one reason we are trying not to use GOTO statements. If we put a STOP statement immediately after the NEXT statement in a loop, or before the branch statement, then we can find out two things:

1. Does the program actually reach this STOP statement? If so, then the program is executing properly, up to this point.
2. What are the values of the variables at this time? We can use the direct mode PRINT to check this.

The best strategy is to place the STOP after the first control structure in the program. A control structure is a loop or a branch in the program. If the first loop or branch is executing correctly, then remove the STOP and place it after the next control structure, repeating the process until you find the error.

Let's look at this program to print the months of the year that have thirty days:

```
10 FOR C = 1 TO 12
20 READ M$,D
30 IF D <> 30 THEN 30
40 PRINT M$
50 NEXT C
60 DATA JANUARY,31,FEBRUARY,28,MARCH,31,APRIL,30
70 DATA MAY,31,JUNE,30,JULY,31,AUGUST,31
80 DATA SEPTEMBER,30,OCTOBER,31,NOVEMBER,30,DECEMBER,31
```

Type in this program without making any changes. When you run it, no output is produced, and it will not stop until you press the RUN/STOP key.

There are two control structures in this program, a FOR-NEXT loop and an IF statement. The first place to add a STOP statement is after the IF statement, like this:

35 STOP

Questions & Answers

When the program is run, we are still in an endless loop, and do not reach the STOP at line 35. When you stop the program with RUN/STOP, do a PRINT C. The result is 1, so the program is still in the first loop. How would you check the value of D?

Answer: PRINT D

In this case D is 31. As 31 is not equal to 30, the test condition in line 30 is true. The program should have branched on the true condition to the end of the loop without printing the month. And there's the bug. Line 30 sends the program to line 30 if D is not equal to 30. Instead, it should send the program to line 50 and repeat the loop.

Correct statement 30, remove the STOP statement and the program should run perfectly.

We can use this same technique for programs that give incorrect results or no results at all. Here is the program to calculate the total number of gifts you will have received from your true love at the end of the song ('Twelve Days of Christmas'.)

```
10 REM TWELVE DAYS OF XMAS
20 FOR D = 1 TO 12
30 T = 0
40 FOR G = 1 TO D
50 T = T + G
60 NEXT G
70 PRINT "DAY"; D; "TOTAL NUMBER OF GIFTS IS"; T
80 NEXT D
90 END
```

The first run of this program shows that the output is not correct. The total number of gifts is too low. So we can test the execution of the program using STOP statements.

Questions & Answers

Where would be the best place to put the first STOP statement?

_____ (a) Before statement 20.
_____ (b) After statement 60.
_____ (c) After statement 80.

Answer: (b) After the end of the nested loop.

Try placing the STOP at line 65 and running the program. Now check the content of variables D and T, using a direct PRINT command. D will be 1 and T will be 1: this is what we would expect. Now type in CONT and the program will resume execution. The next time it stops, check D and T again. This time D is 2, as expected, but T is 3, which is not correct. T should be four, 2 turtle doves and a partridge from day 2 plus 1 partridge from day 1. The variable T, which is supposed to be storing a cumulative total, is being cleared each day in statement 30. The variable T needs to be initialized before the program enters the first loop.

```
10 REM TWELVE DAYS OF XMAS
20 T = 0
30 FOR D = 1 TO 12
40 FOR G = 1 TO D
50 T = T + G
60 NEXT G
70 PRINT "DAY"; D; "TOTAL NUMBER OF GIFTS IS";T
80 NEXT D
90 END
```

The direct command continue, CONT, which we used in this example, is very useful. It will continue the execution of a program if possible. CONT will work if the execution stopped because of an error, or if you have made any changes to the program while the execution has been stopped.

When you have a long program, such as the Math Facts program, it is possible to RUN just a part of it. This technique is helpful when there is a problem with just one section. To test the addition section of the final, subroutine version of the program, we would add a STOP statement before the RETURN statement and use this command:

RUN 100

This will start the program running at line 100, the beginning of the addition subroutine. It will stop before the statement to return to the main menu.

Questions & Answers

To test the multiplication section you would add a STOP and use:

_____ (a) RUN 100
_____ (b) RUN 200
_____ (c) RUN 300
_____ (d) RUN 400

Answer: (c)

I have found these strategies very useful for debugging programs. There are more sophisticated techniques, for example, a special program that will TRACE the execution of a program. A TRACE program utilizes part of the operating system to print to the screen each statement number as it is executed. You can see which paths the program follows. You may find it useful as your programs become longer and more complex. However, I hope that the simpler techniques we have discussed here will help you find most of your bugs.

Questions & Answers

To review, the three categories of software bugs are _____, _____, and _____.

Answer: Syntactical, procedural, analytical.

Top-Down Design

Program design is another important factor in reducing errors. The best programs follow a TOP-DOWN DESIGN. This means that they start executing at the beginning and the control, or flow, of logic moves towards the end without jumping back up into the program. Here are some things that you can do to make your programs TOP-DOWN programs.

1. Put all dimension statements at the beginning of the program.
2. Initialize all counters and variables at the beginning of the program.
3. Keep the main program together.
4. Put all subroutines after the main program.
5. Put all DATA statements at the end of the program.
6. The unbranched paths through the program should be kept together if possible.

The important idea behind TOP-DOWN DESIGN is having all the preparation or "housekeeping" at the beginning of the program, followed by the main program, with the data statements at the end.

Questions & Answers

Are these statements true or false?

_____ (a) DATA should be with the READ STATEMENTS.
_____ (b) DIM statements are part of the preparation of the program and should be at the beginning.
_____ (c) The flow of the logic should be straight through the program without breaks or jumps.

Answer:
(a) False
(b) True
(c) True

Streamlining a Program

Up until now we have been discussing planning and developing a program. Once the progam is written and running smoothly, there are some things we can do to make it execute faster and minimize the amount of space the program takes up in computer memory.

Before you start to streamline a program, there are two things you should do:

- Back up the program by making a copy on another cassette or disk.
- If possible, print a program listing to add to the program documentation.

Now, if something goes wrong while we are adapting the program, we will not have lost it entirely.

If the program is very large, it can be made smaller by removing all the remark statements. Now that we know that the program is working properly and that we have a back-up with all the program remarks intact, we can remove them from the streamlined version.

Questions & Answers

Remark statements do/do not effect the execution of the program.

Answer: Do not. They are there for the programmer's information only. The computer ignores them.

Multiple Statements

Another way to compact the program and speed up the execution time is to consolidate some of the statements into multiple statements.

A multiple statement looks like this:

```
# <statement>: <statement>:<statement>
```

There is one statement number, #, followed by more than one statement. The statements are separated by colons.

Short statements are the most obvious choice for forming multiple statements. For example, assignment statements to initialize a variable can be combined like this:

```
30 T = 0: A = 10
```

Sometimes a short FOR-NEXT loop can be written as one multiple statement.

```
10 FOR C = 1 TO 10: PRINT C: NEXT C
```

The limit to the length of a multiple statement is the same as that for a single statement. A multiple statement cannot exceed 80 characters.

Questions & Answers

Which of these are legal multiple statements?

_____ (a) 10 IF A$ = "NO" THEN PRINT "GOODBYE": GOTO 500
_____ (b) 10 J = 0, K = 0, D = 0; T$ = "MONTHLY REPORT"
_____ (c) 10 FOR I = 1 TO 10: FOR J = 10 TO 1 STEP -1: PRINT LEFT$(N$,1,0) : NEXT J : PRINT : NEXT I
_____ (d) 10 PRINT: PRINT: PRINT (TAB 10) "COMMODORE 64": PRINT: PRINT: PRINT "WELCOMES YOU"

Answer:

(a) and (d) are legal multiple statements.

(b) Multiple statements are separated by colons. This statement should be:

```
J = 0: K = 0: D = 0: T$ = "MONTHLY REPORT"
```

(c) This statement is illegal because it has more than 80 characters.

Abbreviations

Some of the key BASIC statements can be abbreviated, allowing us to put more statements on a line. The most common abbreviation is a question mark (?) for PRINT. Most BASIC words can be abbreviated by typing the first character, then shifting to type the second. It will appear as a graphics character. There is a complete listing of abbreviations in Appendix F.

Abbreviations increase the amount of information that can be included in one statement, but when the program is listed by the computer the abbreviations are written out in full. This may cause problems if the line needs to be edited. You cannot use the screen editor; instead, the whole line will have to be retyped using abbreviations. Usually abbreviations are used to cram the code this way only after the program has been tested and all the errors corrected.

Using abbreviations for BASIC keywords will speed up the time taken to type in a program once you have become familiar with them.

Questions & Answers

Match the following abbreviations with the BASIC statement:

(a) E	(1) READ
(b) R	(2) RND
(c) R	(3) TAB
(d) T	(4) THEN
(e) T	(5) END

Answer: (a) 5 (b) 1 (c) 2 (d) 4 (e) 3

In Chapter 2, we talked about the importance of spaces in the program. While spaces are required in any string information we want printed out, they are not required in the rest of the program. We can save storage space by removing spaces in the program. For example:

```
10 FORC=1TO100:PRINTF=(C+32)*9/5:NEXTC:PRINT"DONE"
```

or

```
70 IFT$="THE END"ORC=100ANDD=0THEN500
```

Questions & Answers

Which of the following strategies will streamline a program?

_____ (a) Remove all REM statements.
_____ (b) Take out spaces wherever possible.
_____ (c) Use multiple statements.
_____ (d) Remove spaces from string.
_____ (e) Use abbreviated BASIC statements.

Answer: (a), (b), and (c) are good strategies.

There is no self-check for Chapter 6, but please practice using the techniques we have discussed here when writing your own programs.

CHAPTER 7

Computer Math

We cannot learn further programming skills for the Commodore 64 without understanding how the computer handles information. We need to understand how the computer stores information, in what form the information is stored, and how computers keep track of the information they have stored. Understanding these things will help us to write programs for datafiles, graphics and sound in future chapters.

By the end of this chapter you will be able to:

- Understand the BINARY numbering system.
- Convert binary numbers to decimal numbers and vice versa.
- Understand the concept of BITs and BYTEs.
- Understand the effect of binary math on logical operators.
- Understand the organization of the computer memory, including the relationships between machine language, the operating system and BASIC.

Decimal Numbering System

We will begin by discussing some of the math associated with computers, and how it relates to the math we use everyday.

Modern society uses the decimal numbering system, using the numerals zero through nine. We have a suitable number of fingers to keep track of the count. You may have heard the expression 'base 10.' This means that we use ten

numbers before we have to increase the 'place' and repeat the numbers. For example, when we add 1 to 9 we 'regroup' or 'carry' 1 to the 'tens' place and write the answer 10, like this:

tens	ones	
	9	
	1	+
1	0	

Another way of looking at a number is to break it down into multiples of its place value. For instance, the number 10 would be:

$10 \times 1 + 1 \times 0 = 10 + 0 = 10$

Here is another example:

	thousands	hundreds	tens	ones
(a)	7	3	8	4
(b)	5	6	1	2
(c)			5	9
(d)		4	0	2

The first number in this table, (a), can be written as:

$$7 \times 1000 + 3 \times 100 + 8 \times 10 + 4 \times 1 = 7000 + 300 + 80 + 4$$
$$= 7384$$

This is the number seven thousand three hundred and eighty-four.

Questions & Answers

Can you write out the other three numbers in the table using this method?

Answer:

(b) $5 \times 1000 + 6 \times 100 + 1 \times 10 + 2 \times 1 = 5000 + 600 + 10 + 2$
$= 5612$

This is the number five thousand six hundred and twelve.

(c) $5 \times 10 + 9 \times 1 = 50 + 9 = 59$
This is the number fifty-nine.
(d) $4 \times 100 + 0 \times 10 + 2 \times 1 = 400 + 0 + 2 = 402$
This is the number four hundred and two.

This review of our childhood math classes will come in very useful when we look at the way the Commodore 64 handles numbers.

Binary Math

The base 10, or decimal system, has not always been the standard numbering system. The Aztecs used a numbering system based on the number twenty, as did the ancestors of the French; the word for eighty in French translates as "four times twenty." There are some tribes in Africa that use a system based on the number five. The computer uses a numbering system based on the number two.

To understand the computer numbering system we need to refer to Chapter 1. When we connected the Commodore 64 computer system, we plugged the computer into the standard wall socket. One thing we know about electrical current is that it either flows through an electrical switch or it does not. There are only two possibilities. If the switch is connected to a light bulb, then the light is either on or off, as shown in Figure 7.1.

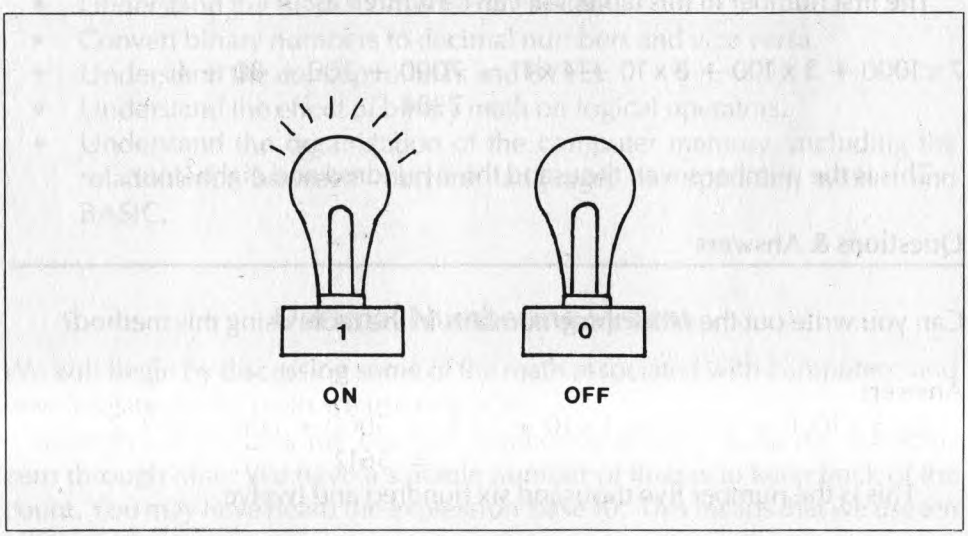


Figure 7.1 A lightbulb illustrates the two binary states- OFF and ON.

Bits

At an electrical level, the computer is a collection of switches or BITS (Binary InTegers). A bit has only two states, "on" and "off," and these are represented by the numbers 0 and 1. Following the example of our exercise with the decimal-numbering system, we will have to move up a place when we want to write the binary equivalent of the number two.

Questions & Answers

Binary	Decimal
0	0
1 +	
1	1
1 +	
10	2
1 +	
11	3

Answer:

11
1 +
100

What will be the binary equivalent of the number 4?

Keeping in mind that the binary number system only uses the numbers 1 and 0, can you complete the rest of this table?

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	
7	
8	
9	
10	

Answer:

Decimal	Binary
6	110
7	111
8	1000
9	1001
10	1010

By stringing BITS together, any decimal number can be represented. We can see this if we repeat the exercise used in the section on decimal numbers, but this time the places will be binary increments. Here is an example:

sixteen	eight	four	two	one
1	0	0	1	1

Notice that the binary number contains only ones and zeroes. It can be represented like this:

$1 \times 16 + 0 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 16 + 0 + 0 + 2 + 1 = 19$

So, this binary number - 10011 - is equivalent to the decimal number 19.

Questions & Answers

Here is an example for you to try. What is the decimal equivalent of this binary number?

sixteen	eight	four	two	one
1	1	1	1	1

Answer:

31

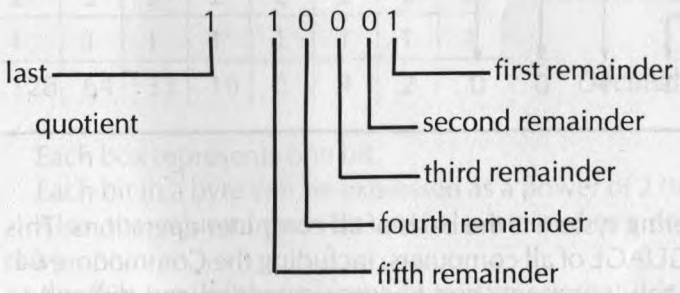
(1x16 + 1x8 + 1x4 + 1x2 + 1x1 = 16 + 8 + 4 + 2 + 1 = 31)

In the last section, we showed how to convert a binary number to a decimal number. But, we also need to do the reverse. One method of converting a decimal number to a binary one is to continue dividing the decimal number by two and look at the remainders.

Look at this example:

```
2 )49
2 )24    1  (first remainder)
2 )12    0  (second remainder)
2  )6    0  (third remainder)
2  )3    0  (fourth remainder)
     1   1  (fifth remainder)
(last quotient)
```

Now we write the binary number by putting the last quotient as the first binary digit and reversing the order of the remainders. The binary equivalent of 49 looks like this:



Here is another example:

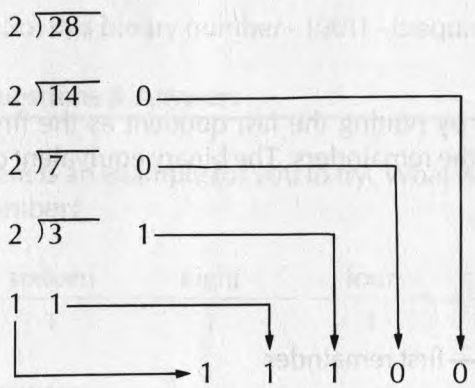
```
2 )28
2 )14    0
2 )7     0
2 )3     1
     1   1
```

Questions & Answers

Which of these is the binary equivalent of 27?

_____ (a) 0011
_____ (b) 11100
_____ (c) 111
_____ (d) 000111

Answer: (b) Start with the last quotient and then reverse the remainders, like this:



The BINARY numbering system is the basis of all computer operations. This is the MACHINE LANGUAGE of all computers, including the Commodore 64.

You can see that binary numbers soon become extended and difficult to remember with any accuracy. We would soon become hopelessly lost trying to write a program using only binary numbers - fortunately we do not have to do this. However, the concept of the BIT and the binary numbering system will be important to us when we program music and "sprites", the animated graphic feature of the Commodore 64.

The next step will be to find out how binary numbers are connected to the programs we write.

Questions & Answers

Before we move on, please fill in the blanks:

(a) The binary numbering system uses the numbers _____ and _____.
(b) The binary numbering system is a base _____ numbering system and the decimal numbering system is a base _____ numbering system.

Answer:
(a) 0,1
(b) 2,10

Bytes

The individual switches, or Bits, in a computer are grouped together into Bytes. One byte has eight bits. This is a diagram of a byte:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Powers of 2
1	1	1	1	1	1	1	1	
128	64	32	16	8	4	2	1	Decimal numbers

Each box represents one bit.

Each bit in a byte can be expressed as a power of 2 (top row).

Also, the value of each bit can be expressed as a decimal number (bottom row).

Another way of describing a byte is to say that it is a binary number with eight place values.

For example, the decimal value of this byte is 16 (16 + 0 + 0 + 0):

0	0	0	1	0	0	0	0
128	64	32	16	8	4	2	1

The decimal value of this byte is 11:

0	0	0	0	1	0	1	1
128	64	32	16	8	4	2	1

The maximum value for any byte is 255:

1	1	1	1	1	1	1	1	
128	+64	+32	+16	+8	+4	+2	+1	= 255

Questions & Answers

What is the decimal value of this byte?

0	1	0	1	0	1	0	1
128	64	32	16	8	4	2	1

Answer: 85

We will be seeing a lot of these bit values when we use the VIC II graphics chip and the SID sound chip in Chapters 10 and 11.

Before we leave binary numbers we should discuss briefly how they are used with the logical operators AND, OR and NOT.

As we described in Chapter 3, most logical operators return a value of either TRUE or FALSE. For example, A < C is either TRUE or FALSE, depending on the values of A and C. However, AND, OR and NOT have more complex evaluation schemes that are not immediately obvious. These evaluations are an advanced programming topic, so we will only cover AND here, as we will need it in programs that use files or a joystick.

If you want to learn more, you should read a mathematics book on boolean algebra or an advanced programming book.

The AND evaluation is done by comparing two bytes, one bit at a time. Like this:

	128	64	32	16	8	4	2	1
First byte	1	0	0	1	0	1	0	1
Second byte	1	1	0	0	1	1	1	0
AND result	1	0	0	0	0	1	0	0

In this example, the first binary number is 149 and the second is 206; the result of ANDing them is 132. You can check this with a direct print statement, like this:

```
PRINT 149 AND 206
132
```

The AND expression can be used to see if one particular bit is "on" in a byte, that is, to see if it is not zero.

End-of-File Markers

At the end of any file, whether it is a program or a file with some other information, the computer puts an end-of-file marker. When the computer reads this end-of-file marker, it turns on one bit in the variable ST. (ST is a reserved variable name.) The bit turned on is the seventh, the one that represents decimal 64. So, if ST has any of these values:

```
0 1 0 0 0 0 0 0
1 1 0 1 0 0 1 0
1 1 1 1 1 1 1 1
```

then the end-of-file marker has been read. In fact, any value of ST in which the seventh bit is 1 indicates that the end-of-file marker has been read.

To test for end-of-file we can use the expression:

```
IF (ST AND 64) = THEN . . . .
```

The AND is evaluated like this:

```
ST        1  1  0  0  0  1  0  0
64        0  1  0  0  0  0  0  0
          ------------------------
ST AND 64 0  1  0  0  0  0  0  0
```

In this case, ST AND 64 does equal 64.

Try this with different values for ST. Notice that if the seventh bit is on, the result of the AND is always 64. If the seventh bit is off, the result is never 64.

In practice, the expression is written as

```
IF ST AND 64 THEN . . .
```

This statement implies that the result of ST AND 64 is to be compared to 64. We will use this statement in a joystick program in Chapter 9:

```
JS = 15 - (JS AND 15)
```

The operators OR and NOT have their own rules for evaluation, but we will not cover them in this book.

Computer Memory Organization

Now we will see how bytes are connected to the programs we write. A byte is the smallest group of bits that we need to handle all our information. The Commodore 64 has an eight bit processor, which means that it processes one byte of information at a time.

This is a good time to consider the importance of the number zero. In most situations when we encounter zero, it represents nothing or no value. This is not the case with computers. Zero does have a value, and it always represents a condition or status. You need to make a habit of remembering zero and always starting to count from zero.

Questions & Answers

If the maximum decimal value that can be represented in one byte is 255, and each value can represent one unique character of information, how many characters can one byte represent?

Answer: 256 - Don't forget zero can represent a character.

The Commodore 64 uses the fact that one byte can represent 256 unique characters to allow the user to include all the keyboard symbols in a program.

Appendix G shows the ASCII and CHR$ codes. All 256 numbers have been assigned to represent a number, letter or symbol. We can use this table with two BASIC functions that are already programmed into the Commodore 64 for us. To try this, we can use the direct PRINT command:

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

The computer will print the letter 'A' on the screen.
Now type:

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

The computer will print 65.
Type this one:

```
PRINT CHR$(122)
```

Questions & Answers

Which of these were displayed on the screen?

_____ (a) A heart.
_____ (b) The letter R.
_____ (c) A diamond.

Answer: (c)

There is another table in Appendix H which contains all the screen codes. This also has 256 entries, and it is used for graphics programs. We will be using this table in Chapters 9 and 10.

Now we know that information is stored in bytes. Another important fact concerning bytes in the computer memory is that they are arranged so that each byte has a unique 'address' or 'location.' The computer keeps track of the stored information by these addresses or locations. This means that we can instruct the computer what to store and where to store it.

A useful analogy to explain this system is the post office. When faced with a roomful of post office boxes, you know which one is yours by the number on the outside. Inside, there will be mail, or 'information,' for you. Your post office box number is the same everyday, but the information delivered to your box changes daily. No matter what is sent to your address, it will end up in your box and you will always know where to pick up your mail. The same thing happens with the computer. The user chooses a name rather than a number for the location of certain information, and the computer sets aside an address in

memory tied to this name. Whenever the name is used in the program, the computer uses the information stored at that location.

We have been using this concept already in our programs when we used variable names. However, we can also refer to the actual location in the computer memory where a piece of information is stored. We will be doing this when we work with the VIC II chip and the SID chip in Chapters 10 and 11.

Questions & Answers

A variable name refers to a specific _____ or _____ in the computer memory.

Answer: Location, address.

Random Access Memory and Read Only Memory

Before we leave this discussion of bytes and computer memory, we should mention the connection between machine language, locations in computer memory and BASIC. In Chapter 2 we discussed briefly the screen display when the computer is first turned on. Let's look at it again:

64K RAM SYSTEM 38911 BASIC BYTES FREE

64K. K is a measurement and 1 K = 1024 bytes.
So 64K = 65536 bytes.

RAM stands for Random Access memory. This memory available to the user. There is another kind of memory ROM, Read Only Memory, that is not available to the user. (BASIC is stored in ROM.)

38911 BASIC BYTES FREE tells us that with the BASIC language loaded into the memory there are actually 38911 bytes left for us to use.

BASIC is one of the ('higher-level') languages; others include FORTRAN, COBOL, LOGO and PASCAL. All of these languages need another step to "translate" them into the binary machine language, and they require either a compiler or an interpreter. BASIC is an example of a higher-level language that

requires an interpreter. The BASIC interpreter is part of the Read Only Memory, ROM, of the Commodore 64.

The Read Only Memory has three main parts:

- The BASIC interpreter, which analyzes our BASIC programs and is responsible for their execution.
- The screen editor, which controls the output to the screen and all the editing functions, including whether the characters are a direct command or a program to be passed to the interpreter.
- The kernal, which is a program to control the flow of data and make sure that things are done in the right order.

The screen editor and the kernal are already programmed into the computer and automatically start when the computer is turned on. They are part of the OPERATING SYSTEM. The operating system is the link between the programmer and the machine. For most of the time the user is unaware that they are there, and that they are operating.

The following table will give you some idea of how the Commodore 64 memory is organized.

Table 7.1

Commodore 64 Memory Map	
Address	**Description**
0 & 1	6510 Registers
2	Start Memory
up to 1023	Operating System Memory
1024 up to 2039	Screen Memory
2040 up to 2047	SPRITE pointers
2048 up to 40959	USER'S Memory for BASIC and Assembly Language Programs

(continued)

Commodore 64 Memory Map	
Address	**Description**
40960 up to 49151	BASIC Interpreter
49152 up to 53247	Special Programs RAM
53248 up to 53294	VIC-II (Graphics) Memory
54272 up to 55295	SID (Music) Registers
55296 up to 56296	Color RAM
56320 up to 57343	Input/Output Registers
57344 up to 65535	Operating System Kernal

Questions & Answers

True or false:

_____ (a) BASIC is a higher-level language.
_____ (b) The operating system programs are in RAM.
_____ (c) BASIC programs require an interpreter.
_____ (d) Most of the time the user is not aware that the operating system is operating.

Answer:

(a), (c) and (d) are true.
(b) False. They are in ROM.

SELF-CHECK

Remember, this test is to help you learn the material covered in this chapter. So, relax.

1. Write the decimal equivalents of these binary numbers:
   (a) 101
   (b) 0
   (c) 1100
   (d) 11
2. Write the binary equivalents of these decimal numbers:
   (a) 1
   (b) 15
   (c) 7
   (d) 4
3. A bit can only have the value _____ or _____.
4. How many bits are in a byte?
5. What is the decimal value of these bytes?
   (a) 0 1 0 0 0 0 1 0
   (b) 1 1 1 1 1 1 1 1
   (c) 0 0 0 0 1 1 1 1
6. What would be the binary content of a byte storing the decimal number 64?
7. True or false:
   (a) The operating system is stored in RAM.
   (b) The kernal controls the output to the screen.
   (c) The screen editor is part of the operating system.
   (d) BASIC is a higher-level language that requires a compiler.

Answers to SELF-CHECK

1. (a) 5
   (b) 0
   (c) 12
   (d) 3

2. (a) 1
   (b) 1111
   (c) 111
   (d) 100
3. zero, one
4. 8
5. (a) 66
   (b) 255
   (c) 15
6. 0 1 0 0 0 0 0 0
7. (a) False. The operating system is part of ROM.
   (b) True.
   (c) True.
   (d) False. BASIC is a higher-level language that requires an interpreter.

Suggestions for Further Exploration.

1. Check out some of the computer journals from your local library and browse through their articles on computer memory and organization.
2. Try counting stairs or stars in binary.
3. Convert all your friends' and relatives' ages to binary numbers.

If you have found this chapter rather difficult, don't despair! You will find yourself understanding this information when you start to use it later on in this book. You may want to come back and review this chapter again.

CHAPTER 8

Datafiles

We have written programs and saved them on disks or cassette tapes, but so far we have not discussed how to save data, or information, on disks or cassettes. There are times when we want to save data to use with more than one program, or on more than one occasion. When information is stored on a cassette or disk like this, it is called a DATAFILE. In this chapter, we will learn how to create and use a datafile.

By the end of this chapter you will be able to:

- Define a datafile and identify its uses.
- Understand the concepts of RECORDS and FIELDS.
- Use the BASIC statements OPEN and CLOSE with a datafile.
- Write programs to create a datafile and access a datafile.
- Understand how to add, delete and modify a datafile.
- Select information from a datafile.
- Sort information into numerical or alphabetical order.

Use of Datafiles

We use a data file for information that we want to keep and have available at any time. For instance, it would be useful to have the names, addresses, and phone numbers of our friends and business acquaintances available for use on the computer. We could add to the list, delete from the list or make changes when people move. Some of my young friends like to catalog their stamp or baseball card collections, and a bird-watching friend keeps a list of his yearly

'spots' on his computer. All of these examples are suitable for saving as a datafile. Also, we shall see that the concept of a datafile is important when we start drawing sprites, the fantastic graphics feature on the Commodore 64.

The data stored in a datafile can be very simple: the number of eggs laid by the family hens, the lap times for a member of the swimming team or the names of the students on the Holmes Junior High School Honor Roll. Each of these examples consists of lists of numbers or strings, each one containing one piece of information. We call each item a RECORD; each lap time or each name is one RECORD in the datafile.

The honor roll records might look like this:

D. Clark
M. Harris
B. Hernandez
D. Hernandez
J. Johnson

Fields

Usually it takes more than one piece of information to make a record. For instance, the coach would like the name of the team members and the time stored for each lap, or the school requires the name and grade stored for each honor student. In these examples, each record would contain two pieces of information. Each piece of information is called a FIELD.

Field # 1	Field # 2	
D. Clark	3.7	
M. Harris	4.0	
B. Hernandez	3.8	Records
D. Hernandez	3.7	
J. Johnson	4.0	

Here is an example using Sam and his spaghetti business. Sam is computerizing his employee records for his spaghetti sauce business. He has expanded his business so that he employs fifty people. For each person he wants to store:

Name
Address
Phone number
Social Security Number
Number of dependents
Weekly wage rate

In this case, Sam's datafile will have 50 records, one for each employee. Each record will have 6 fields.

Here is another example. The Math 13 class at the university has 96 students. The professor is going to give 10 weekly quizzes, a mid-term exam and a final exam. He has decided to store all this information as a datafile on his Commodore 64.

Questions & Answers

(a) How many records will be in the Math 13 datafile?
(b) How many fields will each record have?

Answer:
(a) 96, one for each student.
(b) 13, student name, 10 quizzes, 1 mid-term, 1 final exam.

Sequential Files

It is important to plan the format of the fields and records you will use so that it is easy to use them in a program.

There are two types of datafile: sequential and random. In a sequential file, the records are written or read from the file in sequence, starting from the beginning and accessing the records one after another. In a random file, the records can be written or read in any order, and you can access any record directly without reading through all the records that go before it. In this book, we will learn about sequential datafiles. Random datafiles are an advanced feature available with a 1541 disk drive, and you may want to learn about them at a later date.

If you are using a cassette, then the only type of datafile you can use is a

sequential file. As you may remember, in Chapter 1 we mentioned that a cassette recorder is a sequential device. The tape starts at the beginning and winds forward until the desired information is reached. It is not possible to access any location on the tape at random. Using a disk drive you can have both random and sequential datafiles. This chapter refers to cassette storage and disk storage. The sections for each will be clearly marked. You should skip the sections that do not apply to your system.

Questions & Answers

Match.

(1) Random files	(a) May be stored on a disk.
(2) Sequential files	(b) May be stored on a tape.
	(c) Records must be accessed in order.
	(d) Records may be accessed directly.

Answer: (1) a, d (2) a, b, c

Cassette Tape Datafiles

This section describes how to set up sequential files on a data cassette. Disk users should skip this section.

For our first datafile we will start with some very simple records. We will build a datafile to store the numbers 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

Questions & Answers

How many records will there be?

Answer: 10

How many fields in each record?

Answer: 1

The first step in creating a datafile is to tell the computer what the file is called, and where to find it. If it were a file in a filing cabinet, we would get a new manila folder and write the name of the file on it. The equivalent of this step for a data cassette is the OPEN statement. It looks like this:

OPEN, <file # >, <device # >, <number>, <file name>

OPEN	required BASIC statement.
file #	this may be any number 1 through 255, but it is useful to number files in a program in sequence: 1, 2, 3, etc.
device #	this will be 1 for the data cassette.
number	defines how the file will be used. 0 INPUT from the file to the program. 1 Output, write to the file and an end-of-file marker will be put at the end of the file. 2 Output, like 1 except an end-of-tape marker is put at the end of the file.
file name	this can be any string name to describe the file. The rules for naming programs apply to naming datafiles.

To make our datafile of numbers we will use:

```
10 OPEN 1,1,1,"NUMBERS"
        |  |  └─Write to file with end-of-file marker
        |  └─device number for cassette
        └─file number
```

Questions & Answers

Which of these statements would OPEN a data file called GRADES for output with an end-of-tape marker?

_____ (a) OPEN 1,1,1, "GRADES"
_____ (b) OPEN 1,1,0, "GRADES"
_____ (c) OPEN 2,1,2, "GRADES"
_____ (d) OPEN 2,2,1, "GRADES"

Answer:

(c) OPEN 2,1,2, "GRADES"

- 2 — file number
- 1 — device number for cassette
- 2 — end-of-tape marker

We can use a FOR-NEXT STEP loop to generate the data for our file, but first we will need a new BASIC statement to put it in the file:

PRINT # <file number>, <variable>, <variable>. . .

PRINT #	required BASIC statement.
file number	must match file number in OPEN statement.
variable	numeric or string variables may be used.

Notice where the commas are placed; it is very important to put a comma after PRINT # <file number>.

The FOR-NEXT loop for the number program will look like this:

```
20 FOR C = 10 TO 100 STEP 10
30 PRINT #1, C
40 NEXT C
```

Questions & Answers

What does the number 1 refer to in line 30?

Answer: It is the file number and is the same as the file number in the OPEN statement.

The final step in the program will be to close the datafile. To do that we use a CLOSE statement.

CLOSE <file number>

CLOSE	required BASIC statement.

file number — the number of the file to close. Must be the same as the file number in the OPEN statement.

So the last lines in the program will look like this:

```
50 CLOSE 1
60 END
```

When this program is run, the cassette follows the normal procedure of displaying the instruction to press RECORD and PLAY on the cassette player, and then the screen goes blank while the datafile is written. When the READY prompt appears, the program has finished, and the data file has been written on the tape.

Questions & Answers

The two statements that control access to a datafile are:

Answer: OPEN, CLOSE.

Now we need to retrieve the information from the datafile. How would you write an OPEN statement to input information from the file? (You may need to refer back to page 249.)

Answer: 10 OPEN 1,1,0, "NUMBERS"

We will need a loop to input the numbers from the datafile, and this time we will need an INPUT # statement, which looks like this:

INPUT # <file # >, <variable>, <variable>, ...

INPUT #	required BASIC statement. Notice that there is not a space between INPUT and #.
file #	must be the same file number as the OPEN statement.
variable	string or numeric variable to be used in the program.

Here is the loop, including the INPUT # statement and a PRINT statement to display the contents of the file.

```
20 FOR C = 1 TO 10
30 INPUT #1, X
40 PRINT X
50 NEXT C
```

Notice the variable name in the PRINT statement matches the variable name in the INPUT #! statement.

Questions & Answers

What will the final lines be?

Answer:

```
60 CLOSE 1
70 END
```

Be sure to rewind the tape before you run this program. Remember, on a cassette tape, it is the user's responsibility to keep track of where programs and datafiles are stored.

Disk Datafiles

This section refers to disk drives. If you do not have a disk drive, skip to the next section.

For our first datafile using a disk, we will start with some simple records. We will build a datafile to store the numbers 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

Questions & Answers

How many records will there be?

Answer: 10

How many fields in each record?

Answer: 1

The first step in creating a datafile is to tell the computer what the file is called and where to find it. If it were a file in a filing cabinet, we would get a new manila folder and write the name of the file on it. To do this operation we use the following statement:

OPEN <logical file # >, <device # >, <secondary address>,
"<drive # >:<file name>,<file type>,<mode>"

OPEN	required BASIC statement.
Logical file #	this may be any number 1 through 255. It is useful to number files in a program in sequence: 1, 2, 3, etc.
Device #	this will be 8 for a disk drive. If you are using two disk drives, then the device number for the second drive should be changed to 9. There is a program available from Commodore that will do this.
Secondary address	any number from 2 — 14. 0 and 1 are reserved and 15 is reserved for the command and error channel. (We used it in Chapter 3 to initialize a disk.)
Drive #	this will be 0 for the first drive. It is optional with only one drive.
File name	this can be any string name to describe the file. The rules for programs apply to naming datafiles.
File type	this can be: PRG for a program file. SEQ for a sequential data file. USR for a user file. REL for a relative file.
Mode	this is either READ or WRITE. (They may be abbreviated to R and W.)

To make our datafile of numbers, we will use:

```
10 OPEN 2,8,2"0:NUMBERS,SEQ,R"
```

Questions & Answers

Which of the following statements are true?

_____ (a) The disk drive device number is 8.
_____ (b) R stands for random in this statement.
_____ (c) 2 is used for the logical file number and the secondary address.
_____ (d) The 0 is optional, but it is a good idea to include it.

Answer: All are true except (b). The R stands for read, not random.

We can use a FOR-NEXT loop to generate the data for our file, but we will need a new BASIC statement to put it in the file:

PRINT # <file number>, <variable>, <variable>. . .

PRINT #	a required BASIC statement.
file number	must match the logical file number in the OPEN statement.
variable	numeric or string variables.

Notice where the commas are placed; it is very important to put a comma after PRINT # <file number>.

The FOR-NEXT loop for our program will look like this:

```
20 FOR C = 10 TO 100 STEP 10
30 PRINT #2,C
40 NEXT C
```

Questions & Answers

What does the number 2 refer to in line 30?

Answer: It is the logical file number and is the same as the logical file number in the OPEN statement.

The final step in the program will be to close the datafile. To do that, we use a CLOSE statement.

CLOSE <file number>

CLOSE	required BASIC statement.
file number	the same number as the logical file number in the OPEN statement.

So, the last lines in the program will look like this:

```
50 CLOSE 2
60 END
```

Questions & Answers

The two statements that control access to a datafile are:

Answer: OPEN, CLOSE.

Error Handling

In Chapter 2 when we discussed Disk Operating System (DOS) commands, there was a brief mention of the fact that DOS errors are not displayed on the screen. We will encounter DOS errors, so before we RUN any program that uses datafiles, we need to be able to identify and display them. To do this, first open the communications file at the beginning of the program:

```
5 OPEN 15,8,15
```

After the datafile has been opened in statement 10, we will call a subroutine to check the communication file for errors.

```
15 GOSUB 1000
```

The subroutine, starting at statement 1000, looks like this:

```
1000 REM **** ERROR CHECK *****
1010 INPUT#15, EN, ED$,TN,BN
1020 IF E = 0 THEN RETURN
1030 PRINT EN, ED$,TN,BN
1040 END
```

Four variables are input from file 15. The first variable, EN, is the error number. If EN is zero, then there is no error. If EN is not zero, then all four variables are displayed on the screen, the datafile is closed, and the execution of the program ends. The error number and the error description, ED$, help us debug the program. The track number, TN and the block number, BN, tell us where on the disk the error occurred.

WARNING

Call this error handling routine after every statement that opens a datafile.

Now we need to retrieve the information from the datafile. First OPEN the communication file:

5 OPEN 15,8,15

Questions & Answers

How would you write an OPEN statement to input information from the file? (You may need to refer back in this chapter to the section on OPEN statements.

Answer: 10 OPEN 2,8,2 "0:NUMBERS,SEQ,R"

Now call the error subroutine:

Answer: 15 GOSUB 1000

We will need a loop to input the numbers from the datafile, but this time we will need an INPUT # statement, which looks like this:

INPUT # <file # >, <variable>, <variable>, . . .

INPUT #	required BASIC statement. Notice there is not a space between INPUT and #.
file #	logical file number from OPEN statement.
variable	any string or numeric variable.

Here is the loop including the INPUT # statement and a PRINT statement to display the contents of the NUMBERS file.

```
20 FOR C = 1 TO 10
30 INPUT # 2,X
40 PRINT X
50 NEXT C
```

Questions & Answers

What will the final statements be?

Answer:

```
60 CLOSE 2
70 END
```

Add the error subroutine to the end of the program and RUN it.

Multiple Fields

The rest of this chapter will refer to both cassette and disk datafiles.

The numbers example used a record with only one field. Our next step will be to create a file, called ADDRESS, with records of more than one field. We will use name, address and phone number. Each record should have six fields:

Mr. Thomas Jones	(1st field)
720 E. 32nd Street	(2nd field)
Coopersville	(3rd field)
California	(4th field)
95606	(5th field)
(999) 999-9999	(6th field)

The Address File

We will break the address into separate fields so we can select records for our home town only, or so we can sort the records by zip code for a mass mailing.

Here are the field names we will use:

Name	N$
Street	SR$
City	C$
State	S$
Zip code	Z
Phone	P$

We will store the phone numbers using this format:

(999) 999-9999

Questions & Answers

Why can't the phone number be a numeric variable?

Answer: Parentheses and dashes are characters, not numbers, so the phone numbers must be stored as string variables.

First we need an OPEN statement for the communication file, 15, and to write the datafile "ADDRESS."

Questions & Answers

What will the OPEN statement for the datafile look like?

Answer:

Cassette files: `10 OPEN 1,1,1,"ADDRESS"`

Disk files: `10 OPEN 1,8,2, "0:ADDRESS,SEQ,W"`

Here are four sample addresses and phone numbers to write in the datafile:

Mr. and Mrs. R. Anderson 10 Beech Lane Sweetwater Idaho 53812 (408) 662-7930	Miss Jill Peoples 2044 J Street Apt 12B Dixon California 95616 (727) 753-2499
Dr. J. Doolittle 10 University Ave Suite 5 San Pablo New Mexico 86524 (514) 483-3826	Mr. Carl Jorgenson 124 West Acre Road Woodland Hills California 95825 (727) 686-1413

Here is the program, with a flexible loop, to read in the names, addresses and phone numbers. If we make the upper limit on the loop a variable, then the program can be used again.

```
10 OPEN 15,8,15
20 OPEN 2,8,2,"ADDRESS,SEQ,W"
30 GOSUB 1000
40 PRINT"THIS PROGRAM CREATES AN ADDRESS FILE"
50 INPUT" NAME ";N$
60 PRINT#2,N$
70 INPUT" STREET ";SR$
80 PRINT#2,SR$
90 INPUT" CITY ";C$
100 PRINT#2,C$
110 INPUT" STATE ";S$
120 PRINT#2,S$
130 INPUT" ZIP CODE ";Z
140 PRINT#2,Z
150 INPUT" PHONE NUMBER ";P$
160 PRINT#2,P$
165 N = N+1
170 PRINT"DO YOU HAVE ANOTHER ADDRESS (Y/N)"
180 GET A$ : IF A$ = "" THEN180
190 IF A$ = "Y" THEN30
200 CLOSE 2
210 PRINT N;" ADDRESSES HAVE BEEN WRITTEN IN THE FILE."
220 END
```

(continued)

```
1000 REM **** ERROR CHECK *****
1010 INPUT#15, EN, ED$,TN,BN
1020 IF E = 0 THEN RETURN
1030 PRINT EN, ED$,TN,BN
1040 END
```

Questions & Answers

What will be the answers to the following questions if we use the four sample addresses to test this program?

(a) What is N?
(b) How many records are there?
(c) How many fields are there in each record?
(d) How many data items will there be in the file?

Answer: (a) 4, (b) 4, (c) 6, (d) 24

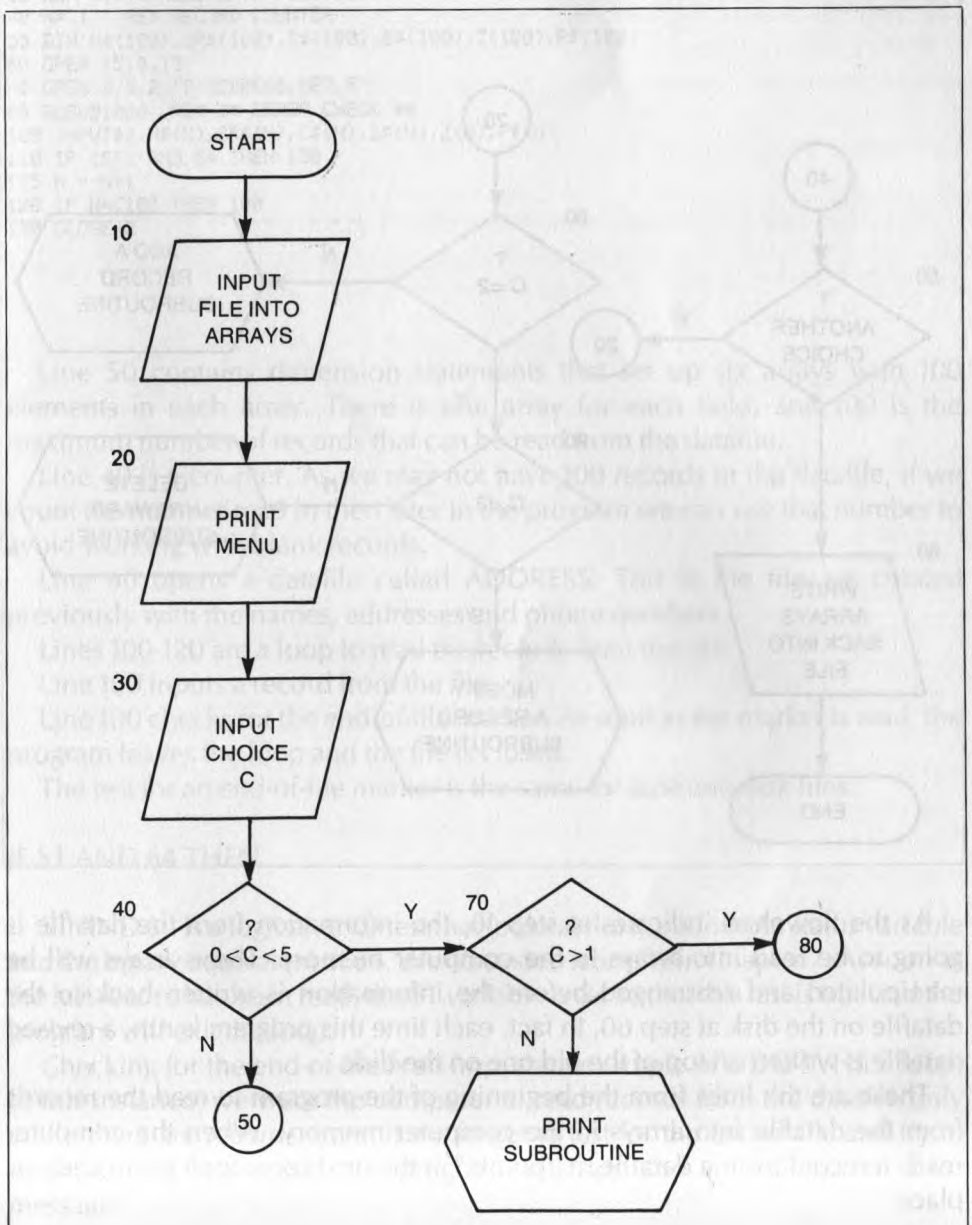
Using the ADDRESS File

Now that the ADDRESS file is created, there are several things we might want to do with it.

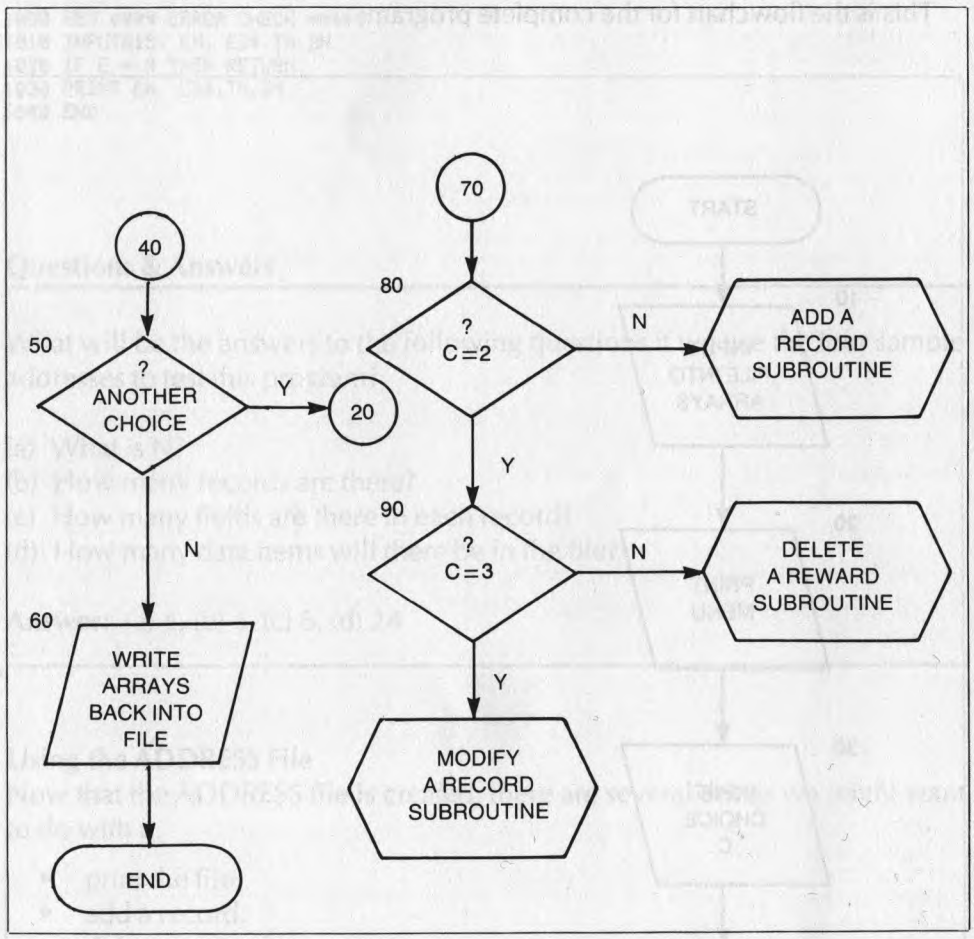
- print the file.
- add a record.
- delete a record.
- change a record.

We can write one program to cover all these options by using a main program with subroutines for the options. As always, we will start by analyzing, programming and testing the smaller modules before we put them together into one big program.

This is the flowchart for the complete program:



(continued)



As the flowchart indicates at step 10, the information from the datafile is going to be read into arrays in the computer memory. These arrays will be manipulated and rearranged before the information is written back to the datafile on the disk at step 60. In fact, each time this program is run, a revised datafile is written on top of the old one on the disk.

These are the lines from the beginning of the program to read the records from the datafile into arrays in the computer memory. When the computer reads a record from a datafile, it "points" at the next record, so it can keep its place.

```
10 REM NAME ADDRESS AND PHONE DATAFILE
20 REM **** MAIN PROGRAM ****
30 REM ***** READ DATA FILE ****
40 N= 1 : REM RECORD COUNTER
50 DIM N$(100),SR$(100),C$(100),S$(100),Z(100),P$(100)
60 OPEN 15,8,15
70 OPEN 2,8,2,"0:ADDRESS,SEQ,R"
80 GOSUB1000: REM ** ERROR CHECK **
100 INPUT#2,N$(N),SR$(N),C$(N),S$(N),Z(N),P$(N)
110 IF (ST) AND 64 THEN 130
115 N = N+1
120 IF N=<100 THEN 100
130 CLOSE 2
```

Line 50 contains dimension statements that set up six arrays with 100 elements in each array. There is one array for each field, and 100 is the maximum number of records that can be read from the datafile.

Line 40 is a counter. As we may not have 100 records in the datafile, if we count the number read in then later in the program we can use that number to avoid working with blank records.

Line 60 opens a datafile called ADDRESS. This is the file we created previously with the names, addresses and phone numbers.

Lines 100-120 are a loop to read the records from the file.

Line 100 inputs a record from the file.

Line 100 checks for the end-of-file marker. As soon as the marker is read, the program leaves the loop and the file is closed.

The test for an end-of-file marker is the same for tape and disk files:

IF ST AND 64 THEN . . . . .

ST AND 64 is a logical AND test that looks at the input/output status variable and the binary equivalent of 64. We discussed it briefly in Chapter 7. As soon as the last record has been read, the ST variable is set so that this test is true and the program will exit the loop.

Checking for the end-of-file is an important example of a DO-WHILE loop. In this instance, we want the computer to read records from the datafile only while there are records there. If we forget to test for the end-of-file and try to read past the last record, the program will stop and we will get an error message.

Line 120 limits the number of records input from the field, to 100. If there are more than a hundred addresses, this statement and the DIM statement must be changed to show the new limit.

To test the program so far, we can add a PRINT statement to check that the records have been read carefully.

```
105 PRINT N$(R); SR$(R); C$(R); S$(R); 2(R);P$(R)
```

If you are using a cassette tape, make sure that the tape is rewound to the beginning of the ADDRESS file before you run the program. Once the program has been tested, this PRINT statement should be removed.

Questions & Answers

What expression is used in an IF statement to test for the end-of-file?

Answer: ST AND 64

Now that the records have been read into arrays in the computer memory, we can start manipulating them.

The first step is to print a menu with the options. Here are the specifications for the menu:

Clear the screen.
Print the title of the program.
Print the options:

1. Print the file.
2. Add a record.
3. Delete a record.
4. Change a record.

Input the user's choice.
Call to the option subroutines.

The subroutines will start at lines 300, 500, 700 and 900.

Questions & Answers

Write a menu for your program before you check mine.

Answer: Your menu should look something like this:

```
140 REM ***** MAIN MENU ******
150 PRINT CHR$(147):PRINT:PRINT:PRINT
160 PRINTCHR$(5)"  NAME, ADDRESS AND PHONE NUMBER FILE"
170 PRINT:PRINT
180 PRINT:PRINTTAB(5)"THESE ARE YOUR OPTIONS:": PRINT
190 PRINT TAB(5) "1. PRINT THE FILE"
200 PRINT TAB(5) "2. ADD A RECORD "
210 PRINT TAB(5) "3. DELETE A RECORD "
220 PRINT TAB(5) "4. CHANGE A RECORD "
240 PRINT:PRINT
250 INPUT "YOUR CHOICE ";CH
260 ON CH GOSUB 380,550,620,710
```

Before we program the option subroutines, we will finish the main program.

When the program returns from a subroutine, we should ask if the user wants to do anything else to the file.

Here is a simple INPUT and test sequence to decide if the user wants another option:

```
270 PRINT: PRINT: PRINT
280 INPUT "WOULD YOU LIKE ANOTHER OPTION"; A$
290 IF LEFT$(A$,1) = "Y" THEN 130
```

Line 290 will return to print the menu if the answer is "Y" or begins with "Y."

If the user does not want to do anything else with the file, then the file will be revised. You will remember from Chapter 2 that we indicate a revision to an already existing file on a disk by putting an '@:' before the file name in the OPEN statement. Cassette files do not have this option.

If the user has finished with the program, the records are written out to the disk using a FOR-NEXT loop. Notice that this time we know exactly how many

records there are because we counted them when they were read into the arrays. We will add or subtract from this number when adding or deleting records.

Here is the complete loop.

```
270 PRINT:PRINT:PRINT
280 INPUT "WOULD YOU LIKE ANOTHER OPTION";A$
290 IF LEFT$(A$,1) = "Y" THEN150
300 OPEN 2,8,2,"@0:ADDRESS,SEQ,WRITE"
310 FOR C = 1 TO N
320 IF N$(C)= "DELETE" THEN 350
330 PRINT#2,N$(C);CHR$(13);SR$(C);CHR$(13);C$(C)
340 PRINT#2,S$(C);CHR$(13);Z(C);CHR$(13);P$(C)
350 NEXT C
360 CLOSE 2
370 END
```

Questions & Answers

What is the last instruction in programming a datafile?

Answer: CLOSE

The four subroutines for the options do not require the use of the datafiles, neither do they require any BASIC statements you haven't seen before. We have read the records from the datafile into arrays in memory and we will use these arrays in the subroutines. There is, of course, more than one way to program each of them. We will look briefly at one possible way of programming them.

Printing the File

Printing the file presents no problem other than deciding which format should be used to print the records. Are they to be printed on the screen or the printer? All on one line or in 'label' format? If they are labels, how many in a row? These questions need to be answered in order to write this subroutine.

I have chosen to print the records in label format in two columns on the screen. Here is a subroutine to do it:

```
380 REM ****** PRINT THE FILE *****
390 A= 1
400 PRINT CHR$(147)
410 FOR P=1 TO 4
420 PRINT TAB(2)N$(A) TAB(20) N$(A+1)
430 PRINT TAB(2)SR$(A)TAB(20) SR$(A+1)
440 PRINT TAB(2)C$(A) TAB(20) C$(A+1)
450 PRINT TAB(2)S$(A);Z(A) TAB(20) S$(A+1);Z(A+1)
460 PRINT TAB(2)P$(A) TAB(20) P$(A+1)
470 PRINT
480 A = A+2
490 IF A> N THEN 540
500 NEXT P
510 PRINT"ARE YOU READY TO CONTINUE?"
520 GET A$: IF A$ = "" THEN 520
530 IF A$ = "Y" THEN 400
540 RETURN
```

Questions & Answers

How many records are printed on the screen at one time?

Answer: 4

What does the variable do?

Answer: It counts the number of records that have been printed.

What does line 490 do?

Answer: It compares the number of records printed with the number stored in the arrays.

What will happen if the user answers "Y" to the question in line 510?

Answer: The program will print the next four records.

What happens if the user answers "NO" to the question in line 510?

Answer: The program returns to the main program.

Adding a Record

The subroutine to add a record to the file is also straightforward. This time our concern is with the size of the array. We do not want to increase the file size beyond 100, the current value of the dimension statement. If the addition will exceed the limit, then the programmer should be warned so that either the size of the arrays can be increased or some records deleted.

```
550 REM **** ADD A RECORD *****
560 N = N + 1
570 IF N>100 THEN PRINT"RECORD NOT ADDED": GOTO600
580 R=N: GOSUB 890
590 PRINT " TOTAL NUMBER OF RECORDS IS NOW ";N
600 PRINT " MAXIMUM FILE SIZE AT THE MOMENT IS 100 "
610 RETURN
```

We need to input the fields for the record to be added to the file. We could include the INPUT statements at this point in this subroutine. However, if we review the flowchart for this program, we see that we will need the same information when we want to change a record. It will shorten the program to code the INPUT statements as a subroutine and to call it when it is needed. Here is the subroutine to INPUT a record:

```
890 REM**** INPUT A RECORD ****
900 INPUT "NAME ";N$(R)
910 INPUT " STREET ";SR$(R)
920 INPUT " CITY "; C$(R)
930 INPUT " STATE ";S$(R)
```

(continued)

```
940 INPUT " ZIP CODE ";Z(R)
950 INPUT " PHONE "; P$(R)
960 RETURN
```

Questions & Answers

Which statement in the ADDITION subroutine calls the INPUT subroutine?

Answer: 580

Which statement in the ADDITION subroutine will be executed after the RETURN statement in the INPUT subroutine?

Answer: 590

Deleting a Record

Deleting records requires a different strategy. If we change the NAME field of the record to be deleted to "DELETE," we can then alter the main program so that when the file is written back to the tape or disk, all records with "DELETE" in the first field are skipped.

The subroutine to make the change in the NAME field looks like this:

```
620 REM ***** DELETE A RECORD ******
630 INPUT "NAME TO BE DELETED ";D$
640 FOR R = 1 TO N
650 IF N$(R) = D$ THEN 680
660 NEXT R
670 IF R > N THEN PRINT " NAME NOT FOUND": GOTO 700
680 N$(R) = "DELETE"
690 PRINT D$; " IS DELETED"
700 RETURN
```

Questions & Answers

What is the variable D$ storing?

Answer: The name in field 1 of the record to be deleted.

What is line 850 testing?

Answer: To see if the record was found in the array N$

To delete option requires that we make an addition to the main program. In the section that rewrites the datafile, we need a line to test for deletions.

330 IF N$(R) = "DELETE" THEN 350

Line 350 is the NEXT statement, so the program does not write any records in the datafile that have "DELETE" in the name field.

Changing a Record

The subroutine for changing a record is similar to the delete subroutine. The record to be changed is identified by the name field and if the record is not found, a message is printed to the user. Once the record is found, the new information is input using the INPUT subroutine starting at line 890.

Questions & Answers

I think you can write the subroutine without any help from me.

Answer: If you would like to check your subroutine, here is my version:

```
710 REM ****** CHANGE A RECORD *****
720 INPUT " NAME OF RECORD TO BE CHANGED ";D$
730 FOR R = 1 TO N
740 IF N$(R) = D$ THEN 770
750 NEXT R
760 PRINT" NO RECORD WITH THAT NAME " : RETURN
770 GOSUB 890
780 RETURN
```

Testing the Program

To test this program, we need to run every possibility for each option. For example:

- Answering YES and NO to the question in the print subroutine.
- Adding records and trying to add records beyond the array size. (Try lowering the array size to10 to do this one.)
- Asking to delete a record that isn't in the file, as well as one that is there.
- Changing records that are in the file, as well as trying to change one that isn't.

If all these possibilities work, then, your datafile program works! Now that you have seen one datafile program developed, you can adapt the technique for other datafiles. Here is the complete program listing.

```
10 REM NAME ADDRESS AND PHONE DATAFILE
20 REM **** MAIN PROGRAM ****
30 REM ***** READ DATA FILE *****
40 N= 1 : REM RECORD COUNTER
50 DIM N$(100),SR$(100),C$(100),S$(100),Z(100),P$(100)
60 OPEN 15,8,15
70 OPEN 2,8,2,"0:ADDRESS,SEQ,R"
80 GOSUB1000: REM ** ERROR CHECK **
100 INPUT#2,N$(N),SR$(N),C$(N),S$(N),Z(N),P$(N)
110 IF (ST) AND 64 THEN 130
115 N = N+1
120 IF N=<100 THEN 100
130 CLOSE 2
140 REM ***** MAIN MENU ******
150 PRINT CHR$(147):PRINT:PRINT:PRINT
160 PRINTCHR$(5)"  NAME, ADDRESS AND PHONE NUMBER FILE"
170 PRINT:PRINT
180 PRINT:PRINTTAB(5)"THESE ARE YOUR OPTIONS:": PRINT
190 PRINT TAB(5) "1. PRINT THE FILE"
200 PRINT TAB(5) "2. ADD A RECORD "
210 PRINT TAB(5) "3. DELETE A RECORD "
220 PRINT TAB(5) "4. CHANGE A RECORD "
240 PRINT:PRINT
250 INPUT "YOUR CHOICE ";CH
260 ON CH GOSUB 380,550,620,710
270 PRINT:PRINT:PRINT
280 INPUT "WOULD YOU LIKE ANOTHER OPTION";A$
290 IF LEFT$(A$,1) = "Y" THEN150
300 OPEN 2,8,2,"@0:ADDRESS,SEQ,WRITE"
310 FOR C = 1 TO N
320 IF N$(C)= "DELETE" THEN 350
330 PRINT#2,N$(C);CHR$(13);SR$(C);CHR$(13);C$(C)
340 PRINT#2,S$(C);CHR$(13);Z(C);CHR$(13);P$(C)
350 NEXT C
360 CLOSE 2
```

(continued)

```
370 END
380 REM ****** PRINT THE FILE *****
390 A= 1
400 PRINT CHR$(147)
410 FOR P=1 TO 4
420 PRINT TAB(2)N$(A) TAB(20) N$(A+1)
430 PRINT TAB(2)SR$(A)TAB(20) SR$(A+1)
440 PRINT TAB(2)C$(A) TAB(20) C$(A+1)
450 PRINT TAB(2)S$(A);Z(A) TAB(20) S$(A+1);Z(A+1)
460 PRINT TAB(2)P$(A) TAB(20) P$(A+1)
470 PRINT
480 A = A+2
490 IF A> N THEN 540
500 NEXT P
510 PRINT"ARE YOU READY TO CONTINUE?"
520 GET A$: IF A$ = "" THEN 520
530 IF A$ = "Y" THEN 400
540 RETURN
550 REM **** ADD A RECORD *****
560 N = N + 1
570 IF N>100 THEN PRINT"RECORD NOT ADDED": GOTO600
580 R=N: GOSUB 890
590 PRINT " TOTAL NUMBER OF RECORDS IS NOW ";N
600 PRINT " MAXIMUM FILE SIZE AT THE MOMENT IS 100 "
610 RETURN
620 REM ***** DELETE A RECORD ******
630 INPUT "NAME TO BE DELETED ";D$
640 FOR R = 1 TO N
650 IF N$(R) = D$ THEN 680
660 NEXT R
670 IF R > N THEN PRINT " NAME NOT FOUND": GOTO 700
680 N$(R) = "DELETE"
690 PRINT D$; " IS DELETED"
700 RETURN
710 REM ****** CHANGE A RECORD *****
720 INPUT " NAME OF RECORD TO BE CHANGED ";D$
730 FOR R = 1 TO N
740 IF N$(R) = D$ THEN 770
750 NEXT R
760 PRINT" NO RECORD WITH THAT NAME " : RETURN
770 GOSUB 890
780 RETURN
890 REM**** INPUT A RECORD ****
900 INPUT "NAME ";N$(R)
910 INPUT " STREET ";SR$(R)
920 INPUT " CITY "; C$(R)
930 INPUT " STATE ";S$(R)
940 INPUT " ZIP CODE ";Z(R)
950 INPUT " PHONE "; P$(R)
960 RETURN
1000 REM **** ERROR CHECK ****
1010 INPUT#15, EN,ED$,TN,BN
1020 IF E = 0 THEN RETURN
1030 PRINT EN;ED$;TN;BN
1040 CLOSE 2
1050 END
```

Selecting Records

There is one other operation that is often used with datafiles that we have not yet discussed. This is selecting a group of records from a file. For example, we might want to send a mailing to all the addresses with a certain zip code. This is not a difficult task, and if we are making a selection based on just one criteria, the zip code, we could do it as the records are read in from the datafile. On the next page is the flowchart for a program to do this using our datafile ADDRESS.

Questions & Answers

Can you write the program?

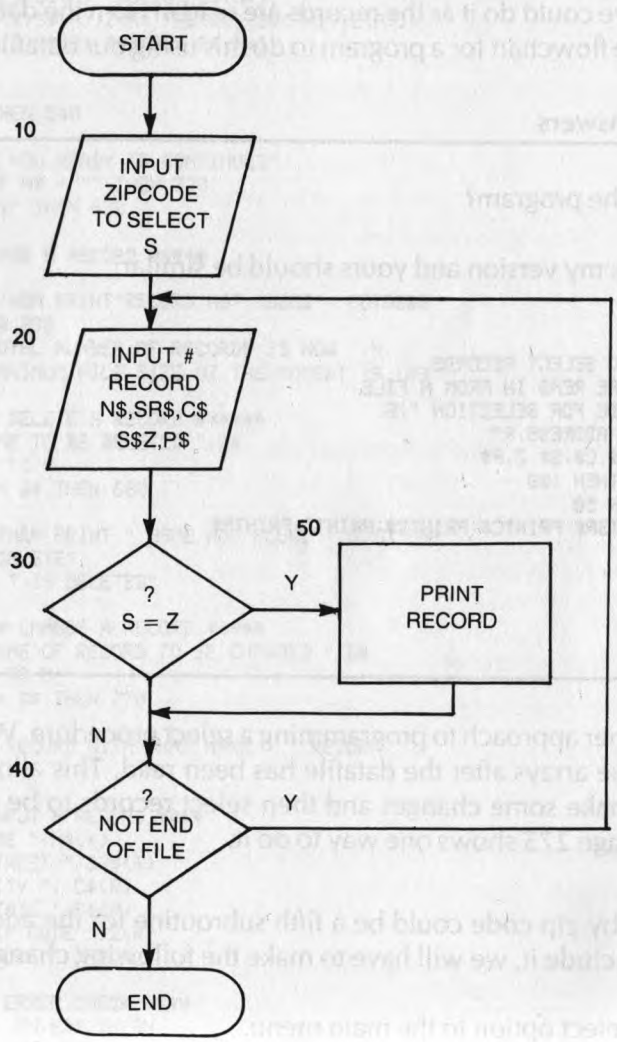
Answer: This is my version and yours should be similar:

```
10 REM PROGRAM TO SELECT RECORDS
20 REM AS THEY ARE READ IN FROM A FILE
30 INPUT "ZIP CODE FOR SELECTION ";S
40 OPEN 2,8,2,"0:ADDRESS,R"
50 INPUT#2,N$,SR$,C$,S$,Z,P$
60 IF ST AND 64 THEN 100
70 IF Z <> S THEN 50
80 PRINT N$:PRINTSR$:PRINTC$:PRINTS$:PRINTZ:PRINTP$
90 GOTO 50
100 CLOSE 2
110 END
```

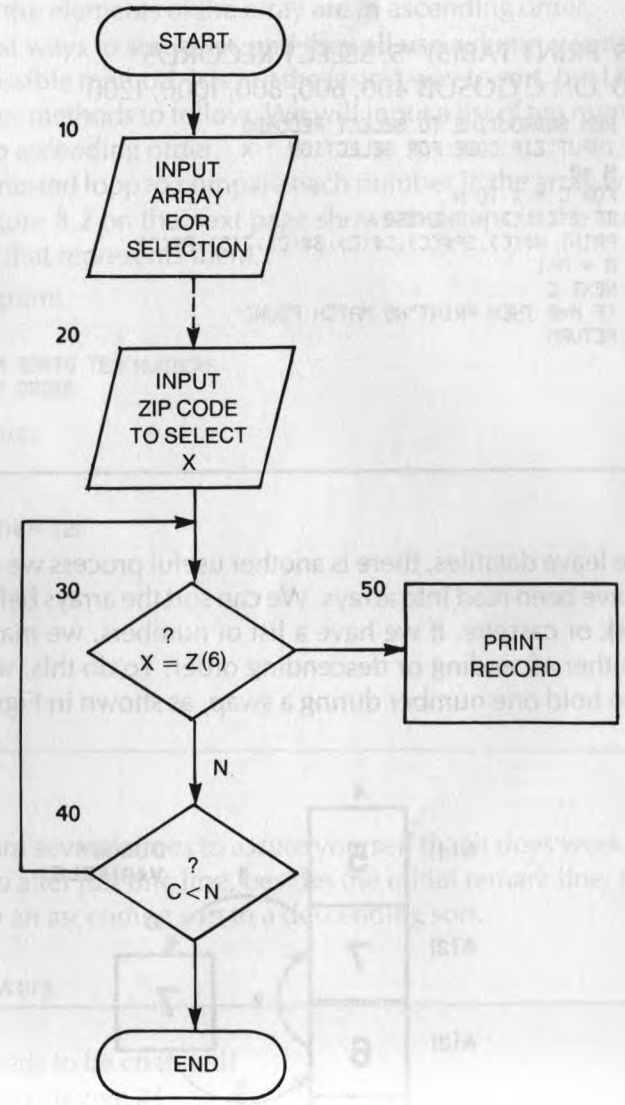
Here is another approach to programming a select procedure. We can select records from the arrays after the datafile has been read. This allows a user to read the file, make some changes and then select records to be printed. The flowchart on page 275 shows one way to do it.

A selection by zip code could be a fifth subroutine for the address datafile program. To include it, we will have to make the following changes:

1. Add the select option to the main menu.
2. Adjust the case statement in line 230 to include a branch to this subroutine, starting at line 1200.
3. Write the subroutine, starting at line 1200.



Flowchart for program to select records as they are read in from a datafile.



Flowchart for program to select records from a datafile after they have been read into an array in the computer memory.

Questions & Answers

Can you make the additions to the address program?

Answer:

(a) 205 PRINT TAB(5) "5. SELECT RECORDS"

(b) 230 ON C GOSUB 400, 600, 800, 1000, 1200

(c)
```
790 REM SUBROUTINE TO SELECT RECORDS
800 INPUT"ZIP CODE FOR SELECTION ";X
810 M =0
820 FOR C = 1 TO N
830 IF Z(C) <> X THEN860
840 PRINT N$(C),SR$(C),C$(C),S$(C),Z(C),P$(C)
850 M = M+1
860 NEXT C
870 IF M=0 THEN PRINT"NO MATCH FOUND"
880 RETURN
```

Sorting

Before we leave datafiles, there is another useful process we can perform if our records have been read into arrays. We can sort the arrays before we store them on the disk or cassette. If we have a list of numbers, we may want to arrange them in either ascending or descending order. To do this, we need a dummy variable to hold one number during a swap, as shown in Figure 8-1.

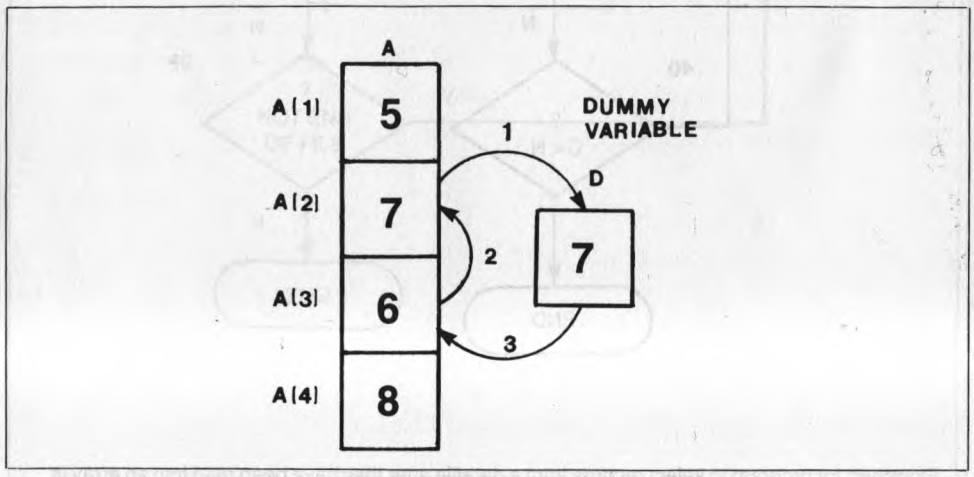


Figure 8.1 Diagram of the steps during a "swap" using a dummy variable.

The second and third items in the list must change places. The arrows show the moves that have to be made. Move 1: put the number in A(2) in the dummy variable P, where it will be safe while we make the next move. Move 2: put the number in A(3) into A(2). Move 3: put the numbers held in the dummy variable into A(3). Now all the elements of the array are in ascending order.

There are several ways to sort a list, and they all use a dummy variable. We will look at one possible method. It is not the fastest way to sort, but I think that it is one of the easier methods to follow. We will input a list of ten numbers and then sort them into ascending order.

This sort uses a nested loop to compare each number in the array with every other number. Figure 8.2 on the next page shows the steps in the bubble sort and the flowchart that represents them.

Here is the program:

```
10 REM THIS PROGRAM SORTS TEN NUMBERS
20 REM IN ASCENDING ORDER
30 FOR C = 1 TO 10
40 INPUT "NUMBER";N(C)
50 NEXT C
60 FOR I = 1 TO 9
70 FOR J = I TO 10
80 IF N(I) < N(J) THEN 120
90 D = N(J)
100 N(J) = N(I)
110 N(I) = D
120 NEXT J
130 NEXT I
140 FOR C = 1 TO 10
150 PRINT N(C)
160 NEXT C
```

Run this program several times to assure yourself that it does work.

We will need to alter just one line, besides the initial remark line, to change this program from an ascending sort to a descending sort.

Questions & Answers

(a) Which line needs to be changed?
(b) How would you change it?

Answer:

(a) line 80
(b) 80 IF N(J) > N(I) THEN 120

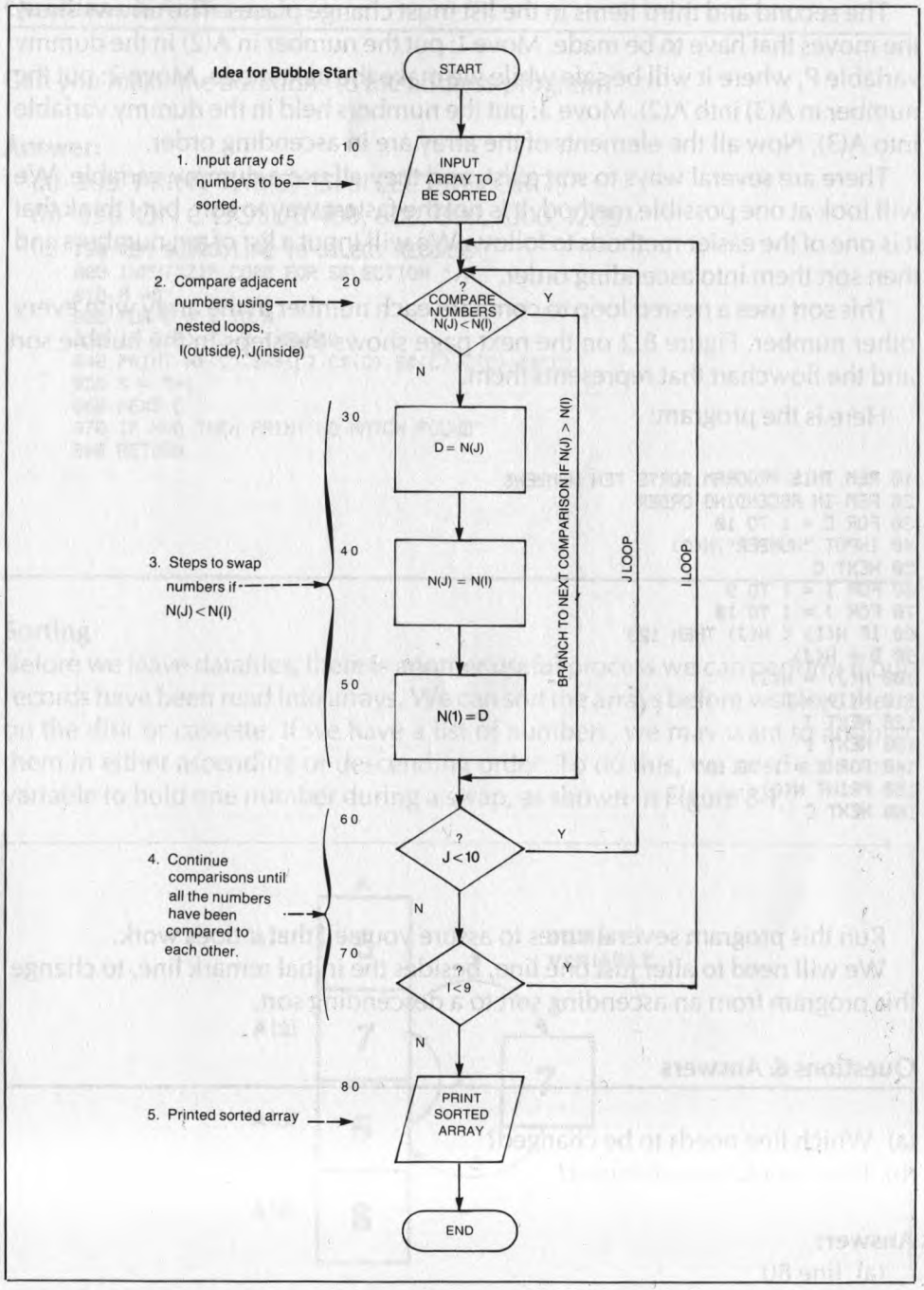


Figure 8.2 Flowchart and steps for a SORT routine.

It is possible to sort lists of alphabetical information into alphabetical order. Here is a list of names:

ADAMS
JONES
JOHNSON
SMITH
ADAMSON
SMALL
BROWN

We will use the same logic we used for the numerical sort, except that this time we are going to compare the ASCII code for each letter in the name. As soon as we find that a letter in the second name is "higher" than the letter in the corresponding position in the first name, we will swap the names using the dummy variable D$. This is the listing for the alphabetic sort program:

```
10 REM THIS PROGRAM SORTS TEN NAMES
20 REM IN ASCENDING ORDER
30 FOR C = 1 TO 10
40 INPUT "NAME";N$(C)
50 NEXT C
60 FOR I = 1 TO 9
70 FOR J = I TO 10
80 IF N$(I) < N$(J) THEN 120
90 D$= N$(J)
100 N$(J) = N$(I)
110 N$(I) = D$
120 NEXT J
130 NEXT I
140 FOR C = 1 TO 10
150 PRINT N$(C)
160 NEXT C
```

Try this program out using your own lists of test data.

We will add a numeric sort routine to our address datafile program which will automatically sort the records by zip code before they are written to the disk or cassette.

The complete listing for the program as it is currently written is on page 271.

Questions & Answers

Where would you put the GOSUB statement to call the sort subroutine?

Answer:
Between lines 290 and 300
295 GOSUB 1400

We will have to make some changes to the numeric sort program so that it sorts more than ten records and sorts on the zip code. We also need to keep the entire record intact, so if a swap is needed, in addition to swapping the zip code field, we must also swap all the other fields in the record at the same time.

Here is the listing for the subroutine.

```
1200 REM****SORT SUBROUTINE****
1210 FOR I = 1 TO N-1
1220 FOR J = I TO N
1230 IF Z(J) < Z(I) THEN 1470
1240 D=Z(J):D1$=N$(J):D2$=SR$(J):D3$=C$(J):
     D4$=S$(J):D5$=P$(J)
1250 Z(J)=Z(I):N$(J)=N$(I):SR$(J)=SR$(I):C$(J)=C$(I):
     S$(J)=S$(I):P$(J)=P$(I)
1260 Z(I)=D:N$(I)=D1$:SR$(I)=D2$:C$(I)=D3$:
     S$(I)=D4$:P$(I)=D5$
1270 NEXT J
1280 NEXT I
1290 RETURN
```

Lines 1240 through 1260 are swapping records. Notice that there are six dummy variables, one for each field in a record, and all fields are moved to a new position in the array.

Questions & Answers

What is stored in the variable N in lines 1210 and 1220?

Answer: The number of records in the arrays.

Datafiles are used for storing many different types of data. You will find that the programs developed in this chapter can be easily modified for use with any data you have.

Before you continue, please take the Self-Check.

SELF-CHECK

Please complete this short Self-Check before you file it away!

1. When searching for a record, _____datafiles start at the beginning of the file and do not go directly to the record.
2. The Browns are making a datafile for all their insured possessions. So far, they have a list of seventeen items. For each item, they want to store a brief description, (thirty characters), its value, the date it was purchased and the name of the insurance company.
   (a) How many records do the Browns currently have?
   (b) How many fields are in each record?
3. The Browns call their datafile 'INSURANCE.'
   (a) Write an OPEN statement to write records to the file for either a cassette or a disk.
   (b) Write an OPEN statement to input information from the file for either a cassette or a disk.
4. Which of these statements is used in a program at the end of a datafile operation?
   (a) STOP
   (b) END
   (c) CLOSE
5. The 29 second-graders at Birch Lane Elementary School have just taken a spelling test. Their teacher has a list with their names and their score on the tests. Write a program to sort this information and print out a list with the child with the highest score first on the list.

Answers to SELF-CHECK

1. Sequential.
2. (a) 17
   (b) 4

3. (a) Cassette: 10 OPEN 1,1,1,"INSURANCE"
   Disk: 10 OPEN 2,8,2,"0:INSURANCE,W"

   (b) Cassette: 10 OPEN 1,1,0,"INSURANCE"
   Disk: 10 OPEN 2,8,2,"0:INSURANCE,R"

4. (c)

5. This is my program and yours should be similar:

```
10 DIM N$(29), G(29)
20 FOR C = 1 TO 29
30 READ N$(C), G(C)
40 NEXT C
50 FOR C = 1 TO 28
60 FOR D = C TO 29
70 IF G(D) < G(C) THEN 110
80 DN$ = N$(D) : DG = G(D)
90 N$(D) = N$(C) : G(D) = G(C)
100 N$(C) = DN$ : G(C) = DG
110 NEXT D
120 NEXT C
130 FOR C = 1 TO 29
140 PRINT N$(C),G(C)
150 NEXT C
160 DATA <name 1>, <score 1>, . . . . .
```

Suggestions for Further Exploration

1. Look at some of the information you deal with everday. Can you identify the records and the fields in each record? How do you think the utility company or phone company stores the record for your monthly bill?
2. Try creating your own datafiles for your household, business or hobbies.
3. How would you use a datafile to create a mailing list for your church or volunteer organization?

CHAPTER 9

Color and Graphics

We began to use the color graphics capabilities of the Commodore 64 when we explored the keyboard in Chapter 1, but there is more, much more, to color and graphics.

Graphics in this book is going to be taken to mean any form of organized output, whether it is a "real" picture or a text picture. This is a wide definition, but the possibilities for graphics on the Commodore 64 are extensive and range from the very simple to the very sophisticated.

We are going to start with the things we already know and work toward some of the more complicated procedures. The graphics options on the Commodore 64 are controlled by the Video Interface Chip, VIC, and they can be divided into three modes:

- The standard 40 column by 25 line display.
- A high resolution 320 by 200 dot display.
- Small moveable objects called sprites.

In this chapter, we will look at the first of these modes: sprites will be covered in Chapter 11. They are truly fascinating and deserve a chapter to themselves. We will not cover the high resolution mode in this book. That is left for you to explore later on.

By the end of this chapter you will be able to:

- Write a graphics program, including reverse graphics from the keyboard.
- Animate a program using the cursor keys.
- Use the BASIC statement POKE.
- Change the screen and background colors.

- Use screen maps and character codes.
- Use the BASIC statement PEEK.
- Use the BASIC statement GET to animate a program.

Color from the Keyboard

This section is a review of creating pictures using color and symbols directly from the keyboard. We explored the keyboard in Chapter 1 and discovered the following information:

- SHIFT will print the symbol printed on the right-hand side on the front of the keys.
- The Commodore key will print the symbol printed on the left-hand side on the front of the keys.
- CTRL and the keys 1 through 8 will change the color of the print to the color printed on the front of the keys.
- The Commodore key and the keys 1 through 18 will give a further eight colors for a total of sixteen colors. A table with all sixteen colors and their keystrokes is given in Chapter 1.

We used these facts in Chapter 3 to create a multi-colored birthday cake. In the following statements I am going to enclose the keys to use inside <and> like this:

<CTRL 3> means use the CTRL key and the number 3 key. <4 SPACES> means press the space bar four times. Here is another example to print the four suits of cards in the correct colors.

```
10 PRINT "<CTRL 3><SHIFT S> <CTRL 1><SHIFT A>
<CTRL 3><SHIFT Z><CTRL 1><SHIFT X>"
```

I think that the the club sign, <SHIFT X>, looks like a tree or a flower.

Questions & Answers

Can you write the statements to do the following?

(a) Print two green trees.
(b) Print four purple flowers.
(c) Print three cyan flowers, one green tree and two orange flowers.

Answer:

(a) 10 PRINT "<CTRL 6><SHIFT X><SHIFT X>"
(b) 10 PRINT "<CTRL 5><SHIFT X><SHIFT X><SHIFT X>
<SHIFT X>"
(c) 10 PRINT "<CTRL 4><SHIFT X><SHIFT X><SHIFT X>
<Commodore 1><SHIFT X><CTRL 8><SHIFT X><SHIFT X>"

In Chapter 1 we also discovered that the CTRL key and key 9 would reverse the foreground and background colors, while CTRL and key 0 would restore them to normal. This was called ReVerSE ON and ReVerSe OFF. These keys are very useful for making solid shapes.

Try this program:

```
10 PRINT "<CTRL 9><CTRL3><4 SPACES>"
20 PRINT "<CTRL9><4 SPACES>"
30 PRINT "<CTRL 9><4 SPACES>"
40 PRINT "<CTRL 9>< 4 SPACES>"
```

Statement 10 does three things. The reverse on <CTRL 9> switches the foreground and background colors: <CTRL 3> changes the print color to red, then four spaces are printed. Notice that while the print color needs to be changed only once to remain that way for the program, the reverse print option must be included in each statement.

We would expect this program to draw a square, but it does not. This is because each individual print position is not square. It is a little taller than it is wide. So this shape needs to be considered if you want to draw a true square.

Questions & Answers

This next program will print bars of color down the screen. I have filled in the first statement for the black bar; can you complete the statements for the other seven bars?

```
10 PRINT"<SHIFT CLR/HOME>"
20 FOR C = 1 TO 20
30 PRINT"<CTRL 1><CTRL 9><4 SPACES>";
40
50
```

```
60
70
80
90
100
110 PRINT
120 NEXT C
130 END
```

Answer:

This is the complete program. Do not forget the semicolon at the end of each line, 30 through 110.

```
10 PRINT "<SHIFT CLR/HOME>"
20 FOR C = 1 TO 20
30 PRINT "<CTRL 1><CTRL 9><4 SPACES>";
40 PRINT "<CTRL 2><CTRL 9><4 SPACES>";
50 PRINT "<CTRL 3><CTRL 9><4 SPACES>";
60 PRINT "<CTRL 4><CTRL 9><4 SPACES>";
70 PRINT "<CTRL 5><CTRL 9><4 SPACES>";
80 PRINT "<CTRL 6><CTRL 9><4 SPACES>";
90 PRINT "<CTRL 7><CTRL 9><4 SPACES>";
100 PRINT "<CTRL 8><CTRL 9><4 SPACES>";
110 PRINT
120 NEXT C
130 END
```

We can also animate our pictures created directly from the keyboard by including the cursor keys inside the quotation marks. I call this program "My Heart Has Wings."

```
10 PRINT"<SHIFT CLR/HOME>"
20 FOR C = 1 TO 20
30 PRINT "<CRSR RIGHT><CTRL 2><Commodore *>
   <CTRL 3><SHIFT S><CTRL 2><SHIFT    ><3 CRSR LEFT>";
40 FOR T = 1 TO 100 : NEXT T
50 PRINT "<SHIFT    ><CTRL 3><SHIFT S><CTRL 2>
   <Commodore *><3 CRSR LEFT>";
```

```
60 FOR T = 1 TO 100 : NEXT T
70 PRINT"<3 SPACES><3 CRSR LEFT>";
80 FOR T = 1 TO 100 : NEXT T
90 NEXT C
100 END
```

Questions & Answers

Look at line 30. Which cursor control would we have to put as the second instruction in the print statement on line 30 to make the heart fly diagonally down and across the screen?

(a) Cursor right
(b) Cursor left
(c) Cursor up
(d) Cursor down

Answer:

(d) will make the heart fly diagonally down and across the screen.
(a) will move it to the right.
(b) will move it to the left; it will also take it off the screen.
(c) will make the heart fly diagonally up the screen, although it is already at the top of the screen, so we do not see the effect.

CHR$ Codes

We touched on one other graphics feature in Chapter 7 when we discussed the ASCII and CHR$ codes show in Appendix G. These codes may be used in the program to replace the quote mode and the characters from the keyboard.

There are 256 codes, which is enough to assign a code number to every character in the set, the reverse form of that character, the cursor controls, colors and other screen functions, such as RETURN and CLR HOME.

Using the information in Appendix G we can print a purple heart using this statement:

```
10 PRINT CHR$ (156), CHR$ (115)
```

One of the most frequently used codes is CHR$(147) which clears the screen. The next time the screen is messy, try this direct command:

```
PRINT CHR$(147)
```

The insert and delete functions can also be included in a program using the character codes. The following program uses the delete function, CHR$(20).

```
10 PRINT CHR$(147)
20 PRINT "WHAT AN AWFUL";
30 FOR C = 1 TO 500: NEXT C
40 PRINT CHR$(20); CHR$(20); CHR$(20); CHR$(20); "DORABLE DRESS!"
```

Questions & Answers

Now look at this short program:

```
10 PRINT CHR$(147)
20 PRINT CHR$(18); CHR$(5); CHR$(42); CHR$(42); CHR$(42)
```

(a) What does statement 10 do?
(b) What does CHR$(18) do in statement 20?

Answer:
(a) It clears the screen.
(b) It turns on reverse print.

Remember: the ASC function is the OPPOSITE of the CHR$ function: So:

```
PRINT ASC(A)
```

will print 65 on the screen.

We used the ASC function when we sorted alphabetic information in Chapter 8.

POKE Statements

Everything we have tried in this chapter has had a fixed position on the screen. We have been printing from the top of the screen and moving down one line at a time. It is possible to select any position on the screen and change it. To do this we use the BASIC statement POKE. It looks like this:

#POKE <location>, <value>

- POKE is the BASIC statement.
- <location> is a specific location memory. In Chapter 7 when we discussed bytes we learned that each byte has a known address or location in the computer memory.
- <value> will be a whole number between 0 and 255. This is the "content" of the byte. The bits at the location specified will be turned on to represent this value.

POKE can be used to change the content of any location on computer memory, but in this chapter we are interested only in the locations associated with the VIC II chip. The VIC II chip controls the color and graphics on the Commodore 64. The locations for the VIC II chip are shown in Appendix I.

We will start using the POKE statement with two locations: one controls the color of the screen and the other controls the color of the border.

Border location 53280
Screen location 53281

We can change the color of the screen and border to any of the sixteen colors available on the Commodore 64. Notice that the code for the colors to be used in the POKE statement is one less than the number on the keys.

0 Black	8 Orange
1 White	9 Brown
2 Red	10 Light Red
3 Cyan	11 Gray 1
4 Purple	12 Gray 2
5 Green	13 Light Green
6 Blue	14 Light Blue
7 Yellow	15 Gray 3

To turn the screen white, use this direct statement:

```
POKE 53281,1
```

Now turn the border black with this direct statement:

```
POKE 53280,0
```

Here is a program that will display all the possible combinations of screen and border colors.

```
10 PRINT CHR$(147)
20 FOR B = 0 TO 15
30 POKE 53280,B
40 FOR S = 0 TO 15
50 POKE 53281,S
60 FOR P = 1 TO 200: NEXT P
70 NEXT S
80 NEXT B
90 END
```

The border color changes in the loop, starting with statement 20. The screen color changes in the loop, starting in statement 40. The loop in statement 60 is an empty loop to create a pause so we can see each change for a few seconds before the next one comes.

Questions & Answers

What will be the screen and border colors when the B and S have the following values:

(a) B = 3, S = 10
(b) B = 10, S = 3
(c) B = 7, S = 7

Answer:

(a) Light red screen with a cyan border.
(b) Cyan screen with a light red border.
(c) Yellow screen with a yellow border.

Memory Maps

Not only can we use POKE to change the color of the whole screen or border, we can use POKE to place any character of any color anywhere on the screen.

Figure 9-1 on the next two pages shows the two memory maps the Commodore 64 uses to do this: one for storing the character and one for the color.

The screen is divided in 25 rows and 40 columns, giving 1000 positions on the screen. Each map has 1000 locations. The color map starts with location 55296 and goes through 56295. These locations are used with the color codes we saw in the last section to change the screen and border color.

The screen memory map starts at locations 1024 and has location 2023 as its final address. It is used with the codes shown in Appendix H.

To show how these maps work we will start by putting a heart in the first location.

```
POKE 1024, 83
```

Nothing will happen until we use another POKE for the color in the same location.

```
POKE 55296,2
```

Now you should see a red heart in the top left-hand corner of the screen.

Each position requires two POKE statements, one for color and one for the character.

Also notice that the screen character table has two sets, SET 1 and SET 2. At the moment we are using SET 1. We will use SET 2 later, but for now we will stay with SET 1, the upper case set.

Questions & Answers

What will the following statements do?

(a)
```
10 POKE 1024,1
20 POKE 55296,1
```
(b)
```
10 POKE 1024, 81
20 POKE 55296, 8
```

Answer:

(a) Put a black A in the top left-hand corner of the screen.

(b) Put an orange circle in the top left-hand of the screen.

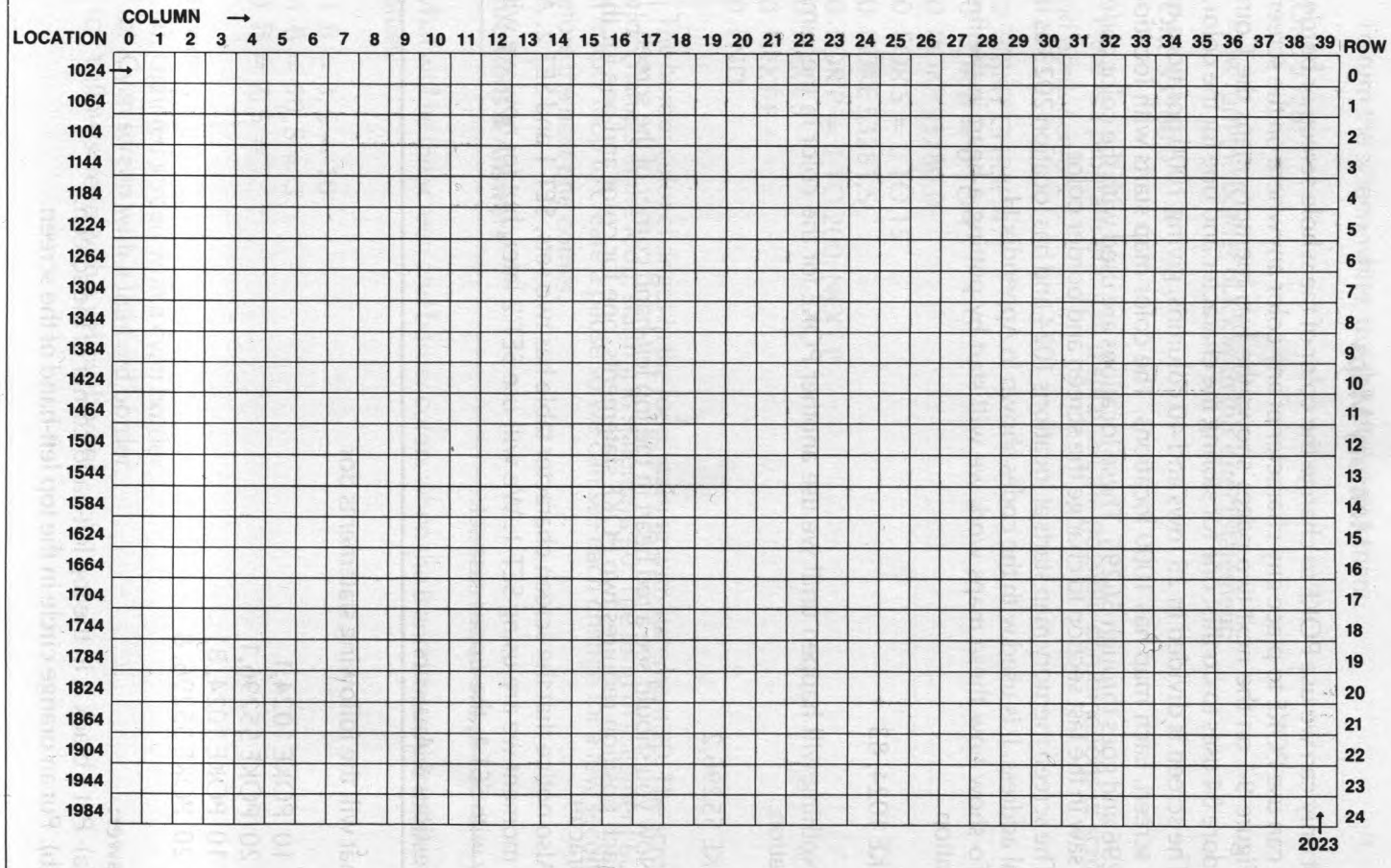
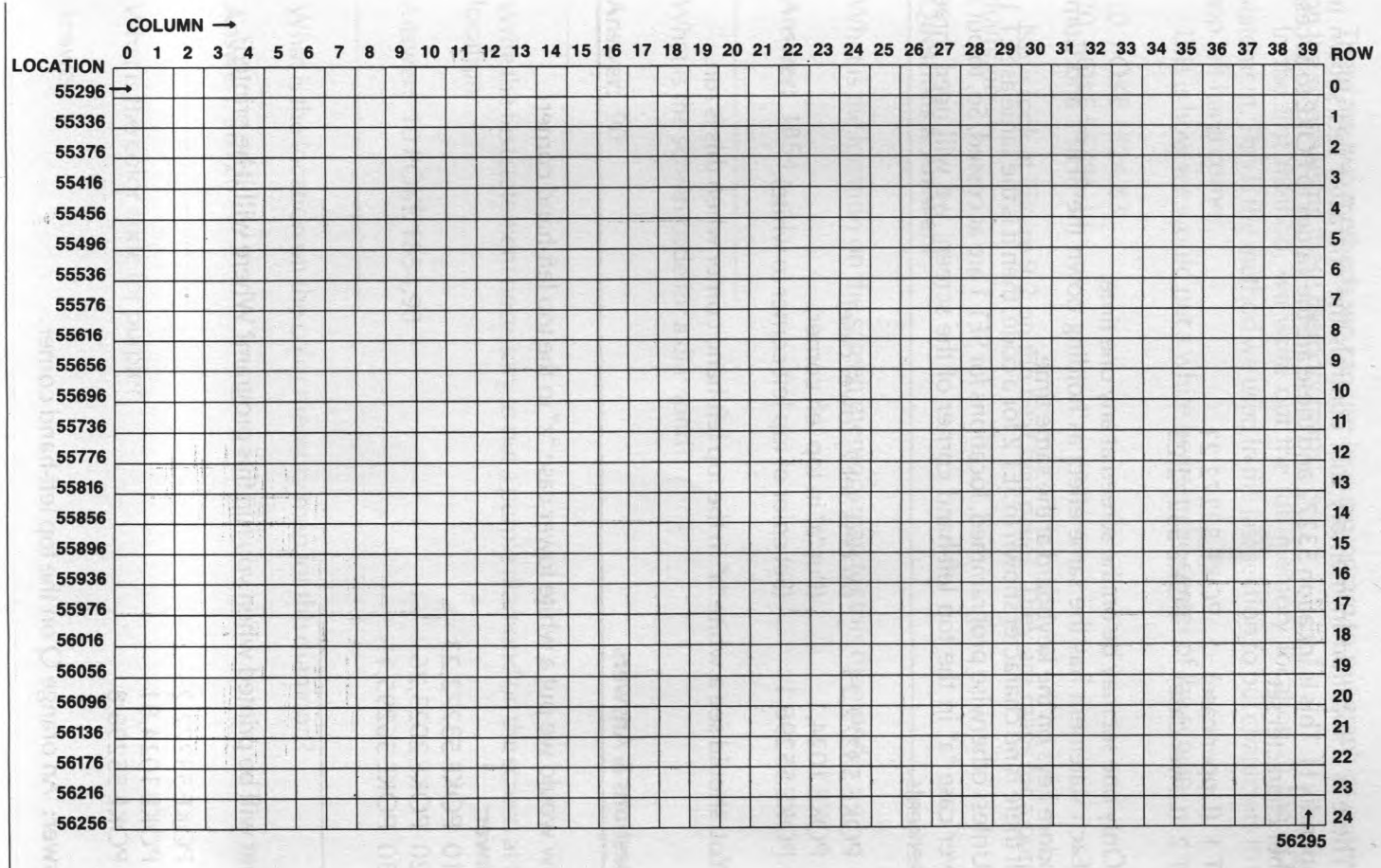


Figure 9.1 Memory maps for the Commodore 64 screen.


COLUMN →
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
ROW
LOCATION
55296 →
55336
55376
55416
55456
55496
55536
55576
55616
55656
55696
55736
55776
55816
55856
55896
55936
55976
56016
56056
56096
56136
56176
56216
56256
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
56295

There is a location in memory that controls the character sets shown in Appendix H. This is location 53272 and these are the values to POKE to access either character set:

SET 1 (Upper case) POKE 53272,21
SET 2 (Lower case) POKE 53272,23

Only one set may be on the screen at any one time.

Each statement has the same effect as holding down the SHIFT and Commodore keys on the keyboard at the same time.

If there is no character shown in SET 2 for a code, then it is the same as SET 1.

Unless otherwise programmed, locations for SET 1 are accessed. So, to put a lower case "a" in the top left-hand corner of the screen, we will need three statements.

```
10 POKE 53272,23     (Access Set 2, lower case).
20 POKE 1024,1       (Put "a" in top left corner).
30 POKE 55296,1      (Set color of top left corner to white).
```

You should see a white "a" in the top left-hand corner when this is run.

Questions & Answers

How would we put a white lower case "z" in the top left-hand corner?

Answer:

```
10 POKE 53272,23
20 POKE 2023,26
30 POKE 56295,1
```

What will be printed when you run this program? Where will it be printed?

```
10 POKE 53272,23
20 POKE 1024,81
30 POKE 55296,8
```

Answer: An orange Q on the top left-hand corner.

There are two methods we can use in our program to put graphics symbols on the screen using these memory maps.

In the first method, we write out the full memory location in the POKE statement. This is the method we used in the last section to put characters in the top left-hand corner.

This is how we would put a white heart in the center of the screen:

```
10 POKE 1524,83
20 POKE 55796,1
```

Now look at Figure 9.2 on the next two pages. They are maps for a purple diamond.

Questions & Answers

What is the location on the screen memory map for this diamond?

Answer: 1854

What is the screen code for a diamond?

Answer: 90

Write the first statement necessary to put a purple diamond on the screen at this location.

Answer: 10 POKE 1854,90

What is the location on the color memory map for the diamond?

Answer: 56126

What is the color code for purple?

Answer: 4

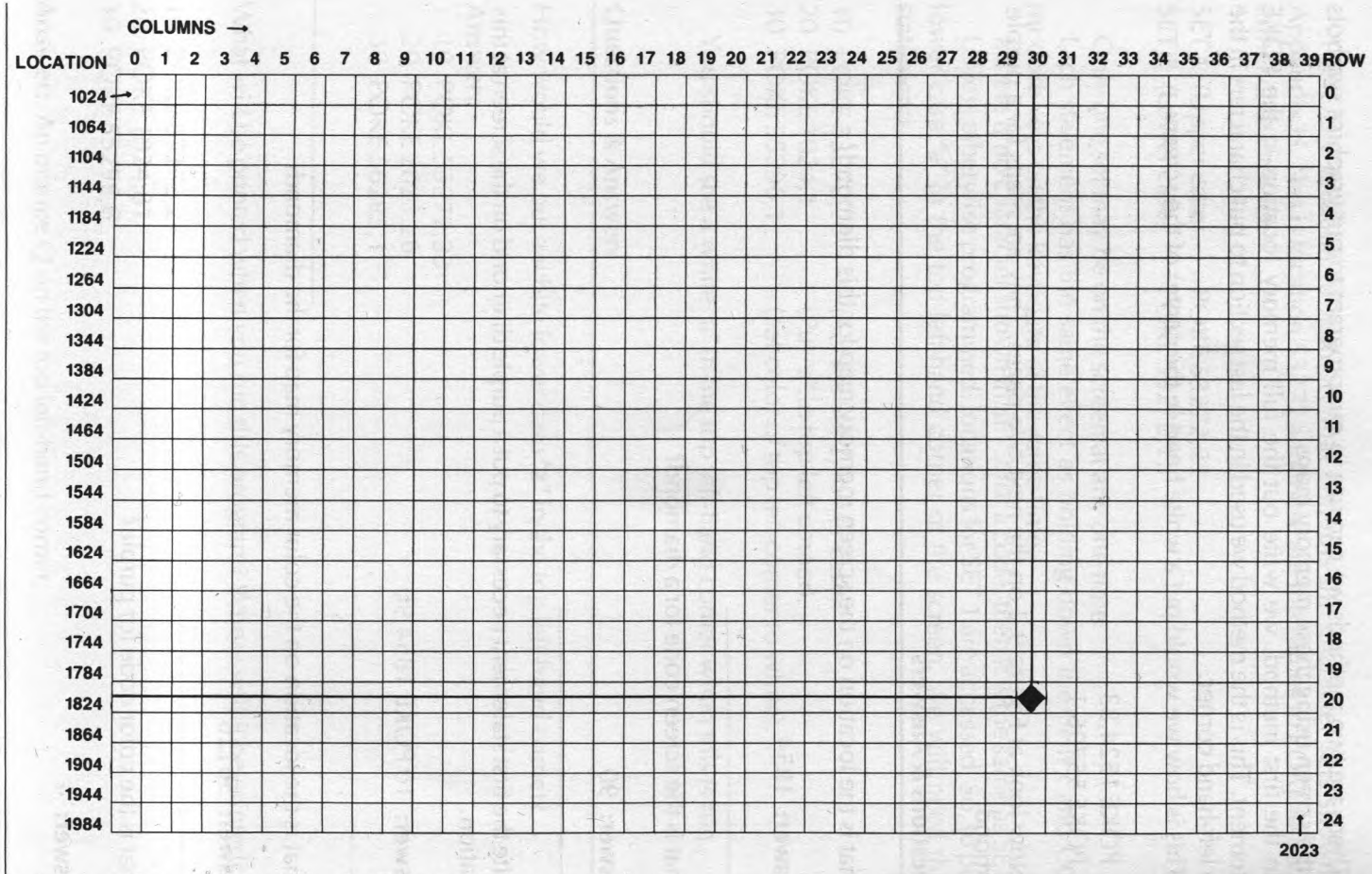
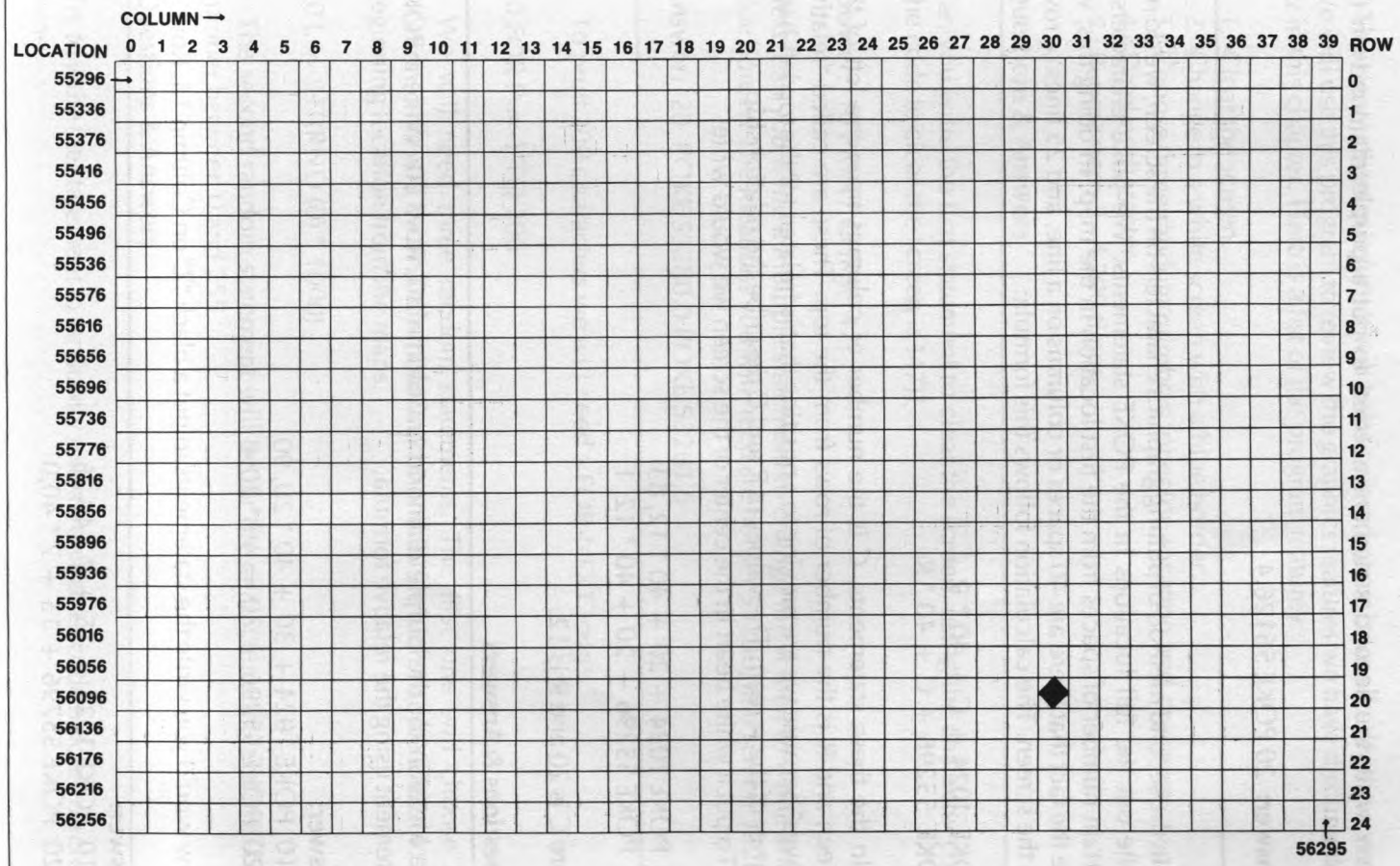


Figure 9.2 Maps for placing a purple diamond on the screen.


COLUMN →
LOCATION
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
ROW
55296 →
55336
55376
55416
55456
55496
55536
55576
55616
55656
55696
55736
55776
55816
55856
55896
55936
55976
56016
56056
56096
56136
56176
56216
56256
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
56295

Now write the second statement needed to put a purple diamond on the screen.

Answer: 20 POKE 56126,4

In the second method of putting graphics characters on the screen, we do not write out the full locations in the POKE statements. We poke characters a certain number of spaces from the first location in the map. In doing this, we use the fact that there are 40 spaces or columns on a line, and 25 lines or rows on the screen. The calculation follows this formula:

```
POKE 1024 + C + 40 * R
POKE 55296 + C + 40 * R
```

In the these statements, C is the number of columns from the left of the screen and R is the number of rows from the top. These are called "relative moves" because we are moving to a position that is "related to" the top left-hand corner of the screen. The values of R and C describe the relationship.

To put a white heart in the center of the screen we would write:

```
10 POKE 1024 + 20 + 40 * 12, 83
20 POKE 55296 + 20 + 40 * 12, 1
```

Here C is 20 and R is 12.

Questions & Answers

The location for the purple diamond is Column 30, Row 20. Write a POKE statement using the relative formula.

Answer:

```
10 POKE 1024 + 30 + 40 * 20, 90
20 POKE 55796 + 30 = 49 * 20,4
```

How would you write the statements to put a black "Z" on Column 15, Row 7?

Answer:

```
10 POKE 1024 + 15 + 7 * 40,26
20 POKE 55796 + 15 + 7 * 40,0
```

This next program is fun. It will print colored characters all over the screen. We will use this program to review the graphics features we have discussed so far in this chapter. Here is a list of the program features:

1. Clear the screen.
2. Change to a white screen with a blue border.
3. Choose three random numbers for color, location and character.
4. Display character.
5. Repeat steps 3 - 4 200 times.

Questions & Answers

Can you write the first statement to clear the screen, using the character code (the code to clear the screen is 147)?

Answer: 10 PRINT CHR$(147)

Next, can you write a multiple statement to change the screen to white and a blue border? The locations you will need are border 53280, and screen 53281.

Answer: 20 POKE 53280,0:POKE 53281,6

To print 200 characters we will need a FOR-NEXT loop:

```
30 FOR P = 1 TO 200
```

We will need three random statements. The first one will choose the location. This will be a number between 0 and 999 which will be added to the beginning location from the maps.

```
40 L = INT(RND (0) * 1000)
```

The second random statement will be for the character code and will be a number between 0 and 255.

Questions & Answers

Try to write the statement to randomly select a character code before you check mine.

Answer: 50 CH = INT (RND (0) * 256)

The last random statement will be for the color code. What will be the limits for this number?

(a) between 0 and 16
(b) between 0 and 8
(c) between 0 and 15
(d) between 1 and 15

Answer: (c)

This is the last random statement.

50 CL = INT (RND (0) * 16)

How many POKE statements will we need to put the random character on the screen?

Answer: 2

Here they are:

60 POKE 1024 + L, CH
70 POKE 55296 + L, CL

Lastly, we must close the loop and end the program.

80 NEXT P
90 END

Here is the complete program:

```
10 PRINT CHR$(147)
20 POKE 53280,6:POKE 53281,1
30 FOR P = 1 TO 200
40 L = INT(RND(0)*1000)
50 CH = INT(RND(0)*256)
60 CL = INT(RND(0)*16)
70 POKE 1024 + L,CH
80 POKE 55296 + L,CL
90 NEXT P
100 END
```

How would you change the program to print 1000 characters?

Answer:
Change the FOR statement to:

30 FOR P = 1 TO 1000

PEEK Statements

Sometimes in our programs we want to know what is stored at a particular location in memory. To do this, we use another BASIC statement, PEEK.

You can think of PEEK as being the opposite of POKE. POKE places something in memory, and PEEK tells you what is there. PEEK statements look like this:

PEEK (<location>)

- PEEK is the BASIC statement.
- <location> location or address in memory between 0 and 65535 and must be in parentheses.

Try this direct statement:

PRINT PEEK (53272)

You will remember that this is the location in memory that switches from uppercase to lowercase. It is 21 for SET 1, uppercase, and 23 for lowercase, SET 2. Normally the statement will return 21.

Questions & Answers

Now try this exercise:

Press RUN STOP and RESTORE.
Type PRINT PEEK (1024) and press RETURN.

What is location 1024?

Answer: It is the first location of the screen memory map.

The computer should respond with 32. Look up 32 in Appendix H. What does it represent?

Answer: A space.

Does this match the screen display?

Answer: Yes. READY is printed on the second line, and there is nothing on the first line of the screen. You can check this by moving the cursor.

GET Statements

We are going to use PEEK in some graphics programs that involve checking the contents of a particular location. The programs we are going to write using PEEK are game programs that require player participation. This means that the player has input into the action of the game from the keyboard. We don't want to wait while the INPUT statement is executed. Ideally, we would like to trap the user input as a key is pressed and have the program execute continuously without any breaks or pauses. To do this, we need the BASIC statement GET.

GET <variable>

- GET is the BASIC statement which will trap data from the keyboard ONE character at a time.
- <variable> may be a numeric or string variable.

If there is no response to a GET statement, then an empty, or null, variable is assumed. If you want the program to wait for a response, test for a null variable like this:

```
10 GET A$ : IF A$ = "" THEN 10
```

We use GET anytime we want to input one character at a time from the keyboard.

For example:

```
10 PRINT "WOULD YOU LIKE A SECRET MESSAGE (Y/N)?"
20 GET A$ : IF A$ = "" THEN 20
```

```
30 IF A$ < > "Y" THEN 50
40 PRINT "YOU ARE WONDERFUL"
50 PRINT "BYE FOR NOW"
60 END
```

Questions & Answers

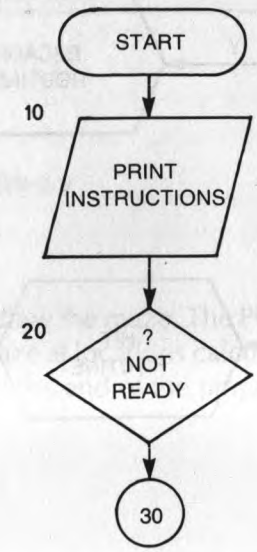
Which of the following sets of data are suitable for use with a GET statement?:

(a) The numbers 0-9
(b) The letters Y and N
(c) The symbols +, −, and =
(d) The years 1983 - 1993

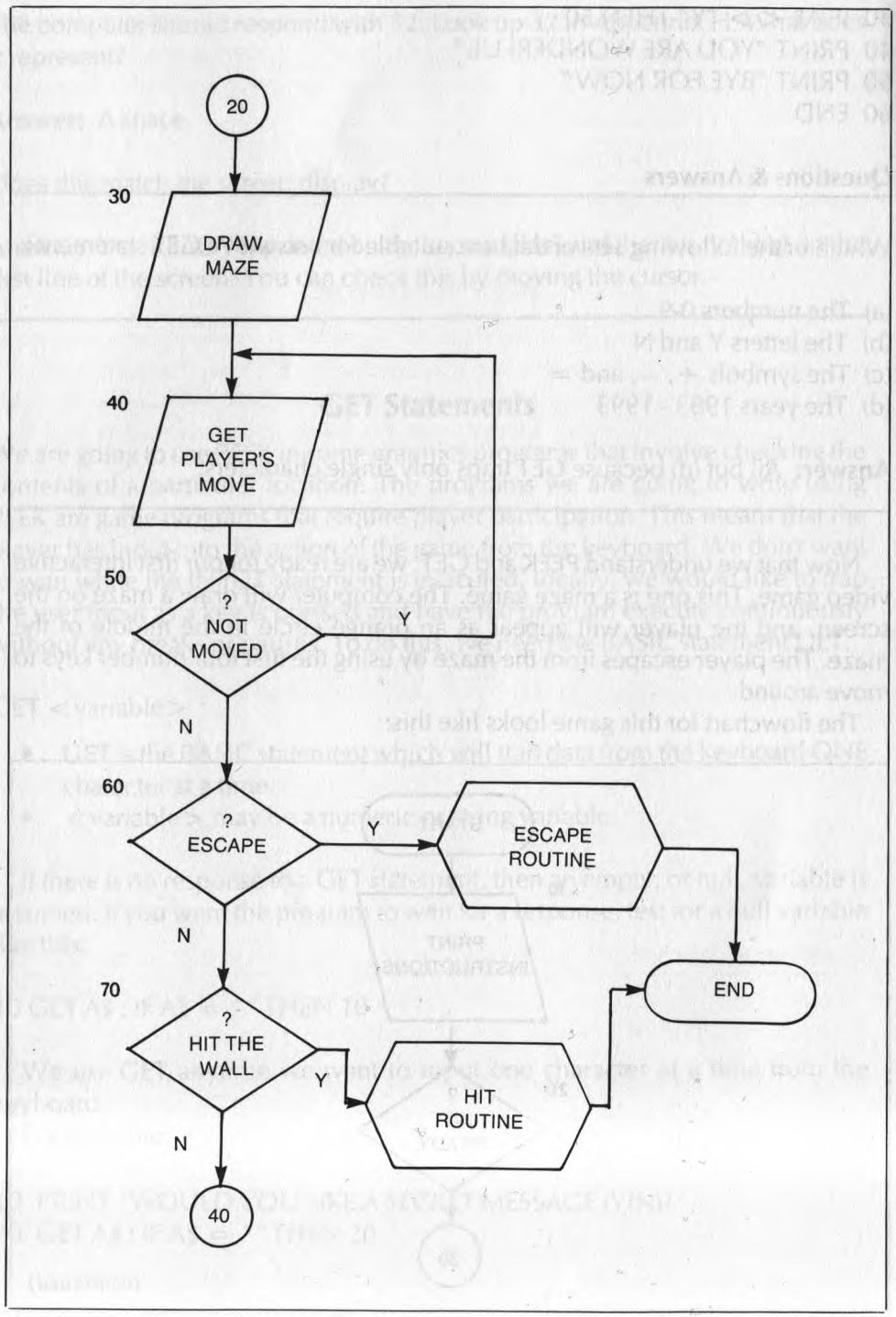
Answer: All but (d) because GET traps only single characters.

Now that we understand PEEK and GET, we are ready for our first interactive video game. This one is a maze game. The computer will draw a maze on the screen, and the player will appear as an orange circle in the middle of the maze. The player escapes from the maze by using the first four number keys to move around.

The flowchart for this game looks like this:



(continued)


20
30
DRAW MAZE
40
GET PLAYER'S MOVE
50
? NOT MOVED
Y
N
60
? ESCAPE
Y
ESCAPE ROUTINE
N
70
? HIT THE WALL
Y
HIT ROUTINE
END
N
40

The first section of the program prints the instructions for the game and waits for the player's response before moving on to draw the maze.

```
10 REM **** MAZE PROGRAM ****
20 REM **** INTRODUCTION ****
30 PRINT CHR$(147)
40 POKE 53280,1:POKE 53281,1
50 FOR C = 1 TO 8: PRINT:NEXT C
60 PRINTTAB(10)"■AMAZING MAZE"
70 PRINT :PRINT
80 PRINT " ■I WILL NOW DRAW A MAZE FOR YOU"
90 PRINT " YOU CAN ESCAPE BY USING KEYS 1-4":PRINT
100 PRINT "1 WILL MOVE YOU DOWN THE SCREEN"
110 PRINT "2 WILL MOVE YOU UP THE SCREEN"
120 PRINT "3 WILL MOVE YOU TO THE LEFT "
130 PRINT "4 WILL MOVE YOU TO THE RIGHT"
140 PRINT
150 INPUT"ARE YOU READY";A$
160 IF LEFT$(A$,1)<>"Y" THEN 150
```

Questions & Answers

(a) Which statement clears the screen?
(b) Which statement changes the screen and border color?

Answer: (a) 30 (b) 40

```
170 PRINT CHR$(147)
180 FOR C = 1 TO 372
190 READ X
200 POKE 1024+X,102:POKE 55296+X,5
210 NEXT C
```

Lines 170 through 210 draw the maze. The POKE statements in line 180 are drawing the walls of the maze at locations calculated by adding a number read from the data statements at the end of the program to the starting locations of the memory maps.

The maze was mapped out first on a grid such as the one in Figure 9-3, and then the map locations were listed in DATA statements. This procedure works well, but there is room for error; have a friend help you proof-read the data statements.

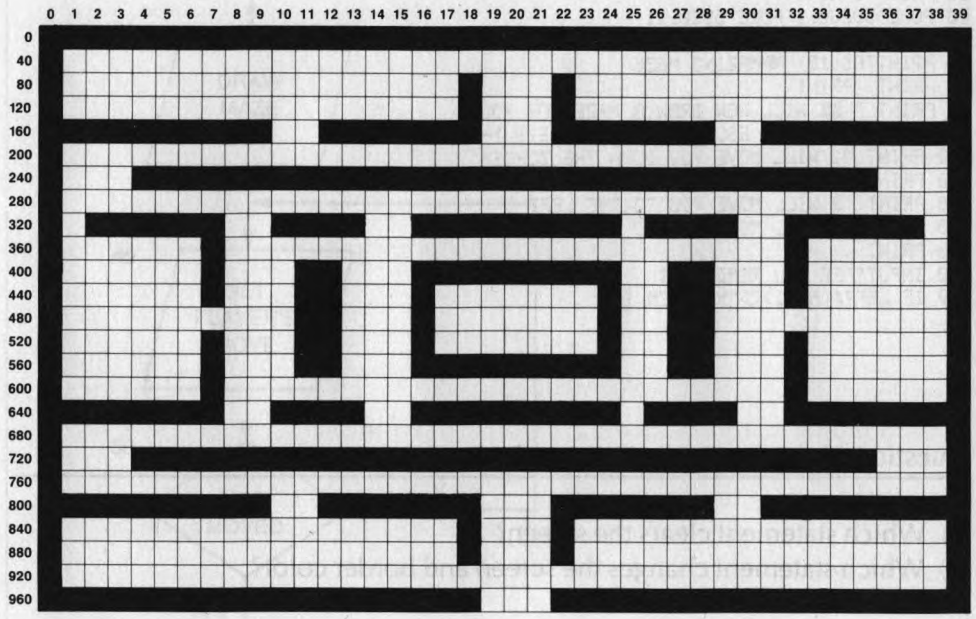


Figure 9.3 Grid for the MAZE program.

Questions & Answers

(a) The starting location for the screen memory map is _____?
(b) The starting location for the color memory map is _____?

Answer: (a) 1024 (b) 55926

```
220 REM ****MOVING AROUND****
230 A = 1524 : B = 55796
240 POKE A,81:POKE B,9
250 GET A$ : IF A$ = "" THEN 250
260 D = VAL(A$)
270 ON D GOSUB 510,540,570,600
290 IF A>2002 AND A< 2006 THEN 410
300 IF PEEK(A)<>102 THEN 240
```

In this section, the player is in the middle of the maze at locations 1524 and 55796 on the screen maps. All of the player's moves will be made relative to this starting position.

We trap the player's move using a GET statement and notice the test to wait for a move on line 250. The move is trapped as a string variable and immediately changed to a number using the VAL function.

The VAL function is one of the pre-programmed functions available on the Commodore 64. It looks like this:

VAL(<string>)

The VAL functions returns a numeric value for a string. If the string is a number, as in this program, then VAL returns that number. For any other string, VAL returns a zero. In the maze program this will make sure that the player moves only in response to the number keys.

The adjustment to the player's position is made by four subroutines located at lines 510 to 620 and accessed by the case statement, line 270.

After the player has moved, the computer checks to see if the player has won by reaching one of the locations 2002, 2003, 2004, 2005 or 2006 on the screen memory map. If so, then the computer executes a 'success' printout to the screen.

Before returning to get the player's next move, the computer checks to see if the player has hit the wall of the maze! If so, then the computer executes a 'crash' sequence before ending the program.

Here is the complete listing for the program:

```
10 REM **** MAZE PROGRAM ****
20 REM **** INTRODUCTION ****
30 PRINT CHR$(147)
40 POKE 53280,1:POKE 53281,1
50 FOR C = 1 TO 8: PRINT:NEXT C
60 PRINTTAB(10)"■AMAZING MAZE"
70 PRINT :PRINT
80 PRINT " ■I WILL NOW DRAW A MAZE FOR YOU"
90 PRINT " YOU CAN ESCAPE BY USING KEYS 1-4":PRINT
100 PRINT "1 WILL MOVE YOU DOWN THE SCREEN"
110 PRINT "2 WILL MOVE YOU UP THE SCREEN"
120 PRINT "3 WILL MOVE YOU TO THE LEFT "
130 PRINT "4 WILL MOVE YOU TO THE RIGHT"
140 PRINT
150 INPUT"ARE YOU READY";A$
160 IF LEFT$(A$,1)<>"Y" THEN 150
170 PRINT CHR$(147)
180 FOR C = 1 TO 372
190 READ X
200 POKE 1024+X,102:POKE 55296+X,5
210 NEXT C
220 REM ****MOVING AROUND****
230 A = 1524 : B = 55796
240 POKE A,81:POKE B,9
250 GET A$ : IF A$ = "" THEN 250
260 D = VAL(A$)
270 ON D GOSUB 510,540,570,600
290 IF A>2002 AND A< 2006 THEN 410
300 IF PEEK(A)<>102 THEN 240
310 REM ****CRASH****
320 FOR B = 0 TO 7
330 POKE 53280,B
340 FOR S = 0TO 15
350 POKE 53281,S
360 FOR P = 1 TO 100:NEXT P
370 NEXT S
380 NEXT B
390 PRINT CHR$(147);"YOU CRASHED - GOODBYE!"
400 END
410 REM **** SUCCESS ****
420 PRINT CHR$(147)
430 FOR T= 0 TO 12
440 PRINT TAB(5+T)"CONGRATULATIONS"
450 NEXT T
460 FOR T= 12 TO 0 STEP -1
470 PRINT TAB(5+T)"CONGRATULATIONS"
480 NEXT T
490 END
500 END
510 REM ****DOWN****
520 A=A+40: B=B+40
530 RETURN
540 REM ****UP****
550 A=A-40: B=B-40
```

(continued)

```
560 RETURN
570 REM ****LEFT****
580 A=A-1: B=B-1
590 RETURN
600 REM ****RIGHT****
610 A=A+1: B=B+1
620 RETURN
700 REM **** DATA FOR MAZE ****
710 DATA 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,
720 DATA 20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39
730 DATA 40,79,80,98,102,119,120,138,142,159,160,161,162,163,164,165
740 DATA 166,167,168,169,172,173,174,175,176,177,178,182,183,184,185
750 DATA 186,187,188,191,192,193,194,195,196,197,198,199,200,239,240
760 DATA 245,246,247,248,249,250,251,252,253,254,255,256,257,258,259
770 DATA 260,261,262,263,264,265,266,267,268,269,270,271,272,273,274
780 DATA 275,279,280,319,320,322,323,324,325,326,327,330,331,332,333
790 DATA 336,337,338,339,340,341,342,343,344,346,347,348,349,352,353
800 DATA 354,355,356,357,359,360,367,392,399,400,407,411,412,416,417
810 DATA 418,419,421,422,423,424,427,428,432,439,440,447,451,452,456
820 DATA 464,467,468,472,479,480,491,492,496,504,507,508,519,520,527
830 DATA 531,532,536,544,547,548,552,559,560,567,571,572,576,577
840 DATA 578,579,580,581,582,583,584,587,588,592,599,600,607,632,639
850 DATA 640,641,642,643,644,645,646,647,650,651,652,653,656,657,658
860 DATA 659,660,661,662,663,664,666,667,668,669,672,673,674,675,676
870 DATA 677,678,679,680,719,720,724,725,726,727,728,729,730,731,732
880 DATA 733,734,735,736,737,738,739,740,741,742,743,744,745,746,747
890 DATA 748,749,750,751,752,753,754,755,759,760,799,800,801,802,803
900 DATA 804,805,806,807,808,809,812,813,814,815,816,817,818,822,823
910 DATA 824,825,826,827,828,831,832,833,834,835,836,837,838,839,840,858
1010 DATA 862,879,880,898,902,919,920,959,960,961,962,963,964,965,966
1020 DATA 967,968,969,970,971,972,973,974,975,976,977,978
1030 DATA 982,983,984,985,986,987,988,989,990,991,992,993,994,995,996
1040 DATA 997,998,999
```

Feel free to change this program to suit your ideas.

Questions & Answers

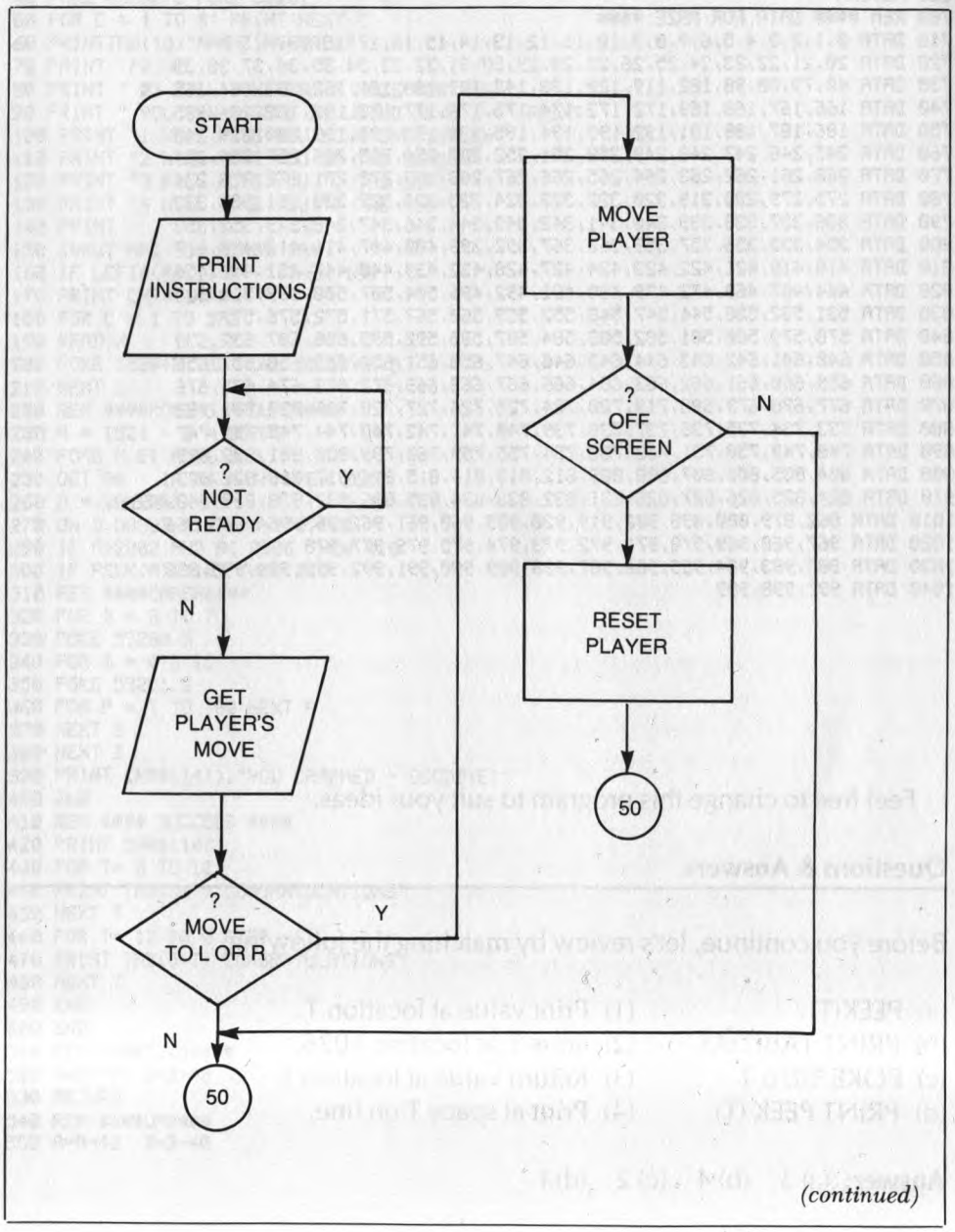
Before you continue, let's review by matching the following:

(a) PEEK(T)	(1) Print value at location T.
(b) PRINT TAB(T)A$	(2) Store T at location 1026.
(c) POKE 1026,T	(3) Return value at location T.
(d) PRINT PEEK (T)	(4) Print at space T on line.

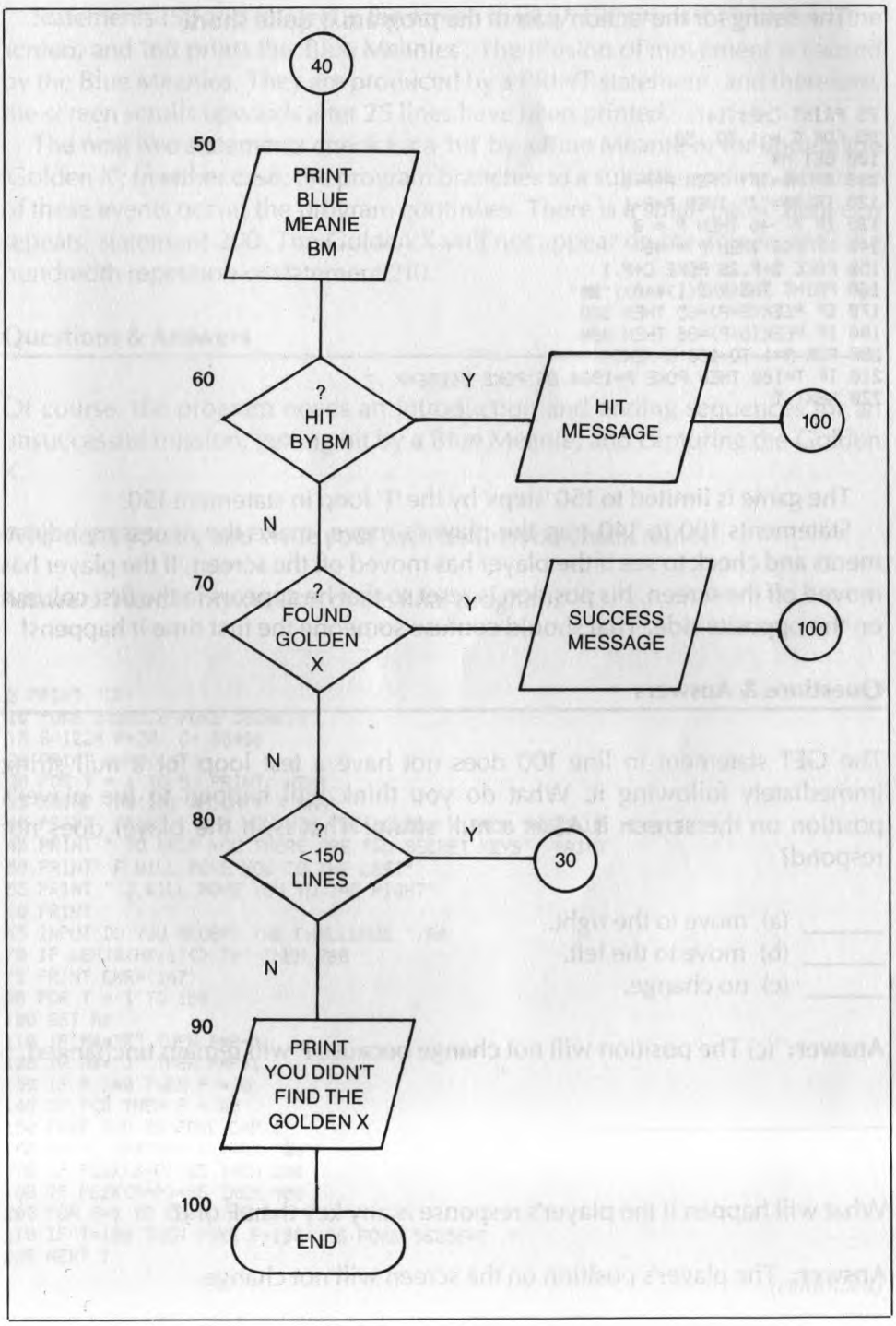
Answer: (a) 3 (b) 4 (c) 2 (d) 1

Here is another video game program. This one is called X-MAN and involves the player, represented by an English pound sign, (£), who must avoid the missiles of the Blue Meanies to reach the Golden X.

The flowchart looks like this:



(continued)


40
50
PRINT BLUE MEANIE BM
60
? HIT BY BM
Y
HIT MESSAGE
100
N
70
? FOUND GOLDEN X
Y
SUCCESS MESSAGE
100
N
80
? <150 LINES
Y
30
N
90
PRINT YOU DIDN'T FIND THE GOLDEN X
100
END

The listing for the 'action' part of the program is quite short.

```
75 PRINT CHR$(147)
95 FOR T = 1 TO 150
100 GET A$
110 IF A$="F" THEN P=P-1
120 IF A$="J" THEN P=P+1
130 IF P >40 THEN P = 0
140 IF P<0 THEN P = 40
150 POKE S+P,28:POKE C+P,1
160 PRINT TAB(RND(1)*40);"■"
170 IF PEEK(S+P)=65 THEN 300
180 IF PEEK(S+P)=86 THEN 400
200 FOR A=1 TO 150-C:NEXT
210 IF T=100 THEN POKE P+1984,86:POKE 56256+P ,7
220 NEXT T
```

The game is limited to 150 'steps' by the 'T' loop in statement 150.

Statements 100 to 140 trap the player's move, make the necessary adjustments and check to see if the player has moved off the screen. If the player has moved off the screen, his position is reset so that he appears in the first column on the opposite side. That should confuse someone the first time it happens!

Questions & Answers

The GET statement in line 100 does not have a test loop for a null string immediately following it. What do you think will happen to the player's position on the screen if A$ is a null string? That is, if the player does not respond?

_____ (a) move to the right.
_____ (b) move to the left.
_____ (c) no change.

Answer: (c) The position will not change because P will remain unchanged.

What will happen if the player's response is any key than F or J?

Answer: The player's position on the screen will not change.

Statements 150 and 160 set up the screen display: 150 prints the player on the screen, and 160 prints the 'Blue Meanies'. The illusion of movement is caused by the Blue Meanies. They are produced by a PRINT statement, and therefore, the screen scrolls upwards after 25 lines have been printed.

The next two statements check for a 'hit' by a Blue Meanie or for finding the 'Golden X'; in either case, the program branches to a suitable ending. If neither of these events occur, the program continues. There is a small pause between repeats, statement 200. The Golden X will not appear on the screen until the hundredth repetition of statement 210.

Questions & Answers

Of course, the program needs an introduction and ending sequences for an unsuccessful mission, getting hit by a Blue Meanie, and capturing the Golden X.

Why don't you try and write your own before you check mine?

Answer: Here is my version of the final program:

```
5 PRINT "■"
10 POKE 53281,0:POKE 53280,0
15 S=1224:P=20: C= 55496
20 PRINT CHR$(147)
30 FOR T = 1 TO 5: PRINT: NEXT
35 PRINT TAB(10)"■MIGHTY X MAN"
40 PRINT: PRINT " CAN YOU WIN THE GOLDEN X FROM THE BLUE MEANIES?"
45 PRINT " TO HELP YOU THERE ARE TWO SECRET KEYS" :PRINT
50 PRINT" F WILL MOVE YOU TO THE LEFT"
55 PRINT " J WILL MOVE YOU TO THE RIGHT"
60 PRINT
65 INPUT"DO YOU ACCEPT THE CHALLENGE ";A$
70 IF LEFT$(A$,1)<> "Y" THEN 260
75 PRINT CHR$(147)
95 FOR T = 1 TO 150
100 GET A$
110 IF A$="F" THEN P=P-1
120 IF A$="J" THEN P=P+1
130 IF P >40 THEN P = 0
140 IF P<0 THEN P = 40
150 POKE S+P,28:POKE C+P,1
160 PRINT TAB(RND(1)*40);"■"
170 IF PEEK(S+P)=65 THEN 300
180 IF PEEK(S+P)=86 THEN 400
200 FOR A=1 TO 150-C:NEXT
210 IF T=100 THEN POKE P+1984,86:POKE 56256+P ,7
220 NEXT T
```

(continued)

```
230 PRINT CHR$(147)
240 FOR L=1 TO 10: PRINT: NEXT L
250 PRINTTAB(5)"■YOU DIDN'T FIND THE GOLDEN X"
260 END
300 REM *** HIT ****
310 PRINT CHR$(147): POKE 53281,1
320 FOR L = 1 TO 10: PRINT: NEXT L
330 PRINT TAB(5)" THE BLUE MEANIES GOT YOU!"
340 END
400 REM *** WIN ***
410 PRINT CHR$(147)
420 FOR A=1 TO 20
430 POKE 53281,RND(1)*15
440 FOR P= 1 TO 50 : NEXT P
450 NEXT A
460 POKE 53281,3:POKE 53281,7
470 PRINT:PRINT:PRINT
480 PRINT TAB(5)"■YOU WON THE GOLDEN X"
```

The Joystick

Before we leave this chapter on graphics, we should mention other forms of controlling the input to the program. The Commodore 64 supports the use of a joystick or game paddles. In Chapter 1 we mentioned that these devices may be connected to the Control Ports on the right-hand side of the computer near the power ON/OFF switch. The memory locations associated with the joystick are 56320 and 56321:

56320 Control PORT 1
56321 Control PORT 2

The joystick moves in eight directions, represented as compass points, and also has a fire button (see Figure 9.4).

When the joystick is moved, it returns the following value for each direction:

North	1
South	2
West	4
East	8
Northwest	6
Southwest	7
Northeast	9
Southeast	10

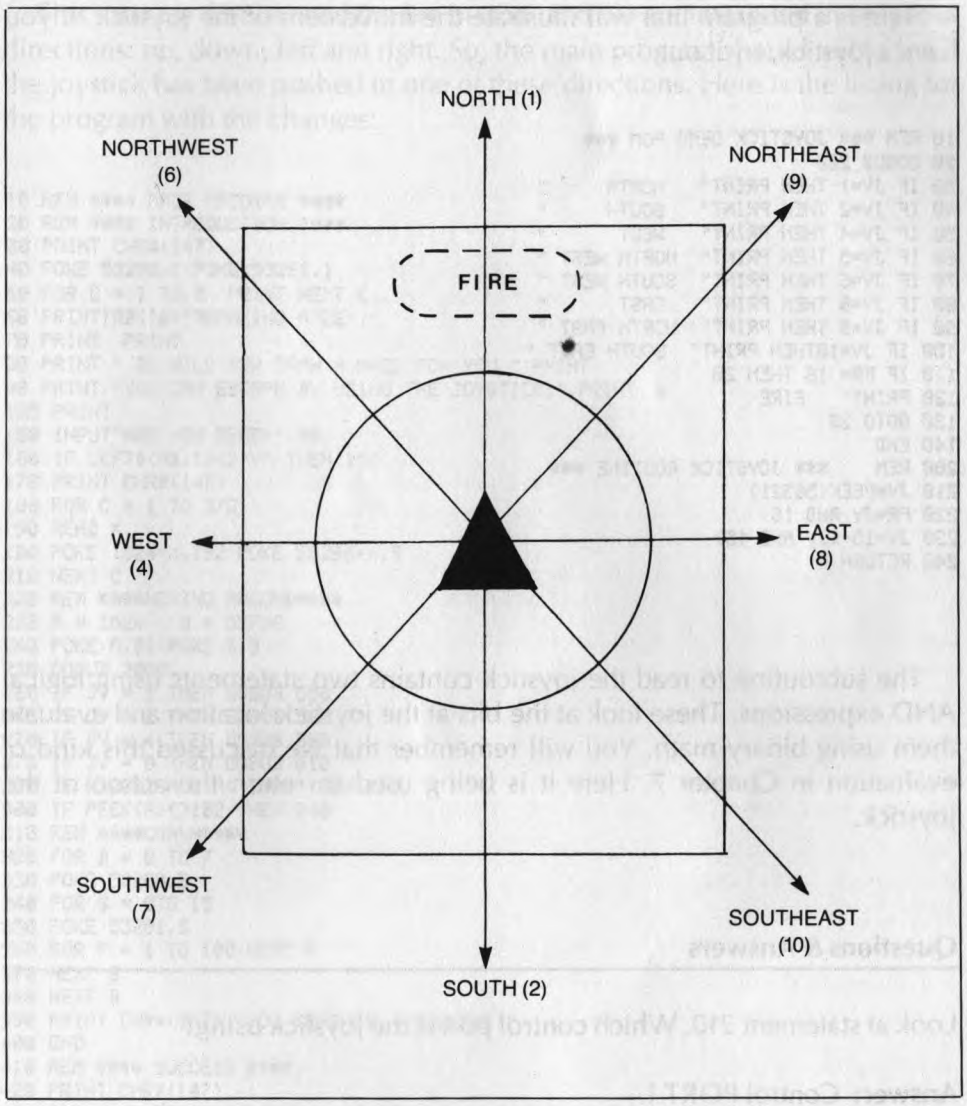


Figure 9.4 Joystick directions.

Here is a program that will illustrate the movement of the joystick. If you have a joystick, try it out.

```
10 REM *** JOYSTICK DEMO PGM ***
20 GOSUB 200
30 IF JV=1 THEN PRINT"   NORTH      "
40 IF JV=2 THEN PRINT"   SOUTH      "
50 IF JV=4 THEN PRINT"   WEST       "
60 IF JV=5 THEN PRINT"  NORTH WEST  "
70 IF JV=6 THEN PRINT"  SOUTH WEST  "
80 IF JV=8 THEN PRINT"   EAST       "
90 IF JV=9 THEN PRINT"  NORTH EAST  "
100 IF JV=10THEN PRINT"  SOUTH EAST  "
110 IF FR= 16 THEN 20
120 PRINT"   FIRE         "
130 GOTO 20
140 END
200 REM    *** JOYSTICK ROUTINE ***
210 JV=PEEK(56321)
220 FR=JV AND 16
230 JV=15-(JV AND 15)
240 RETURN
```

The subroutine to read the joystick contains two statements using logical AND expressions. These look at the bits at the joystick location and evaluate them using binary math. You will remember that we discussed this kind of evaluation in Chapter 7. Here it is being used to return the action of the joystick.

Questions & Answers

Look at statement 210. Which control port is the joystick using?

Answer: Control PORT 1.

If we add this subroutine to the maze program, we can use a joystick to escape from the maze we programmed. Where in the program should we place the subroutine?

Answer: At the end, after the main program and before the DATA statements.

This time we do not need the fire button, and we are interested in only four directions: up, down, left and right. So, the main program should test to see if the joystick has been pushed in one of these directions. Here is the listing for the program with the changes:

```
10 REM **** MAZE PROGRAM ****
20 REM **** INTRODUCTION ****
30 PRINT CHR$(147)
40 POKE 53280,1:POKE 53281,1
50 FOR C = 1 TO 8: PRINT:NEXT C
60 PRINTTAB(10)"■AMAZING MAZE"
70 PRINT :PRINT
80 PRINT " ■I WILL NOW DRAW A MAZE FOR YOU.":PRINT
90 PRINT "YOU CAN ESCAPE BY USING THE JOYSTICK.":PRINT
125 PRINT
150 INPUT"ARE YOU READY";A$
160 IF LEFT$(A$,1)<>"Y" THEN 150
170 PRINT CHR$(147)
180 FOR C = 1 TO 372
190 READ X
200 POKE 1024+X,102:POKE 55296+X,5
210 NEXT C
220 REM ****MOVING AROUND****
230 A = 1524 : B = 55796
240 POKE A,81:POKE B,9
250 GOSUB 2000
260 IF JV = 1 THEN GOSUB 550
265 IF JV = 2 THEN GOSUB 520
270 IF JV = 4 THEN GOSUB 580
275 IF JV = 8 THEN GOSUB 610
290 IF A>2002 AND A< 2006 THEN 410
300 IF PEEK(A)<>102 THEN 240
310 REM ****CRASH****
320 FOR B = 0 TO 7
330 POKE 53280,B
340 FOR S = 0TO 15
350 POKE 53281,S
360 FOR P = 1 TO 100:NEXT P
370 NEXT S
380 NEXT B
390 PRINT CHR$(147);"YOU CRASHED - GOODBYE!"
400 END
410 REM **** SUCCESS ****
420 PRINT CHR$(147)
430 FOR T= 0 TO 12
440 PRINT TAB(5+T)"CONGRATULATIONS"
450 NEXT T
460 FOR T= 12 TO 0 STEP -1
470 PRINT TAB(5+T)"CONGRATULATIONS"
480 NEXT T
490 END
510 REM ****DOWN****
520 A=A+40: B=B+40
530 RETURN
540 REM ****UP****
```

(continued)

```
550 A=A-40: B=B-40
560 RETURN
570 REM ****LEFT****
580 A=A-1: B=B-1
590 RETURN
600 REM ****RIGHT****
610 A=A+1: B=B+1
620 RETURN
700 REM **** DATA FOR MAZE ****
710 DATA 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,
720 DATA 20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39
730 DATA 40,79,80,98,102,119,120,138,142,159,160,161,162,163,164,165
740 DATA 166,167,168,169,172,173,174,175,176,177,178,182,183,184,185
750 DATA 186,187,188,191,192,193,194,195,196,197,198,199,200,239,240
760 DATA 245,246,247,248,249,250,251,252,253,254,255,256,257,258,259
770 DATA 260,261,262,263,264,265,266,267,268,269,270,271,272,273,274
780 DATA 275,279,280,319,320,322,323,324,325,326,327,330,331,332,333
790 DATA 336,337,338,339,340,341,342,343,344,346,347,348,349,352,353
800 DATA 354,355,356,357,359,360,367,392,399,400,407,411,412,416,417
810 DATA 418,419,421,422,423,424,427,428,432,439,440,447,451,452,456
820 DATA 464,467,468,472,479,480,491,492,496,504,507,508,519,520,527
830 DATA 531,532,536,544,547,548,552,559,560,567,571,572,576,577
840 DATA 578,579,580,581,582,583,584,587,588,592,599,600,607,632,639
850 DATA 640,641,642,643,644,645,646,647,650,651,652,653,656,657,658
860 DATA 659,660,661,662,663,664,666,667,668,669,672,673,674,675,676
870 DATA 677,678,679,680,719,720,724,725,726,727,728,729,730,731,732
880 DATA 733,734,735,736,737,738,739,740,741,742,743,744,745,746,747
890 DATA 748,749,750,751,752,753,754,755,759,760,799,800,801,802,803
900 DATA 804,805,806,807,808,809,812,813,814,815,816,817,818,822,823
910 DATA 824,825,826,827,828,831,832,833,834,835,836,837,838,839,840,858
1010 DATA 862,879,880,898,902,919,920,959,960,961,962,963,964,965,966
1020 DATA 967,968,969,970,971,972,973,974,975,976,977,978
1030 DATA 982,983,984,985,986,987,988,989,990,991,992,993,994,995,996
1040 DATA 997,998,999
2000 REM *** JOYSTICK ROUTINE ***
2020 POKE 56322,0
2030 JV=PEEK(56320)
2040 JV=15-(JV AND 15)
2050 POKE 56322,255
2070 RETURN
```

That is all there is to including the use of a joystick in a program.

The joystick subroutine we used in this program could be used in any other program that uses a joystick. Also, you could use the same testing technique to detect any of the eight directions of movement or the pressing of the fire button.

Questions & Answers

Match the following joystick values to the actions.

(a) 0	(1) moving up
(b) 1	(2) moving up and to the right
(c) 2	(3) moving to the left
(d) 3	(4) moving up and to the left
(e) 4	(5) moving down and to the left
(f) 5	(6) moving down
(g) 6	(7) moving right
(h) 7	(8) moving down and to the right
(i) 8	(9) not moving at all
(j) 9	
(k) 10	

Answer: (a)9, (b) 1, (c) 6, (d) 9, (e) 3, (f) 4, (g) 5, (h) 9, (i) 7, (j) 2, (k) 8.

Please don't go to Chapter 10 until you have completed the Self-Check.

SELF-CHECK

To make sure that you are completely in the picture, please take this short test.

1. How many colors are available on the Commodore 64?
2. What will the following statement print in the screen?

   10 PRINT CHR$(153) CHR$(63)
3. What is the CHR$ code to clear the screen?
4. How would you put a red asterisk in the bottom right-hand corner of the screen using POKE statements?
5. What information is stored at these locations?

   (a) 53281
   (b) 53280

6. Here is a short listing from a program:

```
210 GET A$ : IF A$ = "" THEN 210
220 A = VAL(A$)
230 ON A GOSUB 600,700,800
250 END
```

Which of the following user response's will send the program to the subroutine starting at line 600?

(a) 4
(b) 2
(c) No response
(d) 1

7. Match the following statements to their actions:

(a) POKE	(1) Returns only one character from the keyboard.
(b) INPUT	(2) Returns the value stored at a specific location in memory.
(c) GET	(3) Stores information at a specific location in memory.
(d) PEEK	(4) Returns one or more characters from the keyboard.

Answers to SELF-CHECK

1. 16
2. A green question mark.
3. 147
4. POKE 2023,42
   POKE 56295,2
5. (a) Screen color
   (b) Border color
6. d. In this section, code statement 210 is getting the player's response from the keyboard. If there is no response, the execution returns to the GET statement. Once the user has responded then the VAL function converts the string variable to a numeric variable, if at all possible. The case statement, 230, branches to the subroutines, and a response of 1 will branch to the subroutine starting at statement 600.
7. (a) 3, (b) 4, (c) 1, (d) 2.

Suggestions for Further Exploration

1. Perhaps you could add some color subroutines to programs already written. How about a color routine for a math champ in the Math Facts program?
2. Try changing the screen and border colors in some of the programs that you have written earlier. You could also change the color of the type. Try out different color combinations until you find your favorites.

Have fun !

CHAPTER 10

Sound and Music

So far, we have been silent on the subject of sound and music on the Commodore 64. The graphics programs are fun, but they can be enhanced by some sound effects or background music. The Commodore 64 has a great sound system, with three voices and the ability to sound like many different instruments. We can create a wide range of music and a large variety of sound effects on the Commodore 64. In fact, music does not have to remain in the background, but can be a solo performance in its own right.

By the end of this chapter you will be able to:

- Understand the concepts of *waveform, attack/decay, sustain/release,* and *frequency*.
- Use the SID chip to write programs including melody lines, harmony, complex tunes and sound effects.
- Include rhythm in your programs.
- Understand the use of *pulse*.

Sound

In this chapter, I have included the music for the tunes to be programmed. If you do not read music, do not be alarmed – I have included it as an aid for those who do. The rest of us will manage very well using other forms of notation.

Before we start exploring the SID (Sound Interface Device) chip on the Commodore 64, we need to spend a little time discussing the theory of sound.

Sound has certain characteristics or properties. One characteristic is that sound travels in waves. We tend to think of waves of water in the ocean or ripples in a pond. These waves have a rounded appearance, but sound waves can take different shapes, or waveforms. For example, the sound waveforms in Figure 10-1 are available on the Commodore 64.

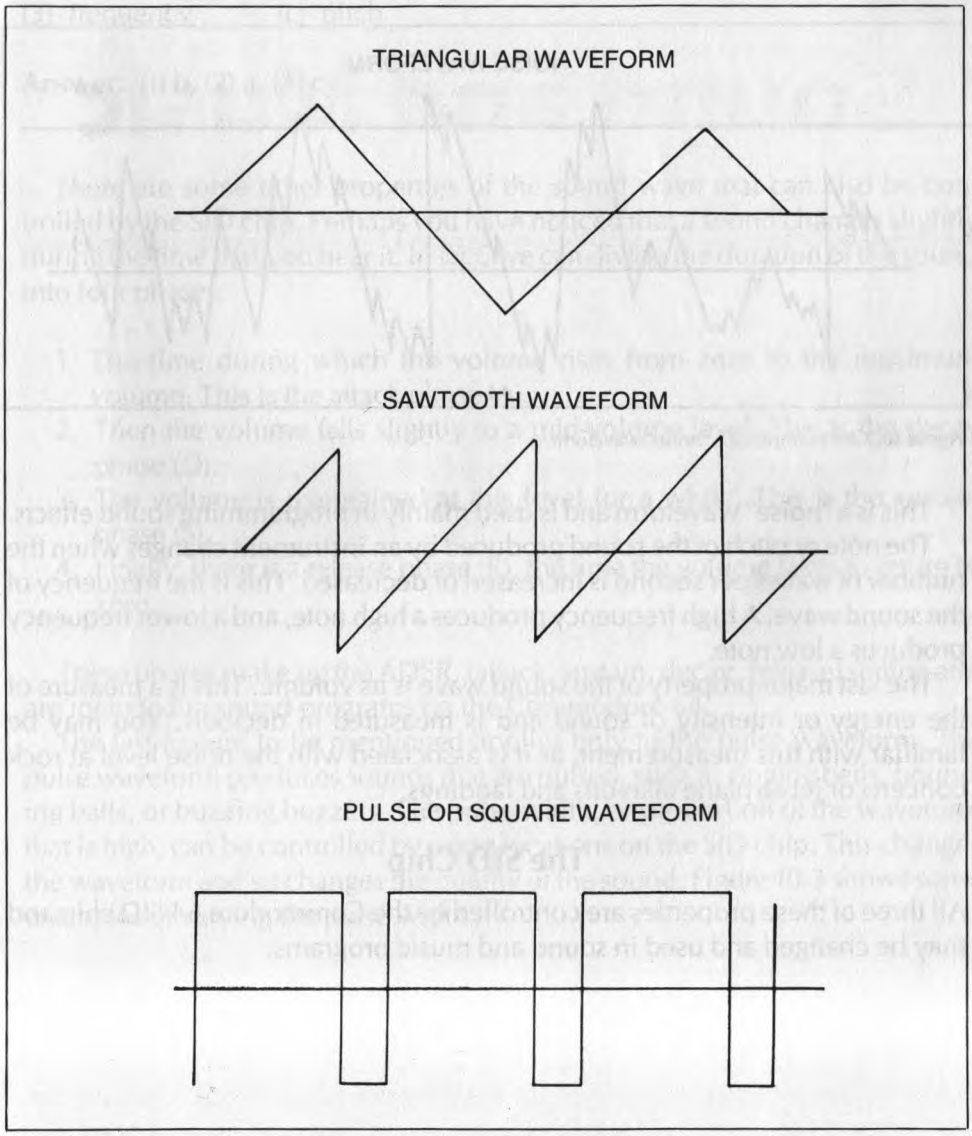


Figure 10.1 Sound waveforms available on the Commodore 64.

The waveform of a sound affects its quality. For instance, a violin, a piano and a trumpet can all play the same tune, but they sound different. Each instrument produces a unique waveform. It is this unique waveform that makes the piano sound like a piano and violin sound like a violin.

Figure 10-2 shows another waveform associated with sound. It is an erratic, random waveform.

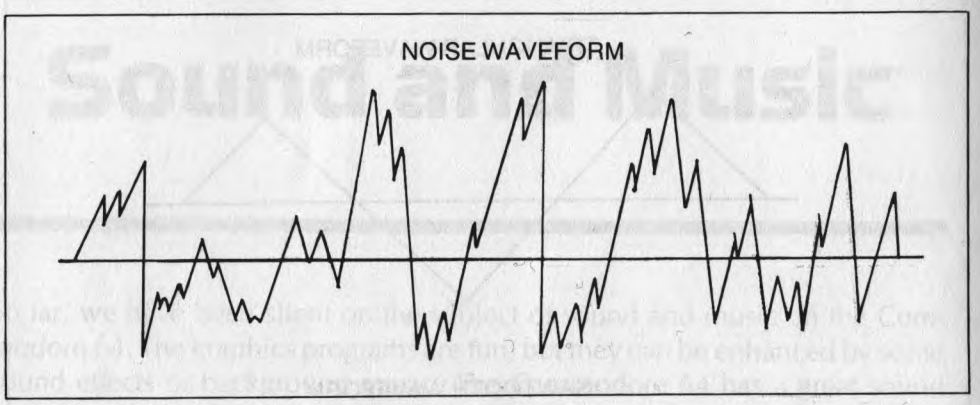


Figure 10.2 An example of a "noise" waveform.

This is a "noise" waveform and is used mainly in programming sound effects.

The note or pitch of the sound produced by an instrument changes when the number of waves per second is increased or decreased. This is the frequency of the sound wave. A high frequency produces a high note, and a lower frequency produces a low note.

The last major property of the sound wave is its volume. This is a measure of the energy or intensity of sound and is measured in decibels. You may be familiar with this measurement, as it is associated with the noise level at rock concerts or jet airplane take-offs and landings.

The SID Chip

All three of these properties are controlled by the Commodore 64 SID chip and may be changed and used in sound and music programs.

Questions & Answers

Match the following causes with the effects they create.

(1) waveform	(a) volume
(2) intensity	(b) quality
(3) frequency	(c) pitch

Answer: (1) b, (2) a, (3) c

There are some other properties of the sound wave that can also be controlled by the SID chip. Perhaps you have noticed that a sound changes slightly during the time that you hear it. In fact, we can divide the duration of the sound into four phases:

1. The time during which the volume rises from zero to the maximum volume. This is the attack phase (A).
2. Then the volume falls slightly to a mid-volume level. This is the decay phase (D).
3. The volume is maintained at this level for a while. This is the sustain phase (S).
4. Finally, there is a release phase (R), the time the volume takes to return to zero.

These phases make up the ADSR, (attack, sustain, decay, release) setting and are included in sound programs on the Commodore 64.

The last feature to be mentioned applies only to the pulse waveform. The pulse waveform produces sounds that are pulsed, such as ringing bells, bouncing balls, or buzzing buzzers. The pulse width, or proportion of the waveform that is high, can be controlled by using locations on the SID chip. This changes the waveform and so changes the quality of the sound. Figure 10-3 shows some examples of varying the pulse width.

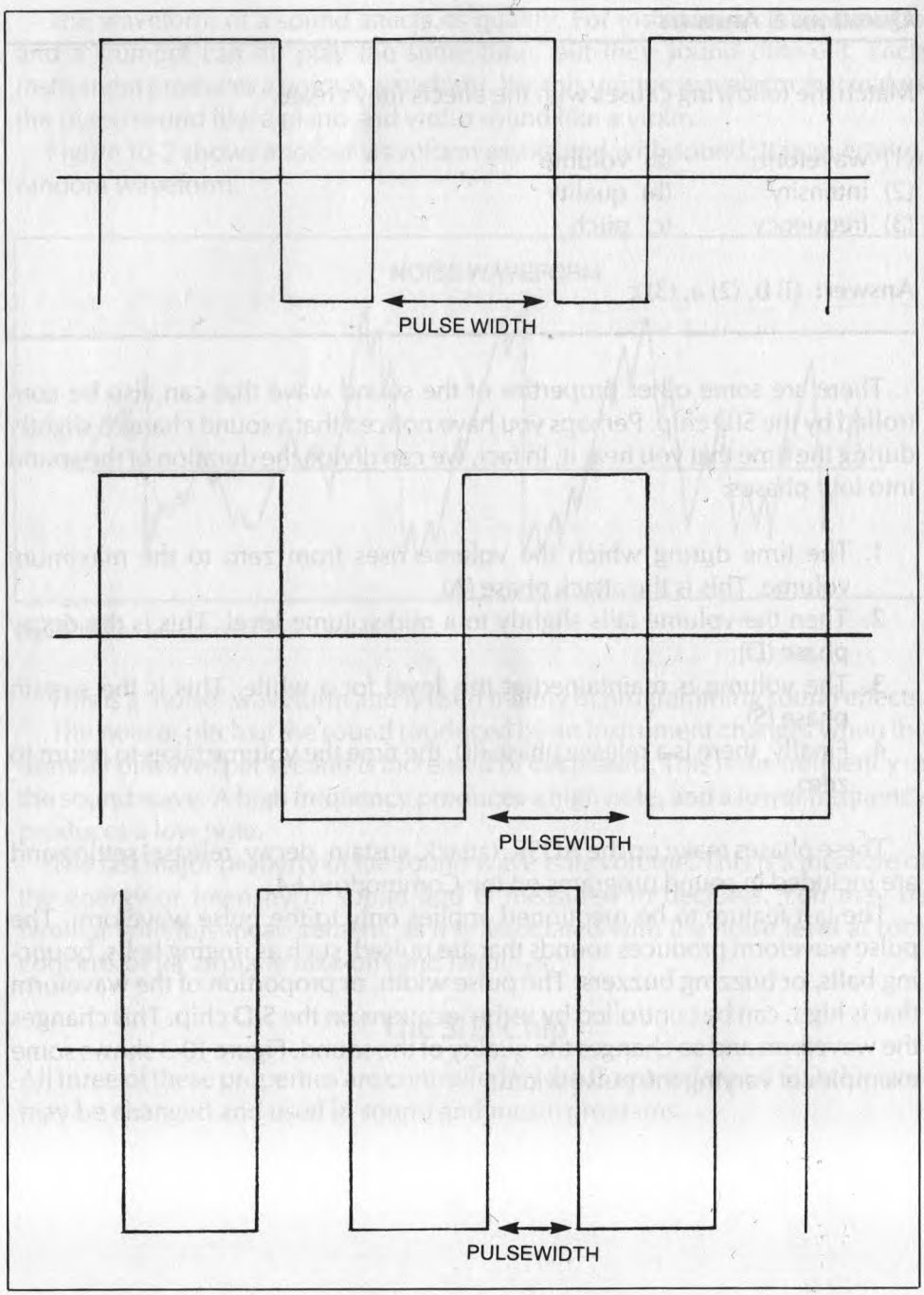


Figure 10.3 Examples of square or pulse waveforms with varying pulse widths.

Questions & Answers

Which of these statements are true?

_____ (a) The volume falls from the maximum to the mid-volume level in the decay phase.
_____ (b) The pulse setting is used only with the sawtooth wave.
_____ (c) The ADSR controls are included in every sound program.
_____ (d) The sustain phases maintains the maximum volume before it decays to zero.

Answer:

(a) True.
(b) False. The pulse setting is used with the pulse waveform.
(c) True.
(d) False. The sustain phase maintains a mid-volume level before it decays to zero.

Table 10.1 shows a map of the SID chip. This, as mentioned, is the Sound Interface Device on the Commodore 64.

Table 10.1 Locations for the SID chip.

Locations for the SID Chip

Location	Relative Location	Function	Voice
54272	S	Location frequency	1
54273	S+1	High frequency	1
54274	S+2	Waveform	1
54275	S+3	High pulse	1
54276	S+4	Low pulse	1
54277	S+5	Attack/delay	1
54278	S+6	Sustain/release	1
54279	S+7	Low frequency	2
54280	S+8	High frequency	2
54281	S+9	Waveform	2
54282	S+10	High pulse	2
54283	S+11	Low pulse	2

(continued)

Locations for the SID Chip

Location	Relative Location	Function	Voice
54284	S+12	Attack/delay	2
54285	S+13	Sustain/release	2
54286	S+14	Low frequency	3
54287	S+15	High frequency	3
54288	S+16	Waveform	3
54289	S+16	High pulse	3
54290	S+17	Low pulse	3
54291	S+18	Attack/delay	3
54292	S+19	Sustain/release	3
54296	S+24	Volume	All

As you can see from the map, the SID chip uses memory locations 54272 through 54296. In the programs developed in this chapter, we will use a relative addressing system similar to the one we used in Chapter 9 for screen locations. We will set S = 54272, the starting location of the SID chip at the beginning of each program. This numbering system is included in Figure 10-4. Its use will lessen the amount of typing that you have to do.

Music Using One Voice

There are three oscillators, or VOICES, and each voice uses seven locations.

VOICE 1	54272 – 54278	S to S+6
VOICE 2	54279 – 54285	S+7 to S+13
VOICE 3	54286 – 54292	S+14 to S+20

The volume control is located at 54296, S+24, and controls all three voices. Locations 54293 – 54295 are filter settings, which we will not be using. The locations on the SID chip are used with the POKE statement.

Questions & Answers

Can you write a POKE statement to give the volume a value of 10?

Answer: POKE S + 24,10 or POKE 54296, 10

The statements in a sound program follow a certain order.

- At the beginning of the program, the volume is turned on.
- For each voice to be used, the attack/decay and sustain/release values are set.
- A waveform is selected for each voice.
- After each note is played, the waveform is turned off.
- At the end of the program, the volume is turned off.

It is also a good idea to set all the locations on the SID chip to zero before starting a music program.

Here is a program to illustrate the pattern. It will play the C major scale using Voice 1.

```
10 S= 54272
20 FOR C= S TO S + 24
30 POKE S,0
40 NEXT C
50 POKE S+24,15
60 POKE S+5,9 : POKE S+6,0
70 FOR N = 1 TO 8
80 READ HF, LF
90 POKE S+1,HF:POKE S,LF
100 POKE S+4,17
110 FOR D= 1 TO 200:NEXT D
120 POKE S+4,16
130 NEXT N
140 POKE S+24,0
150 END
160 DATA 17,37,19,63,21,154,22,227
170 DATA 25,177,28,214,32,94,34,175
```

Line 10 sets the variable S equal to the starting location of the SID chip, 54272.

Lines 20 through 40 initialize the SID chip by poking a zero in all locations using a FOR-NEXT loop.

Line 50 sets the volume. The volume may be set to any value between 1 and 15.

Line 60 sets the AD and SR locations for voice 1. These may be set to any value from 0 to 255. We will discuss this in more detail later.

Lines 70 through 130 play the scale.

Line 80 reads the high and low frequency of the note in the scale. Each note has two frequencies. The SID chip has a range of seven octaves (fifty-six notes). The frequencies for all seven octaves are listed in Appendix J. The data statements at the end of the program contain the frequencies for octave 3, which starts at the equivalent of Middle C on the piano.

Line 100 sets the waveform. The waveforms have these values:

Triangle 17

Sawtooth 33

Pulse 65

Noise 129

Line 110 is an 'empty loop.' This creates a pause during which the note is heard. It is the duration or length of the note.

Line 120 ends the waveform. This is done by poking a number one less than the original waveform value.

Finally, before the program ends, line 140 turns off the volume by poking zero at location S + 24, (54296).

Type in the program and RUN it. Don't forget to turn up the volume control on your TV or monitor. Next we will make changes to this program and RUN those to hear the effects.

Questions & Answers

Which of these lines will change the waveform in this program from a triangle waveform to a sawtooth waveform?

_____ (a) 100 POKE S + 4, 16
_____ (b) 100 POKE S + 4, 33
_____ (c) 100 POKE S + 4, 65

Answer: (b)

Which of these lines will turn off the sawtooth waveform?

_____ (a) 120 POKE S + 24, 32
_____ (b) 120 POKE S + 4, 32
_____ (c) 120 POKE S + 4, 33

Answer: (b)

Run the program using the sawtooth waveform. Now, replace line 100 with this line for the pulse waveform:

```
100 POKE S + 4, 65
```

Don't forget to change line 120 to:

```
120 POKE S + 4, 64
```

There are two other locations associated with the pulse waveform. These are the high and low pulse-width settings, and for voice 1 these locations are S + 2 and S + 3. We will add line 105 for the pulse width.

```
105 POKE S + 2,9 : POKE S + 3,9
```

Remember, only use the pulse settings with the pulse waveforms.

Questions & Answers

Which of these statements are true?

_____ (a) The pulse settings affect the intensity of the note.
_____ (b) The pulse settings affect the waveform.
_____ (c) There are two pulse settings, high and low pulse.
_____ (d) The pulse settings are used only with the pulse form.

Answer:
(b), (c) and (d) are true.
(a) The pulse settings affect the quality, not the intensity.

The last waveform to try is the noise waveform. Usually this waveform is used for sound effects, such as explosions, but we can include it in this program to hear its effect.

Questions & Answers

We will remove line 105 and change lines 100 and 120. Can you make the changes?

Answer:

```
100 POKE S + 4, 129
120 POKE S + 4, 128
```

Let's see what happens when we change the ADSR settings in the C scale program.

Questions & Answers

What are the locations for the ADSR settings using Voice 1?

Answer: S + 5 and S + 6

The ADSR settings represent four phases of the note. The attack and decay phases are set at one location, and the sustain and release phases at a second location.

Attack				Decay			
Higher			Lowest	Higher			Lowest
128	64	32	16	8	4	2	1

Each of the eight settings shown in the table will turn on one bit in the location POKEd. (We discussed turning on individual bits in Chapter 7.) POKE S + 5,128 will turn on the eighth bit in location S + 5. POKE S + 5,1 will turn on the first bit in location S + 5 and give the lowest possible decay level.

Any combination of these values may be used by simply adding them together. For example, to get a high attack level (128) with the lowest decay

level (1) we would add the values (128 + 1 =129) and POKE it like this:

POKE S + 5, 129.

The highest possible level of attack is achieved by adding 128, 64, 32 and 16 (240) and POKEing it.

POKE S + 5, 240

Questions & Answers

What is the highest possible level for decay?

Answer: 15 (8+4+2+1)

Similarly, we can combine attack and decay levels into one statement. The highest possible attack level combined with the highest possible decay level is:

POKE S + 5, 255

The settings for sustain and release are calculated in the same way.

Sustain				Release			
Higher			Lowest	Higher			Lowest
128	64	32	16	8	4	2	1

Questions & Answers

The sustain-release location for voice 1 is S + 6. How would you set it to the maximum sustain and release?

Answer: POKE S + 6, 255

It is also possible to have a zero level for any of the phases. For example:

POKE S + 5, 64

This is a medium attack level with zero decay for voice 1.

Questions & Answers

Use the table to write POKE statements for the following settings.

(a) Voice 1, High sustain, zero release
(b) Voice 2, High attack, zero decay
(c) Voice 3, Med sustain, lowest release

Answer:

(a) POKE S + 6,128
(b) POKE S + 12,128
(c) POKE S + 20,65

Try substituting different ADSR settings in line 60 in the scale program.

Music and sound programs allow you to be creative, and you will find combinations of waveforms and ADSR settings that you like more than others. Experiment for a while until you find your favorites. After all, what is music to some people is noise to others!

Here are some waveform and ADSR settings which approximate the sound of musical instruments:

Instrument	Waveform	AD	SR	Pulse
xylophone	triangle	9	9	none
piano	pulse	9	6	Hi 8, Lo 255
violin	sawtooth	5	6	none
organ	triangle	0	240	none
oboe	triangle	64	0	none
flute	triangle	96	0	none
trumpet	sawtooth	96	0	none
accordion	triangle	102	0	none
harpsichord	sawtooth	9	0	none

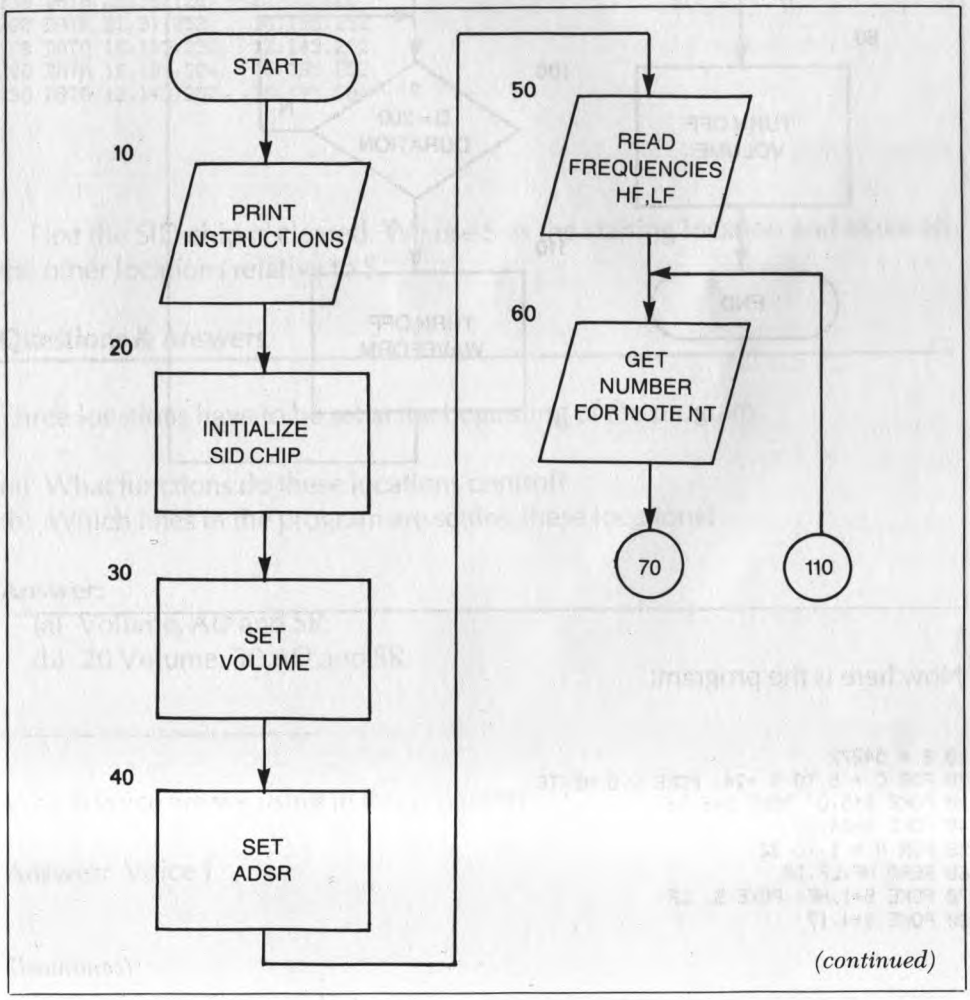
I am sure you can add to the list, as well as come up with your own unique sounds.

Questions & Answers

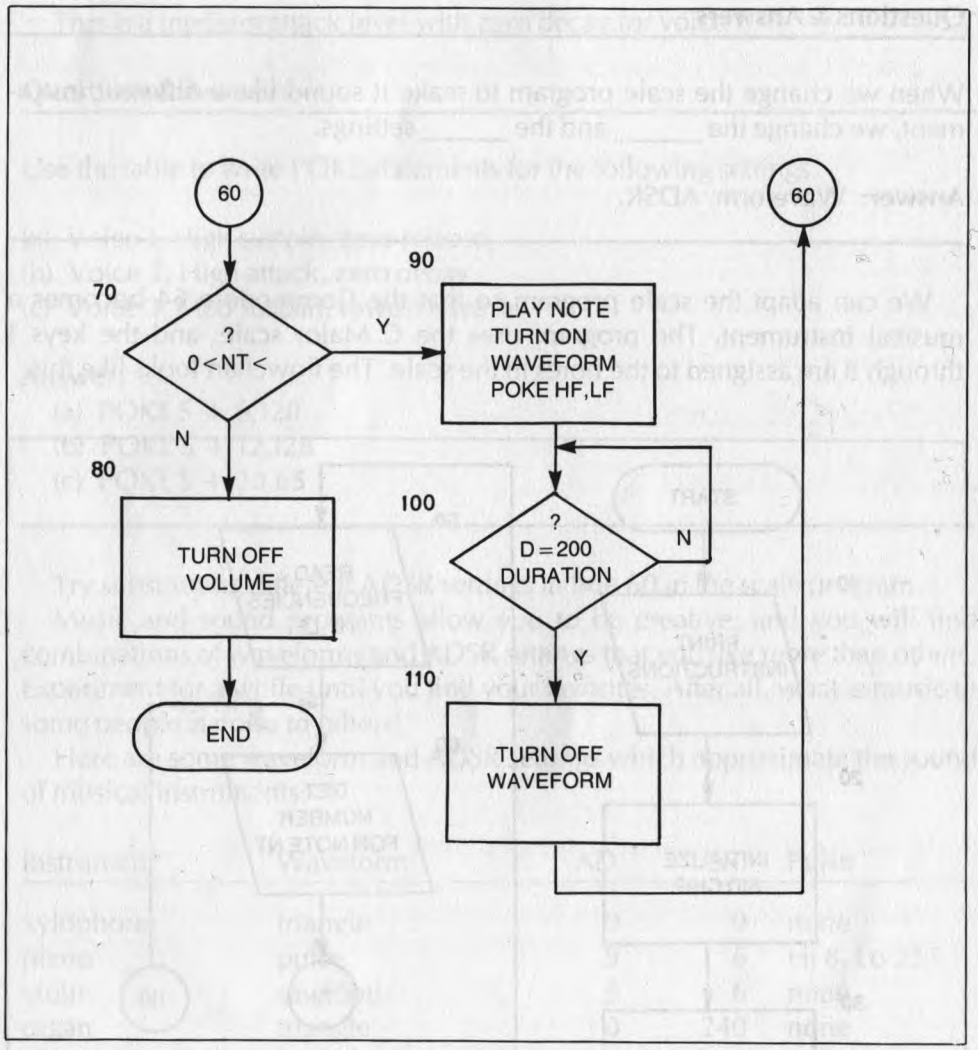
When we change the scale program to make it sound like a different instrument, we change the _____ and the _____ settings.

Answer: Waveform, ADSR.

We can adapt the scale program so that the Commodore 64 becomes a musical instrument. The program uses the C Major scale, and the keys 1 through 8 are assigned to the notes in the scale. The flowchart looks like this:



(continued)



Now here is the program:

```
10 S = 54272
20 FOR C = S TO S +24: POKE C,0:NEXTC
30 POKE S+5,0: POKE S+6,96
40 POKE S+24,15
50 FOR N = 1 TO 32
60 READ HF,LF,DR
70 POKE S+1,HF: POKE S, LF
80 POKE S+4,17
```

(continued)

```
90 FOR T= 1 TO DR: NEXT T
100 POKE S+4,16
110 NEXT N
115 POKE S+24,0
120 POKE S+24,0
130 END
140 DATA 16,195,252, 18,209,252
150 DATA 21,31,252,  16,195,252
160 DATA 16,195,252, 18,209,252
170 DATA 21,31,252,  16,195,252
180 DATA 21,31,252,  22,96,252
190 DATA 25,30,504,  21,31,252
200 DATA 22,96,252,  25,30,504
210 DATA 25,30,126,  28,49,126
220 DATA 25,30,126,  22,96,126
230 DATA 21,31,252,  16,195,252
240 DATA 25,30,126,  28,49,126
250 DATA 25,30,126,  22,96,126
260 DATA 21,31,252,  16,195,252
270 DATA 16,195,252, 12,143,252
280 DATA 16,195,504, 16,195,252
290 DATA 12,143,252, 16,195,504
```

First the SID chip is cleared. We use S as the starting location and make all the other locations relative to S.

Questions & Answers

Three locations have to be set at the beginning of the program.

(a) What functions do these locations control?
(b) Which lines in the program are setting these locations?

Answer:
(a) Volume, AD and SR.
(b) 20 Volume, 30 AD and SR.

Which voice are we using in this program?

Answer: Voice 1.

The high and low frequencies are read into the arrays from the data statements at the end of the program.

Why is there no dimension statement at the beginning of the program?

Answer: The arrays have eight elements, and arrays with less than eleven elements do not need dimension statements.

The key pressed by the player is trapped by a GET statement as a string variable, which is then changed to a number using the VAL function. The program branches to a subroutine where the choice of key is analyzed. If the key is not a number between 1 and 8, then the control returns to the main program and the program ends.

If the key is a valid choice, then the note is played in lines 110 to 170.

Questions & Answers

In this program, the waveform used is a:

_____ (a) triangle
_____ (b) sawtooth
_____ (c) pulse
_____ (d) noise

Answer: (a)

What does line 170 do?

Answer: It turns off the waveform.

I like this program, but it could be improved with some graphics. Here is one way to make the screen and border colors change every time a note is played. Add this line:

175 POKE 53281,NT: POKE 53280,NT

Perhaps you can think up your own graphics subroutine to add to the program.

Creating Rhythms

So far, the programs we have written have played notes of the same length, or duration. We have not considered how to play a tune with a rhythm or beat. To do this, we would need to control the length of the note, as well as its high and low frequencies.

In musical notation the different notes indicate the relative length of the note. For example, a quarter note is half the length of half note.

Note	Length	Rest	Dotted Note	Length
Eighth ♪	126	𝄾	♪.	189
Quarter ♩	252	𝄽	♩.	378
Half 𝅗𝅥	504	𝄼	𝅗𝅥.	756
Whole 𝅝	1008	𝄻	𝅝.	1512

Figure 10.4 Relative lengths of musical notes.

Notice in Figure 10-5 that the lengths in the table are not nice round numbers, such as 250 or 1000. This is because some notes are twice the value of others, while dotted notes are three times the value. To accommodate these relationships, we need durations that can be divided by 2 or by 3.

For our first tune, we will use the song "Brother John" ("Frere Jacques"). I hope this tune will be familiar to most readers. The music is shown in Figure 10.6.

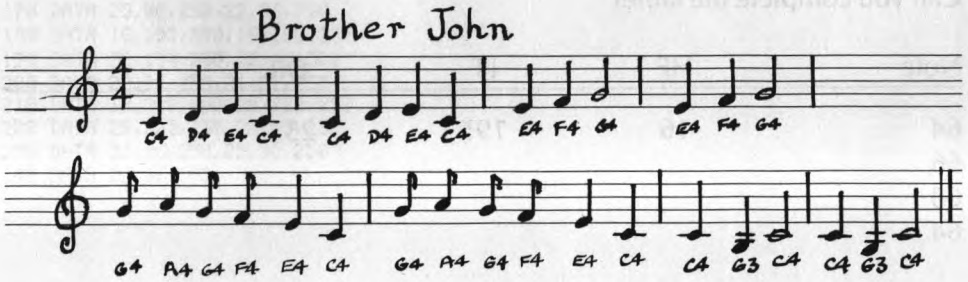



Figure 10.6 "Brother John" Music.

We will begin the program in the usual way, by clearing the SID chip and setting the column and ADSR locations.

```
10 S = 54272
20 FOR C = S TO S + 24: POKE C, 0: NEXT C
30 POKE S + 5, 0: POKE S + 6, 96
40 POKE S + 24,15
```

Now, read in the high and low frequency for the note and the duration. The loop to read and play the notes looks like this:

```
50 FOR N = 1 TO 32
60 READ HF, LF, DR
70 POKE S + 1, HF: POKE S, LF
80 POKE S + 4,17
90 FOR T = 1 TO DR: NEXT T
100 POKE S + 4, 16
110 NEXT N
120 POKE S + 24, 0
130 END
```

Each time a note is read, it is played and its length is used in an empty loop, line 90.

Line 120 turns off the volume.

The DATA statements will follow at the end of the program. Here is a table for the first four notes. I have filled in the values for the first note using the frequency tables in the Appendix J and the duration table shown in Figure 10.5

Questions & Answers

Can you complete the table?

Note	HF	LF	DR
64	16	195	252
66			
68			
64			

Answer:

Note	HF	LF	DR
64	16	195	252
66	18	209	252
68	21	31	252
64	16	195	252

Try writing the DATA statements by yourself. I recommend putting the values for two notes in each DATA statement. This will make them easy to proofread.

Answer: Here is a listing for the program with the DATA statements:

```
10 S = 54272
20 FOR C = S TO S +24: POKE C,0:NEXTC
30 POKE S+5,0: POKE S+6,96
40 POKE S+24,15
50 FOR N = 1 TO 25
60 READ HF,LF,DR
70 POKE S+1,HF: POKE S, LF
80 POKE S+4,17
90 FOR T= 1 TO DR: NEXT
100 POKE S+4,16 :FOR P= 1 TO 10:NEXTP
110 NEXT N
115 POKE S+24,0
120 DATA 16,195,250,16,195,250
130 DATA 18,209,250,16,195,250
140 DATA 22,96,250,21,31,750
150 DATA 16,195,250,16,195,250
160 DATA 18,209,250,16,195,250
170 DATA 25,30,250,22,96,750
180 DATA 16,195,500,16,195,500
190 DATA 33,134,500,28,49,500
200 DATA 22,96,250,21,31,250
210 DATA 18,209,250,29,223,250
220 DATA 29,223,250,28,49,250
230 DATA 22,96,250,25,30,250
240 DATA 22,96,500
```

Music for One to Three Voices

Now we have a tune with rhythm, but it is using only one voice. It is time to program the other two voices to sing along. One interesting arrangement is to program a tune played as a round by the three voices. Figure 10-7 shows the music for the "Tallis Canon":

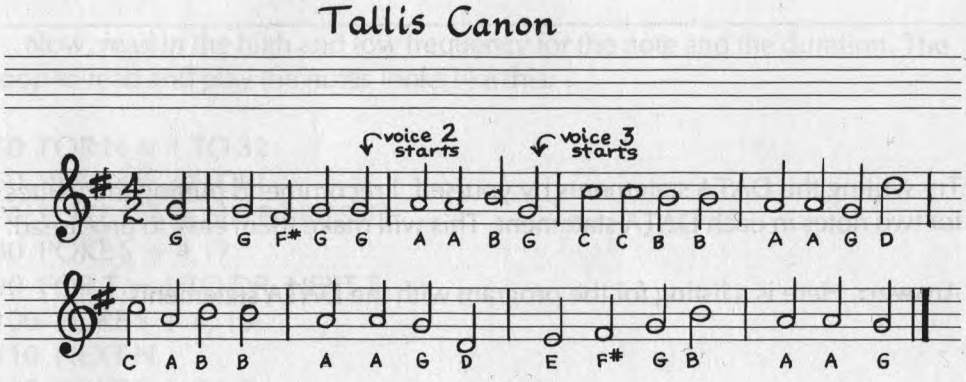
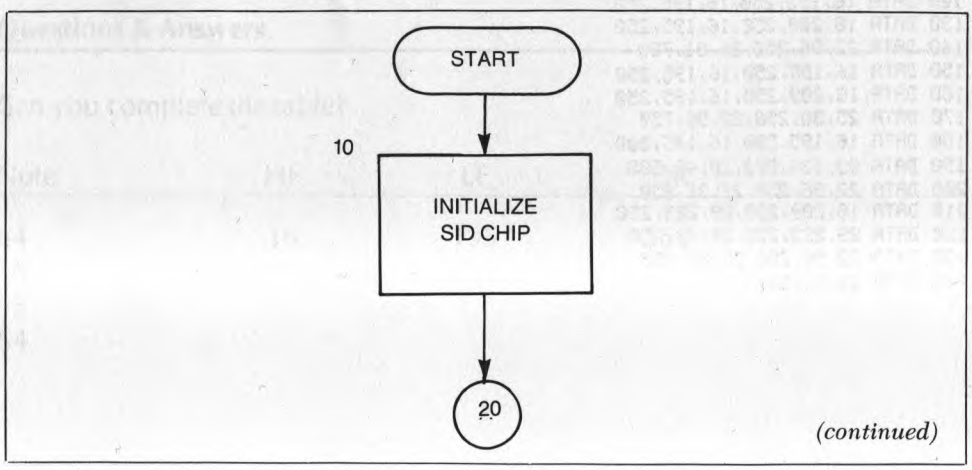


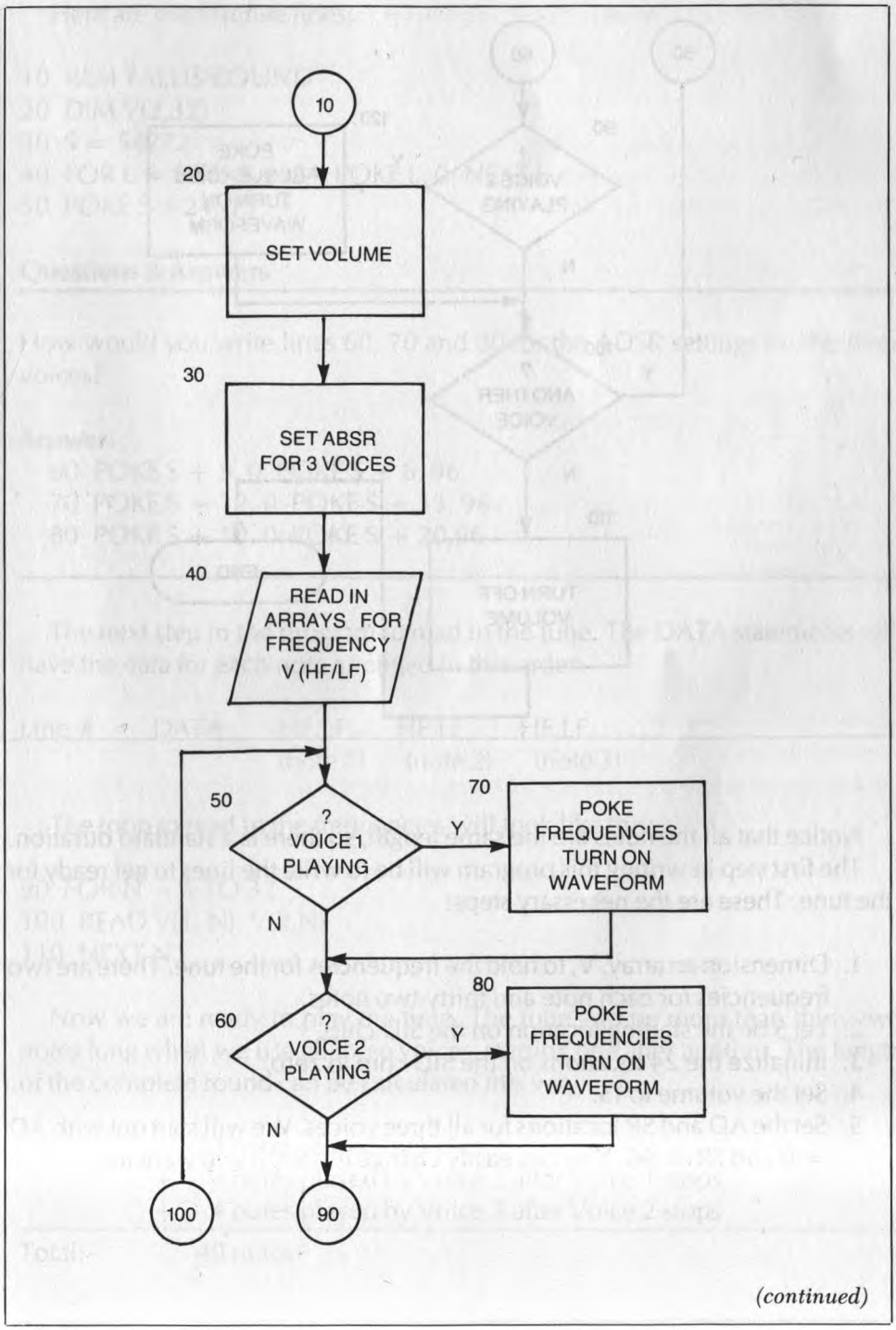
Figure 10.6 "Tallis Canon" music.

Notice that Voice 1 starts first, and after four notes Voice 2 begins, and four notes later Voice 3 begins. All three voices play the complete tune, so that Voice 3 finishes last, playing the final four notes by itself.

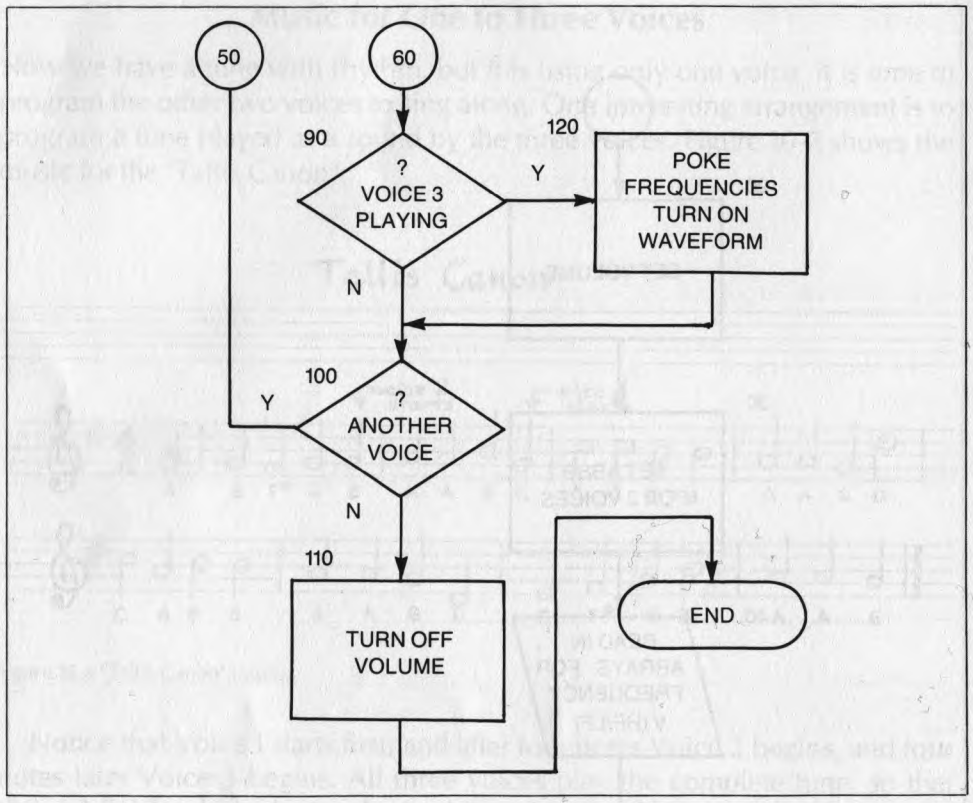
We can program this using one array to store the notes if we are careful to test how many notes have been played and which voices are playing. Here is the flowchart:



(continued)


10
20
SET VOLUME
30
SET ABSR
FOR 3 VOICES
40
READ IN
ARRAYS FOR
FREQUENCY
V (HF/LF)
50
?
VOICE 1
PLAYING
Y
70
POKE
FREQUENCIES
TURN ON
WAVEFORM
N
60
?
VOICE 2
PLAYING
Y
80
POKE
FREQUENCIES
TURN ON
WAVEFORM
N
100
90

(continued)



Notice that all the notes are the same length, so there is a standard duration.

The first step in writing this program will be to write the lines to get ready for the tune. These are the necessary steps:

1. Dimension an array, V, to hold the frequencies for the tune. There are two frequencies for each note and thirty-two notes.
2. Let S be the starting location on the SID chip.
3. Initialize the 24 locations on the SID chip to zero.
4. Set the volume to 15.
5. Set the AD and SR locations for all three voices. We will start out with AD = 0 and SR = 96. You can easily change it later, if you want to.

Here are the first five lines:

```
10 REM TALLIS ROUND
20 DIM V(2,32)
30 S = 54272
40 FOR L = S TO S + 24: POKE L, 0: NEXT L
50 POKE S +24, 15
```

Questions & Answers

How would you write lines 60, 70 and 80 for the ADSR settings for the three voices?

Answer:

```
60 POKE S + 5, 0: POKE S + 6, 96
70 POKE S + 12, 0: POKE S + 13, 96
80 POKE S + 19, 0: POKE S + 20,96
```

The next step in the program to read in the tune. The DATA statements will have the data for each note arranged in this order:

Line #	DATA	HF,LF, (note 1)	HF,LF , (note 2)	HF,LF, ... (note 3)

The loop to read in the frequencies will look like this:

```
90 FOR N = 1 TO 32
100 READ V(1, N), V(2,N)
110 NEXT N
```

Now we are ready to play the tune. The tune will be more than thirty-two notes long when we use all three voices, starting one after another. The length of the complete round can be calculated this way:

	32 notes played by Voice 1
+	4 notes played by Voice 2 after Voice 1 stops
+	4 notes played by Voice 3 after Voice 2 stops
Total:	40 notes

We will need a FOR-NEXT loop with 40 counts to play the complete round. Now it is a question of checking to see if a voice should or should not be playing. This is the schedule for each voice:

Voice 1 plays for counts 1 – 32
Voice 2 plays for counts 4 – 36
Voice 3 plays for counts 8 – 40

Here is the main loop to play the tune:

```
120 FOR N = 1 TO 40
130 IF N>32 THEN170
140 POKE S+4,17
150 POKE S+1,V1(1,N):POKE S,V1(2,N)
160 IF N<4 THEN230
170 IF N>36 THEN210
180 POKE S+11,17
190 POKE S+8,V1(1,N-4):POKE S+7,V1(2,N-4)
200 IF N< 9 THEN230
210 POKE S+18,17
220 POKE S+15,V1(1,N-8): POKE S+14,V1(2,N-8)
230 FOR D = 1 TO 500:NEXT D
240 POKE 2+4, 16: POKE S+11,16: POKE S+18, 16
250 NEXT N
```

Questions & Answers

Finally, there is one step to end the program. What is it?

Answer: Turn off the volume.

How do you do that?

Answer: 290 POKE S + 24,0

After the end of the program come the DATA statements. Try writing these yourself from the table in Appendix J before you check my listing.

Answer: Here is the complete program with the DATA statements:

```
10 REM TALLIS ROUND
20 DIM V1(2,32)
30 S = 54272
40 FOR L = S TO S+24: POKE L,0: NEXT L
50 POKE S+24,15
60 POKE S+5,0: POKE S+6,96
70 POKE S+12,0 :POKE S+13,96
80 POKE S +19,0: POKE S+20,96
90 FOR N = 1 TO 32
100 READ V1(1,N),V1(2,N)
110 NEXT N
120 FOR N = 1 TO 40
130 IF N>32 THEN170
140 POKE S+4,17
150 POKE S+1,V1(1,N):POKE S,V1(2,N)
160 IF N<4 THEN230
170 IF N>36 THEN210
180 POKE S+11,17
190 POKE S+8,V1(1,N-4):POKE S+7,V1(2,N-4)
200 IF N< 9 THEN230
210 POKE S+18,17
220 POKE S+15,V1(1,N-8): POKE S+14,V1(2,N-8)
230 FOR D = 1 TO 500:NEXT D
240 POKE 2+4, 16: POKE S+11,16: POKE S+18, 16
250 NEXT N
260 POKE S+24,0
270 END
280 DATA 25,30,25,30,23,181,25,30,25,30
290 DATA 28,49,28,49,31,165,25,30
300 DATA 33,135,33,135,31,165,31,165,28,49,28,49,25,30
310 DATA 37,162,33,165,28,49,31,165,31,165,28,49,28,49,25,30
320 DATA 18,209,21,31,23,181,25,30,31,165,28,49,28,49,25,30
```

We could write a variation of the "Tallis Canon Round" program quite easily. At the moment, all three voices are singing the same tune at the same pitch, but it might sound better with each voice singing the same tune in a different octave. We could make the first voice the original level, then have the second voice come in higher and the third voice lower. We would not need to change much of the program to do this.

Each voice would need a separate set of DATA statements, because the frequencies change as we move up and down an octave. So we will need three arrays to store the notes for the tune.

20 DIM V(2,32), V2(2,32), V3(2,32)

We need to substitute these arrays in the main loop and work out the appropriate DATA statements.

Questions & Answers

Voice 2 will start with G in octave 5. What are the high and low frequencies for this note?

Answer: 50, 60

Voice 3 will start with G in octave 3. What are the high and low frequencies for this note?

Answer: 12, 143.

See if you can complete the program on your own before you check my version.

Answer: Here is the complete listing for this variation of the "Tallis Canon."

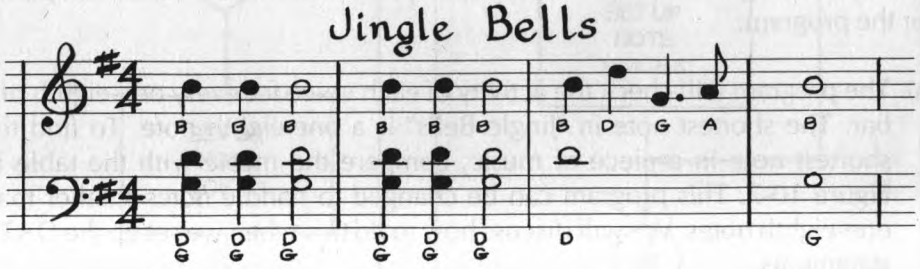
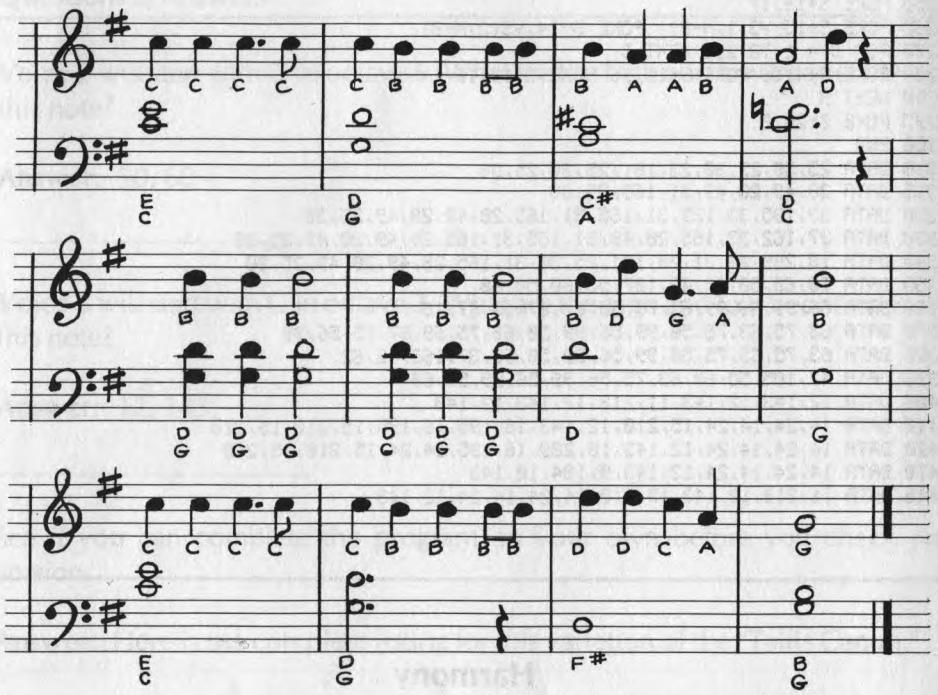
```
5 DIM V1(2,32),V2(2,32),V3(2,32)
10 S = 54272
20 FOR L = S TO S+24
30 POKE L,0
40 NEXT L
45 POKE S+24,15
50 POKE S+5,0: POKE S+6,96
52 POKE S+12,0 :POKE S+13,96
54 POKE S +19,0: POKE S+20,96
60 FOR N = 1 TO 32
70 READ V1(1,N),V1(2,N)
80 NEXT N
90 FOR N = 1 TO 32
100 READ V2(1,N),V2(2,N)
110 NEXT N
120 FOR N = 1 TO 32
130 READ V3(1,N),V3(2,N)
140 NEXT N
180 FOR N = 1 TO 40
185 IF N>32 THEN 215
190 POKE S+4,17
200 POKE S+1,V1(1,N):POKE S,V1(2,N)
210 IF N<4 THEN 270
215 IF N>36 THEN 250
220 POKE S+11,17
230 POKE S+8,V2(1,N-4):POKE S+7,V2(2,N-4)
240 IF N< 9 THEN 270
```

(continued)

```
250 POKE S+18,17
260 POKE S+15,V3(1,N-8): POKE S+14,V3(2,N-8)
270 FOR D = 1 TO 256:NEXT D
280 POKE S+4,16:POKE S+11,16:POKE S+18,16
290 NEXT N
295 POKE S+24,0
300 END
310 DATA 25,30,25,30,23,181,25,30,25,30
315 DATA 28,49,28,49,31,165,25,30
320 DATA 33,135,33,135,31,165,31,165,28,49,28,49,25,30
330 DATA 37,162,33,165,28,49,31,165,31,165,28,49,28,49,25,30
340 DATA 18,209,21,31,23,181,25,30,31,165,28,49,28,49,25,30
350 DATA 50,60,50,60,47,107,50,60,50,60
360 DATA 56,99,56,99,63,75,50,60,67,15,67,15
370 DATA 63,75,63,75,56,99,56,99,50,60,75,69,67,15,56,99
380 DATA 63,75,63,75,56,99,56,99,50,60,37,162,42,62
390 DATA 47,105,50,60,63,75,56,99,56,99,50,60
400 DATA 12,143,12,143,11,218,12,143,12,143
410 DATA 14,24,14,24,15,210,12,143,16,195,16,195,15,210,15,210
420 DATA 14,24,14,24,12,143,18,209,16,195,14,24,15,210,15,210
430 DATA 14,24,14,24,12,143,9,104,10,143
440 DATA 11,218,12,143,15,210,14,24,14,24,12,143
```

Harmony

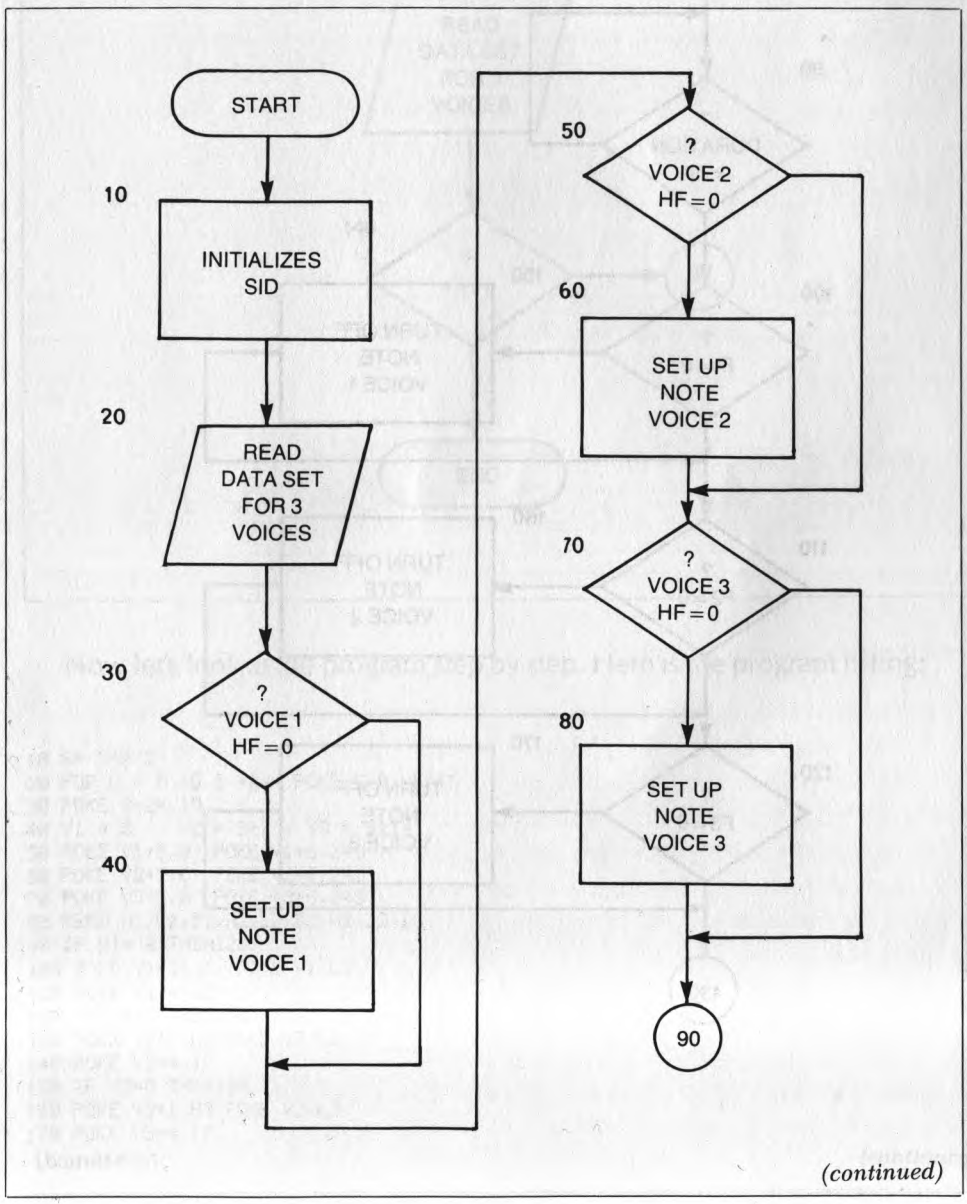
These programs for the "Tallis Canon" are pretty, but they will work only for that tune. The notes are all the same length in all three voices, but this is not true for all harmony. Look at this arrangement of "Jingle Bells," shown in Figure 10.8.

Figure 10.7 "Jingle Bells" Music

(continued)

What we need is an all-purpose program to play a tune using all three voices with each voice playing independently of the other two. There are several ways of doing this. Here is one method that is easy to follow, but not necessarily the most efficient. Once you understand how it works, however, you can easily modify the program to play whatever tune you want. These are the main points for the program:

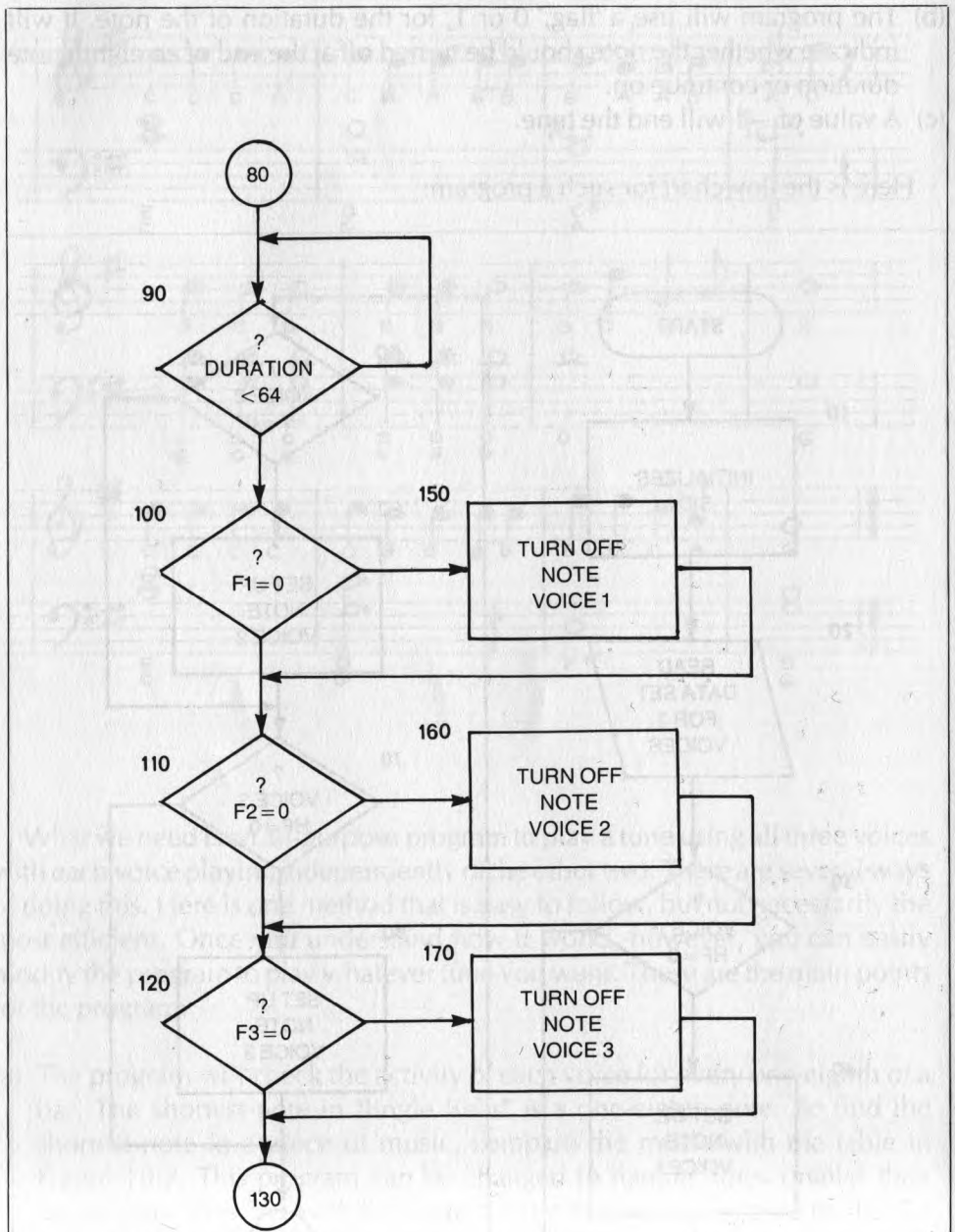
(a) The program will check the activity of each voice for every one-eighth of a bar. The shortest note in "Jingle Bells" is a one-eighth note. To find the shortest note in a piece of music, compare the music with the table in Figure 10-7. This program can be changed to handle notes smaller than one-eighth notes. We will discuss how to do this when we set up the DATA statements.

(b) The program will use a 'flag,' 0 or 1, for the duration of the note. It will indicate whether the note should be turned off at the end of an eighth note duration or continue on.
(c) A value of −1 will end the tune.

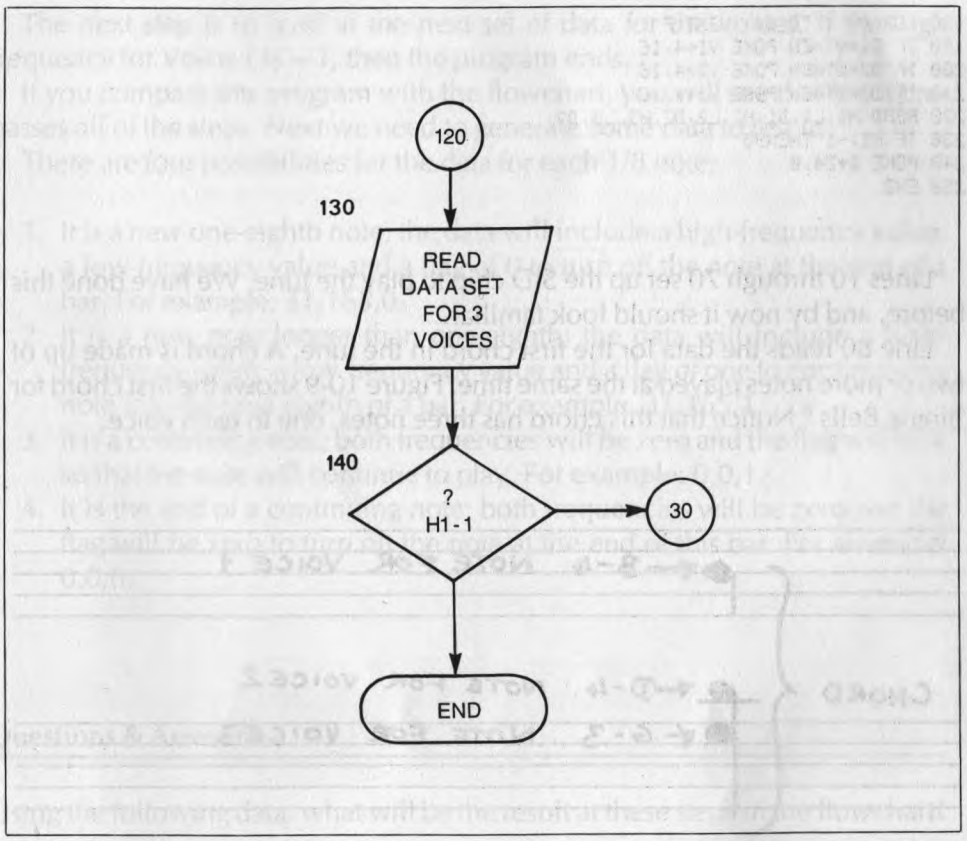
Here is the flowchart for such a program:



(continued)


80
90
? DURATION < 64
100
? F1 = 0
150
TURN OFF NOTE VOICE 1
110
? F2 = 0
160
TURN OFF NOTE VOICE 2
120
? F3 = 0
170
TURN OFF NOTE VOICE 3
130

(continued)



Now let's look at the program step by step. Here is the program listing:

```
10 S= 54272
20 FOR C = S TO S +24: POKE C,0:NEXTC
30 POKE S+24,15
40 V1 = S  : V2 = S+7 : V3 = S+14
50 POKE V1+5,0: POKE V1+6,240
60 POKE V2+5,0: POKE V2+6,240
70 POKE V3+5,0: POKE V3+6,240
80 READ H1,L1,D1,H2,L2,D2,H3,L3,D3
90 IF H1= 0 THEN120
100 POKE V1+1,H1: POKE V1,L1
110 POKE V1+4,17
120 IF H2=0 THEN150
130 POKE V2+1,H2:POKE V2,L2
140 POKE V2+4,17
150 IF H3=0 THEN180
160 POKE V3+1,H3:POKE V3,L3
170 POKE V3+4,17
```

(continued)

```
180 FOR P= 1 TO 64: NEXT P
190 IF D1=0THEN POKE V1+4,16
200 IF D2=0THEN POKE V2+4,16
210 IF D3=0THEN POKE V3+4,16
220 READ H1,L1,D1,H2,L2,D2,H3,L3,D3
230 IF H1>-1 THEN90
240 POKE S+24,0
250 END
```

Lines 10 through 70 set up the SID chip to play the tune. We have done this before, and by now it should look familiar.

Line 80 reads the data for the first chord in the tune. A chord is made up of two or more notes played at the same time. Figure 10-9 shows the first chord for "Jingle Bells." Notice that this chord has three notes, one in each voice.

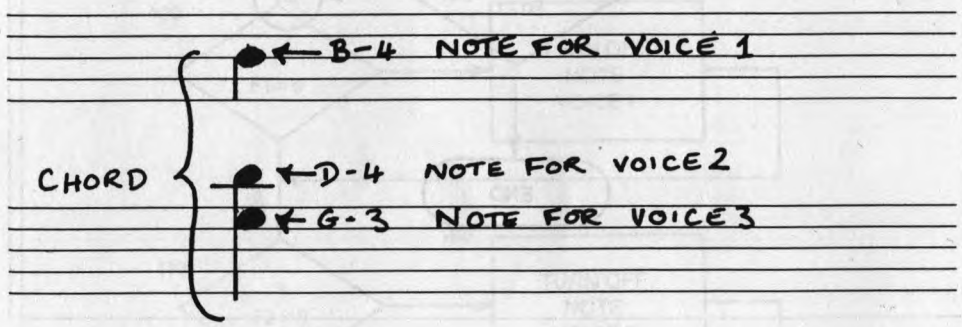


Figure 10.8 The first chord of "Jingle Bells"

Each note will have data for a high frequency (H1, H2, H3), low frequency (L1, L2, L3), and a flag (F1, F2, F3) that control the duration of the note. Each chord will have nine data values. If it is a chord that has only two notes, then only two voices will play, and the data for the third voice will be all zeroes.

Line 90 checks to see if the high frequency value of the note in Voice 1 is 0. If it is, the program moves to line 120. If it is not, lines 100 and 110 are executed; 110 sets the waveform, and 100 pokes the high and low frequencies for the note in voice 1.

Lines 120 through 140 and 150 through 170 perform the same functions for voice 2 and voice 3, respectively.

Line 180 is a loop set to the length of an eighth note, 126.

Lines 190 through 210 check the flag for each voice. If this flag is 0, then the voice waveform is turned off; otherwise, the note continues.

The next step is to read in the next set of data for the voices. If the high frequency for Voice 1 is −1, then the program ends.

If you compare this program with the flowchart, you will see that it encompasses all of the steps. Next we need to generate some data to test it.

There are four possibilities for the data for each 1/8 note:

1. It is a new one-eighth note; the data will include a high-frequency value, a low-frequency value and a flag of 0 to turn off the note at the end of a bar. For example: 31,165,0.
2. It is a new note longer than one-eighth; the data will include a high-frequency value, a low-frequency value and a flag of one to continue this note into the next eighth of a bar. For example: 31,165,1.
3. It is a continuing note; both frequencies will be zero and the flag will be 1 so that the note will continue to play. For example: 0,0,1.
4. It is the end of a continuing note; both frequencies will be zero and the flag will be zero to turn off the note at the end of this bar. For example: 0,0,0.

Questions & Answers

Using the following data, what will be the result at these steps in the flowchart?

	HF	LF	DR
Voice 1	31	165	0
Voice 2	1	0	1
Voice 3	0	0	0

(a) step 30
(b) step 70
(c) step 100
(d) step 110

Answer:

(a) The note for Voice 1 will be set.
(b) The note for Voice 3 will not be set.
(c) The note for Voice 1 will be turned off.
(d) The note in Voice 2 will continue and not be turned off.

Setting up the DATA statements is not difficult, but they need to be thought through carefully. You will need the table of frequencies in Appendix J and the table of comparative note lengths in Figure 10-5.

We have decided that the shortest note in "Jingle Bells" is a one-eighth note. We will divide each bar, the notes between the upright lines on the sheet music, into eight and write a set of data for each eighth of a bar. One data set will contain the high frequency, low frequency and flag for each voice. This means we will need 8 sets of data for each bar. There are 16 bars in the tune, so that means 128 pieces of data all together — a lot of data statements. It might be helpful to have someone else help proofread these statements.

The first chord in "Jingle Bells" is a quarter note. There is a B in Voice 1, a D in Voice 2 and a G in Voice 3. This chord will require two sets of data. The first eighth will turn the notes in the chord on, continue them through the second eighth, and then turn them off. Here is the first set of data to turn the chord on:

Voice 1	Voice 2	Voice 3
31,165,1	18,209,1	9,104,1

The high and low frequencies were taken from the frequency table in Appendix J. Notice that the flag in all three voices is 1, because the note extends beyond this eighth bar.

The second set of data for this chord looks like this:

```
0,0,0,0,0,0,0,0,0.
```

The frequencies in all three voices are zero, so that the notes in the chord will not change. The flags in all three voices are zero, so that the chord will be turned off at the end of this bar.

To compact the data a little, we will put two sets in each DATA statement, like this:

```
400 DATA 31,165,1,18,209,1,9,104,1,0,0,0,0,0,0,0,0,0.
```

The second chord in "Jingle Bells" is the same as the first, so the second DATA statement will be a repeat of the first.

The third chord has the same notes for each voice, but it is a half note.

Questions & Answers

How many sets of data will it require?

Answer: 4

The first one will look like this:

420 DATA 31,165,1,18,209,1,19,104,1.

The chord will continue. What will the high frequency be for each voice?

Answer: 0

What will the low frequency be for each voice?

Answer: 0

What will the flag be for each voice?

Answer: 1

The first two sets of data look like this:

420 DATA 31,165,1,18,209,1,19,104,1,0,0,1,0,0,1,0,0,1.

The chord continues through two more eighth periods, so we need two more sets of data. Here is the third set:

430 DATA 0,0,1,0,0,1,0,0,1.

The chord is turned off at the end of the fourth set of data. Which of the following sets of data is correct?

(a) 0,0,0,0,0,0,0,0,0.
(b) 0,0,1,0,0,1,0,0,1.

Answer:
(a) A flag of zero will turn off the notes in the chord.

The second bar, chords 4, 5, and 6, repeat the three chords in the first bar, so the DATA statements for this bar will be a repeat of those for the first bar.

The third bar uses only Voices 1 and 2, so the data for Voice 3 will be all zeros for the next 8 sets, or four statements. In this bar the second voice plays the same note, D3, for the whole bar. The note frequencies for the note in the second voice are set in the first set of data for the bar and the flag is 1. The frequency values for the second voice will be 0 for the remaining seven data sets, and the flag will be 1 until the last set, when it will be zero.

The first voice is playing the tune in this bar. Notice that the third note in Voice 1 is a dotted note. If you check the table of note values, you will see that this is worth three one-eighth notes, so it will continue through three sets of data. The last note in this bar for Voice 1 is an 'A' note. It is an eighth note, so it will be turned off after one period and will need only one set of data.

Here is the data for this bar:

```
480 DATA 31,165,1,18,209,1,0,0,0,0,0,0,0,0,1,0,0,0
490 DATA 42,62,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0
500 DATA 25,30,1,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0
510 DATA 0,0,0,0,0,0,0,0,0
```

Questions & Answers

Which of the following is the data for the last eighth note?

_____ (a) 28,49,1,0,0,1,0,0,0
_____ (b) 28,49,1,0,0,0,0,0,0
_____ (c) 28,49,0,0,0,1,0,0,0
_____ (d) 28,49,0,0,0,0,0,0,0

Answer: (d) Voice 1 plays a note for one-eighth, Voice 2 finishes at the end of this set and Voice 3 is off.

The data sets are coded in this way until the eighth bar. At the end of the eighth bar, there is a rest. It is represented by the squiggly lines in the music. The table in Figure 10-5 shows that this is a half-note rest. This means that all three voices are silent for four sets of data, or two DATA statements.

Here is a complete listing for the tune, but I hope you will try to work out the remaining DATA statements for yourself.

```
10 S= 54272
20 FOR C = S TO S +24: POKE C,0:NEXTC
30 POKE S+24,15
40 V1 = S  : V2 = S+7 : V3 = S+14
50 POKE V1+5,0: POKE V1+6,240
60 POKE V2+5,0: POKE V2+6,240
70 POKE V3+5,0: POKE V3+6,240
80 READ H1,L1,D1,H2,L2,D2,H3,L3,D3
90 IF H1= 0 THEN120
100 POKE V1+1,H1: POKE V1,L1
110 POKE V1+4,17
120 IF H2=0 THEN150
130 POKE V2+1,H2:POKE V2,L2
140 POKE V2+4,17
150 IF H3=0 THEN180
160 POKE V3+1,H3:POKE V3,L3
170 POKE V3+4,17
180 FOR P= 1 TO 64: NEXT P
190 IF D1=0THEN POKE V1+4,16
200 IF D2=0THEN POKE V2+4,16
210 IF D3=0THEN POKE V3+4,16
220 READ H1,L1,D1,H2,L2,D2,H3,L3,D3
230 IF H1>-1 THEN90
240 POKE S+24,0
250 END
400 DATA 31,165,1,18,209,1,9,104,1,0,0,0,0,0,0,0,0,0
410 DATA 31,165,1,18,209,1,9,104,1,0,0,0,0,0,0,0,0,0
420 DATA 31,165,1,18,209,1,9,104,1,0,0,1,0,0,1,0,0,1
430 DATA 0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0
440 DATA 31,165,1,18,209,1,9,104,1,0,0,0,0,0,0,0,0,0
450 DATA 31,165,1,18,209,1,9,104,1,0,0,0,0,0,0,0,0,0
460 DATA 31,165,1,18,209,1,9,104,1,0,0,1,0,0,1,0,0,1
470 DATA 0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0
480 DATA 31,165,1,18,209,1,0,0,0,0,0,0,0,0,1,0,0,0
490 DATA 42,62,1,0,0,1,0,0,0,0,0,0,0,1,0,0,0
500 DATA 25,30,1,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0
510 DATA 0,0,0,0,0,1,0,0,0,28,49,0,0,0,0,0,0,0
520 DATA 31,165,1,12,143,1,0,0,0,0,0,1,0,0,1,0,0,0
530 DATA 0,0,1,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0
540 DATA 0,0,1,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0
550 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
560 DATA 33,135,1,21,31,1,16,195,1,0,0,0,0,0,1,0,0,1
570 DATA 33,135,1,0,0,1,0,0,1,0,0,0,0,0,1,0,0,1
580 DATA 33,135,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1
590 DATA 0,0,0,0,0,1,0,0,1,33,135,0,0,0,0,0,0,0
600 DATA 33,135,1,21,31,1,16,195,1,0,0,0,0,0,1,0,0,1
610 DATA 31,165,1,0,0,1,0,0,1,0,0,0,0,0,1,0,0,1
```

(continued)

```
620 DATA 31,165,1,0,0,1,0,0,1,0,0,0,0,0,1,0,0,1
630 DATA 31,165,0,0,0,1,0,0,1,31,165,0,0,0,0,0,0,0
640 DATA 31,165,1,17,195,1,14,24,1,0,0,0,0,0,1,0,0,1
650 DATA 28,49,1,0,0,1,0,0,1,0,0,0,0,0,1,0,0,1
660 DATA 28,49,1,0,0,1,0,0,1,0,0,0,0,0,1,0,0,1
670 DATA 31,165,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0
680 DATA 28,49,1,18,209,1,16,195,1,0,0,1,0,0,1,0,0,1
690 DATA 0,0,1,0,0,1,0,0,1,0,0,0,0,0,1,0,0,1
700 DATA 42,62,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0
710 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
720 DATA 31,165,1,18,209,1,9,104,1,0,0,0,0,0,0,0,0,0
730 DATA 31,165,1,18,209,1,9,104,1,0,0,0,0,0,0,0,0,0
740 DATA 31,165,1,18,209,1,9,104,1,0,0,1,0,0,1,0,0,1
750 DATA 0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0
760 DATA 31,165,1,18,209,1,9,104,1,0,0,0,0,0,0,0,0,0
770 DATA 31,165,1,18,209,1,9,104,1,0,0,0,0,0,0,0,0,0
780 DATA 31,165,1,18,209,1,9,104,1,0,0,1,0,0,1,0,0,1
790 DATA 0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0
800 DATA 31,165,1,18,209,1,0,0,0,0,0,0,0,0,1,0,0,0
810 DATA 42,62,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0
820 DATA 25,30,1,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0
830 DATA 0,0,0,0,0,1,0,0,0,26,49,0,0,0,0,0,0,0
840 DATA 31,165,1,12,143,1,0,0,0,0,0,1,0,0,1,0,0,0
850 DATA 0,0,1,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0
860 DATA 0,0,1,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0
870 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
880 DATA 33,135,1,21,31,1,16,195,1,0,0,0,0,0,1,0,0,1
890 DATA 33,135,1,0,0,1,0,0,1,0,0,0,0,0,1,0,0,1
900 DATA 33,135,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1
910 DATA 0,0,0,0,0,1,0,0,1,33,135,0,0,0,0,0,0,0
920 DATA 33,135,1,18,209,1,9,104,1,0,0,0,0,0,1,0,0,1
930 DATA 31,165,1,0,0,1,0,0,1,0,0,0,0,0,1,0,0,1
940 DATA 31,165,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0
950 DATA 31,165,0,0,0,0,0,0,0,31,165,0,0,0,0,0,0,0
960 DATA 42,165,1,11,218,1,0,0,0,0,0,0,0,1,0,0,0
970 DATA 42,165,1,0,0,1,0,0,0,0,0,0,0,1,0,0,0
980 DATA 33,135,1,0,0,1,0,0,0,0,0,0,0,1,0,0,0
990 DATA 28,49,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0
1000 DATA 25,30,1,15,210,1,12,143,1,0,0,1,0,0,1,0,0,1
1010 DATA 0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1
1020 DATA 0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1
1030 DATA 0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0
```

You will find that this program will work for any music from Bach to the latest pop tune – with two changes.

First, you have to decide what is the shortest note in the tune. For instance, if the smallest note in a piece of music is a one-sixteenth note, then divide each bar into 16 and write 16 sets of data for each bar.

Second, the empty loop for the duration of the note in line 180 must be changed to 64 to reflect the shorter duration. Once the tune is written, you can use this loop to change its tempo. If you remember, we chose the numbers to

represent the lengths of the notes because they were easily divisible by 2 and 3. This meant that we could have a fairly accurate relationship between the relative lengths of the note. These numbers do not necessarily have to set the timing for the tune. Some tunes are played faster than others, and "Jingle Bells" definitely sounds better played at a faster speed. Change the counter in line 180 to 64 or 48, and see if you agree.

Questions & Answers

Why is the last DATA statement in our program a set of −1s?

Answer: To indicate the end of the tune. When −1 is encountered, the program leaves the main loop, turns off the SID chip and ends.

This is only one method of writing programs that will play a tune in harmony. There are many different approaches to the problem, and there have been several different algorithms published on how to do this. You may want to spend a little time and arrive at your own solution to the problem.

Sound Effects

Before we leave the SID chip, we should talk about sound effects. As you have seen already, the SID chip has amazing range and versatility. In this section we are going to discuss how to use the SID chip to create sound effects. You will find many listings for sound effects in computer magazines and books, so rather than providing a library of sounds, we will develop some ideas for creating our own.

We have not used the noise waveform yet, and it is ideal for certain effects, such as explosions, footsteps and thunderclaps. Changing the frequencies does not have much effect on the noise waveform, but we can change the volume of the noise. We have always set the volume on the SID chip to 15, which is the maximum value, but it can be any number from 0 to 15.

Here is a program that sounds like a wave at the beach. A FOR-NEXT loop increasing the volume creates the effect.

```
10 S= 54272
15 FOR T = 1 TO 6
20 FOR V = 1 TO 15
30 POKE S+24,V
40 POKE S+4,129
50 POKE S+1,8: POKE S,97
60 FOR P = 1 TO 50: NEXT P
70 NEXT V
120 NEXT T
130 POKE S+4,128: POKE S+24,0
140 END
```

Questions & Answers

How could you add to the program to make the sound fade away completely?

Answer: Use a FOR-NEXT loop decreasing from 15 to 0 like this:

```
10 S= 54272
15 FOR T = 1 TO 6
20 FOR V = 1 TO 15
30 POKE S+24,V
40 POKE S+4,129
50 POKE S+1,8: POKE S,97
60 FOR P = 1 TO 50: NEXT P
70 NEXT V
80 FOR V = 15 TO 1 STEP -1
90 POKE S+24,V
100 POKE S+1,8: POKE S,97
110 FOR P = 1 TO 100: NEXT P
120 NEXT V
130 NEXT T
140 POKE S+4,128: POKE S+24,0
150 END
```

If we put the complete 'wave' into a loop, then we can make a series of waves.

One technique for producing a siren-like sound is to use a loop to sweep up and down the frequency table. The frequencies given in the table are the closest approximation for that particular note, but any number between 1 and 255 will produce a sound.

```
10 S= 54272
20 POKE S+24,15
30 FOR T = 1 TO5
40 POKE S+4,17
50 FOR V = 1 TO 255
60 POKE S+1,V
70 POKE S,V
80 NEXT V
85 FOR V= 255 TO 1 STEP -1
87 POKE S+1,V: POKE S,V
89 NEXT V
90 POKE S+4,16
100 FOR P = 1 TO 100: NEXT P
110 NEXT T
```

Questions & Answers

Which waveform is being used in the program?

Answer: Triangle.

Try changing this waveform and see what happens.

A random function is useful in making sound effects, such as birds chirping or this example of 'alien speech.'

```
10 S= 54272
20 FOR L = S TO S+24: POKE L,0:NEXT L
30 POKE S+24,15
40 POKE S+5,63: POKE S+6,96:POKE S+12,96:POKE S+13,93
50 FOR C = 1 TO 20
60 POKE S+4,17: POKE S+11,17
70 M = INT(RND(1)*30)+100
80 POKE S+1,M: POKE S,M: POKE S+8,M+8: POKE S+7,M+8
90 FOR D = 1 TO M : NEXT D
100 POKE S+4,16: POKE S+11,16
110 NEXT C
120 POKE S+24,0
```

The random statement selects the high and low frequencies for the voices.

Questions & Answers

What range will the random number be in?

Answer: 100 to 300.

How many voices are used for this sound effect?

Answer: 2

I hope that this has given you some ideas for experimenting with your own sound effects.

There is more to programming the SID chip than we have covered in this book. After you have become confident in creating music and sound effects with the features covered in this chapter, you might like to explore some of the advanced features on the SID chip for yourself.

Please work the Self-Check before moving on to Chapter 11.

SELF-CHECK

1. Are these statements true or false?
   (a) A change in the volume of a sound is caused by a change in its pitch.
   (b) The four waveforms available on the SID chip are triangle, sawtooth, square and random noise form.
   (c) The SID chip controls four voices.
   (d) The ADSR settings are coded using two bytes for each voice on the SID chip.
2. (a) Which waveform uses pulse settings?
   (b) How many locations on the SID chip are used for pulse settings?
3. What is the maximum value the AD and SR locations can have?
4. (a) What is the maximum volume setting?
   (b) What is the relative setting for the volume on the SID chip?

5. How many frequencies are there for each note?
6. (a) The wave sound effect is created by varying the _____ of the sound.
   (b) The siren effect is created by varying the _____ of the sound.

Answers to SELF-CHECK

1. (a) False.
   (b) True.
   (c) False.
   (d) True.
2. (a) Square or pulse wave.
   (b) 6 (two for each voice).
3. 255
4. (a) 15
   (b) S+24
5. 2
6. Volume, frequency.

Suggestions for Further Exploration with Sound

It is difficult to recommend only one or two ideas for you to try next. However, I can suggest creating your own music and sound effects to add to the programs you have already written.

The program written in this chapter for playing any tune represents just one approach to the problem. Now that you understand the various parts of a sound program, you can design your own multi-purpose sound program.

CHAPTER 11

Sprites

Some would say that we have saved the best for last. The sprite graphics feature on the Commodore 64 is an exciting and versatile graphics option. A sprite is a user-designed object that can be moved around the screen independently of the background. There can be up to eight sprites on the screen at one time, each moving independently of the others, passing in front or behind each other. This gives the display a three-dimensional effect and allows us to create interesting programs.

Sprites may be used with any other graphics feature, such as screen and border colors, poking characters using the screen and color memory maps, or using the CHR$ function in PRINT statements. These graphics features were covered in Chapter 9 and you may want to review them before you enter the wonderful world of sprites.

By the end of this chapter you will be able to:

- Understand the function of the memory locations for the sprite features on the VIC II graphics chip.
- Create a sprite.
- Store and retrieve sprites as datafiles.
- Move a sprite around the screen.
- Move more than one sprite around the screen.
- Change the color of a sprite.
- Expand a sprite horizontally and vertically.
- Animate a sprite.
- Use the collision identification in a program.

The VIC II Chip

We have already learned all the BASIC statements we will need to program sprites. In addition, having used the SID chip to make music in Chapter 10, understanding the memory map for the VIC II graphics chip will be much easier. The complete map for the VIC II chip is shown in Appendix I. You can see that we have already used two of the locations on the VIC II chip to change the screen and border colors. In this chapter, we will look at the problem of programming sprites. Each sprite location on the chip will be explained and used as it is needed.

Questions & Answers

The two BASIC statements we will use with the VIC II locations are POKE and PEEK. Match the statements to their activity:

(a) PEEK — 1. returns the value store at a specific location.
(b) POKE — 2. stores a given value at a specific location.

Answer: (a) 1 (b) 2

Creating and Using Sprites

The first step in using a sprite is to create it. Unlike the graphics character set, which is part of the Commodore 64's operating system, sprites are designed and created by the user. The size of a sprite is limited to 24 dots, or "pixels," in width and 21 dots in height. A grid with these dimensions will look like Figure 11.1 on the next page.

Notice that each row of the grid is divided into three groups of eight squares. Each group of eight squares is coded by one byte in the computer memory. In Chapter 2 we said that a byte was made up of eight bits. Each bit can be either 'on' or 'off.' As each bit in a byte can be represented by a power of 2, maximum decimal value that can be represented in a byte is 255.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	1	1	1	1

128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255

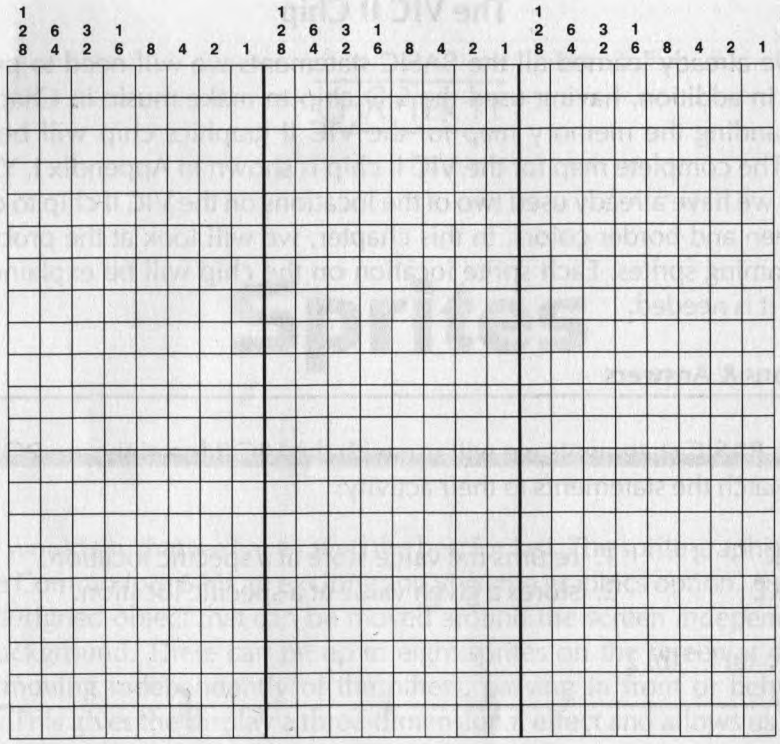


Figure 11.1 A 24 * 21 grid for creating sprites.

This arrangement should be familiar to you if you worked through the examples using the SID chip in Chapter 10. The decimal equivalent for each bit is written across the top of the grid, and it is repeated for each byte.

Now let's use the grid to draw a sprite. A musical quarter note, as shown in Figure 11.2, is a good shape for our first sprite.

We must turn the drawing into numbers that represent which bits in which byte should be turned on. We can then use these numbers in DATA statements. To do this add up the numbers associated with the shaded squares in each byte for each row. This means that each row will be coded in three numbers, and for each sprite, there will be twenty-one sets of three numbers (Figure 11-3).

The quarter note is drawn in the middle of the grid, and all the shaded areas are in the middle eight columns. This means that for each row the first and third

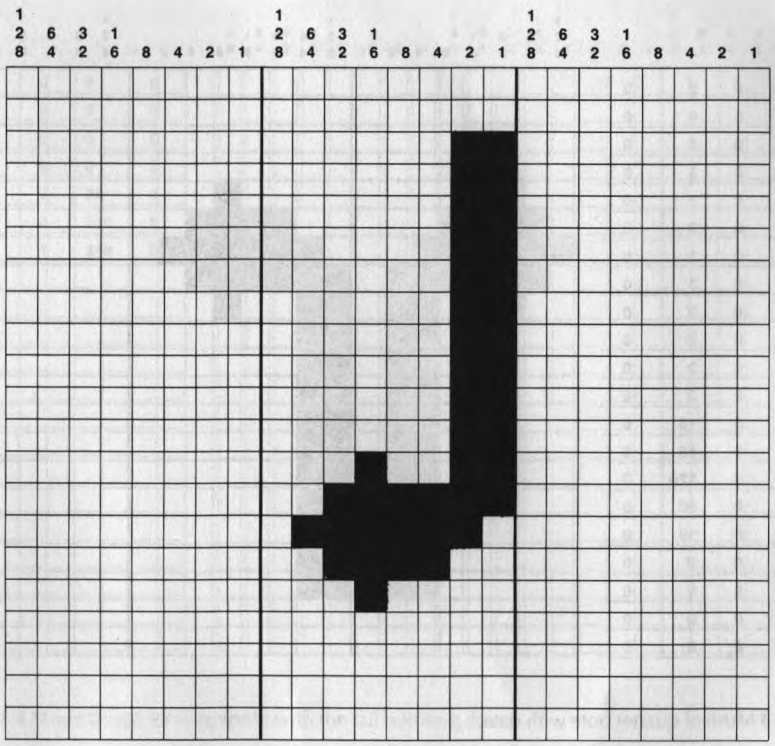


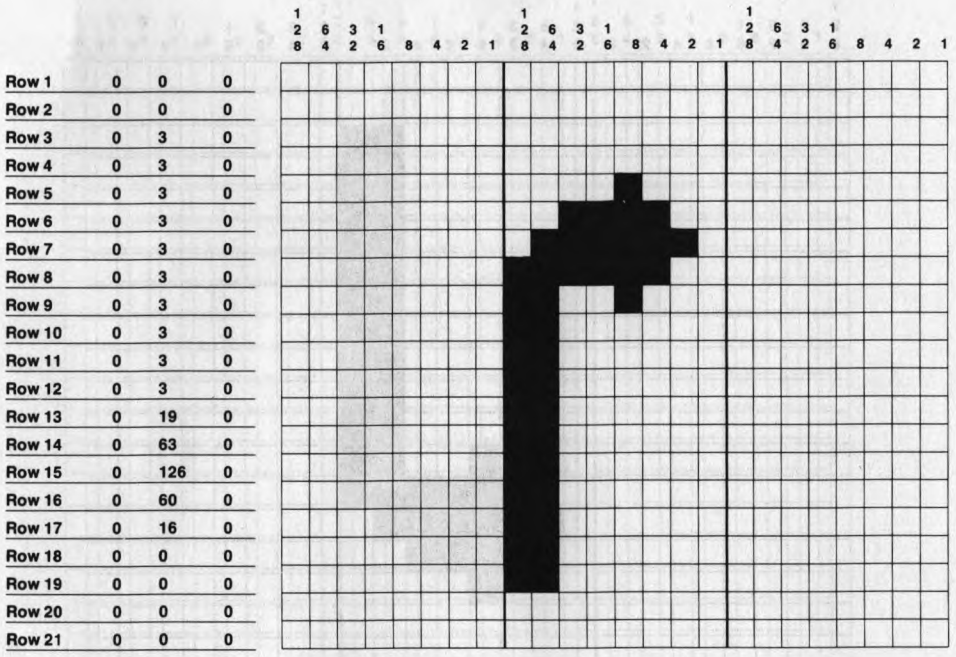
Figure 11.2 Diagram for a musical quarter note sprite.

bytes will be zero, and the note will be coded in the second byte. The first two rows are blank so all three numbers are 0.

0,0,0
0,0,0

The second byte in the third row has the two right-hand squares shaded. The number that represents that these two bits are "on" (shaded) is 2 + 1. So the third row is coded:

0, 3, 0



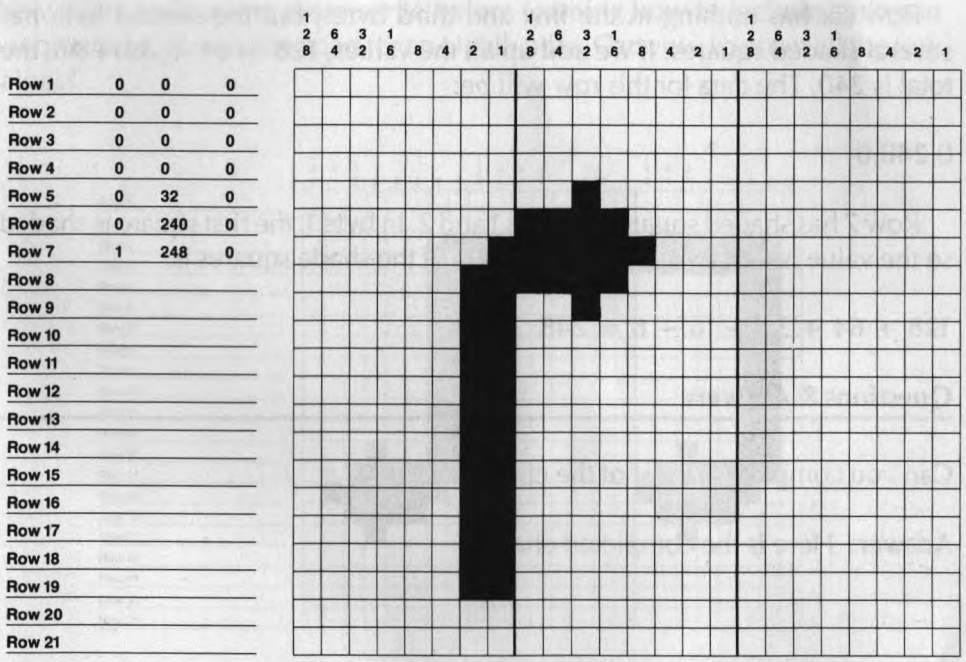
Row 1	0	0	0
Row 2	0	0	0
Row 3	0	3	0
Row 4	0	3	0
Row 5	0	3	0
Row 6	0	3	0
Row 7	0	3	0
Row 8	0	3	0
Row 9	0	3	0
Row 10	0	3	0
Row 11	0	3	0
Row 12	0	3	0
Row 13	0	19	0
Row 14	0	63	0
Row 15	0	126	0
Row 16	0	60	0
Row 17	0	16	0
Row 18	0	0	0
Row 19	0	0	0
Row 20	0	0	0
Row 21	0	0	0

Figure 11.3 Musical quarter note with data.

The next nine rows are the same as the third row, 0,3,0. These rows make up the "tail" of the note. Row 13 starts the "head" of the note. Here is an expanded version of the code for the next five lines. The middle number is calculated by adding together the decimal equivalent of each bit that is "on."

```
0, 16+2+1, 0          = 0, 19, 0
0, 32+16+8+4+2+1, 0   = 0, 63, 0
0, 64+32+8+4+2, 0     = 0, 126, 0
0, 32+16+8+4, 0       = 0, 60, 0
0, 16, 0              = 0, 16, 0
```

Figure 11.4 is the grid for a sprite quarter-note with the tail hanging down.



Row 1	0	0	0
Row 2	0	0	0
Row 3	0	0	0
Row 4	0	0	0
Row 5	0	32	0
Row 6	0	240	0
Row 7	1	248	0
Row 8			
Row 9			
Row 10			
Row 11			
Row 12			
Row 13			
Row 14			
Row 15			
Row 16			
Row 17			
Row 18			
Row 19			
Row 20			
Row 21			

Figure 11.4 Musical quarter note sprite with the tail pointing down.

Questions & Answers

Is the note drawn in the second byte only?

Answer: No, it is drawn in the first and second bytes.

Let's code this one step at a time. The first four rows are all zero. In row five, what are the values for the following bytes?

(a) Byte 1
(b) Byte 2
(c) Byte 3

Answer: (a) 0 (b) 32 (c) 0

Row six has nothing in the first and third bytes, but the second byte has several shaded squares. If we add up all the values, 128 + 64 + 32 + 16, the total is 240. The data for this row will be:

0,240,0

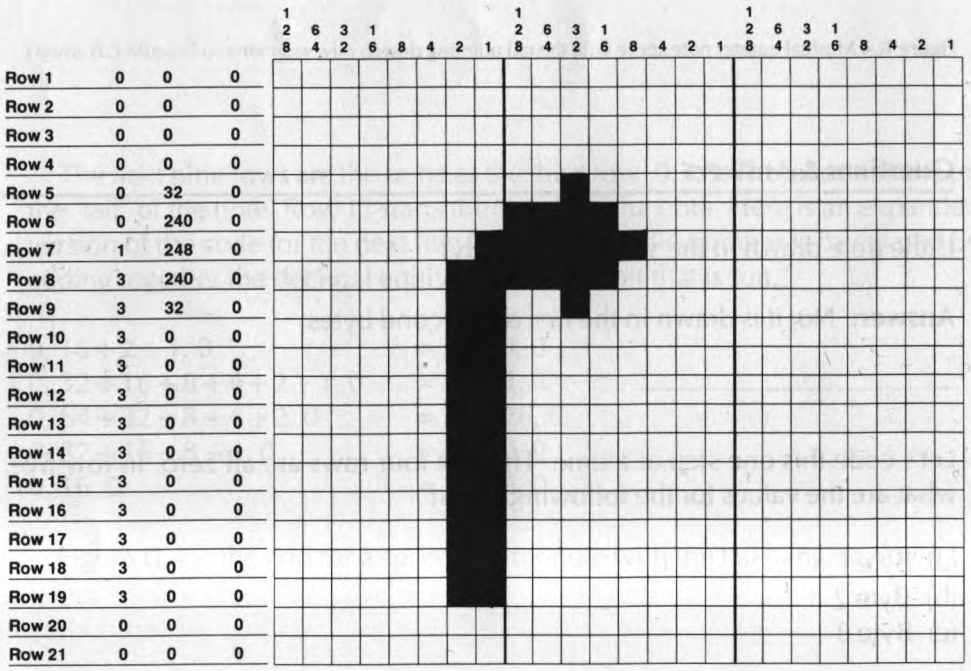
Row 7 has shaded squares in bytes 1 and 2. In byte 1, the first square is shaded so the value is 1. In byte 2, the total value of the shade squares is:

128 + 64 + 32 + 16 + 8 = 248

Questions & Answers

Can you complete the rest of the chart?

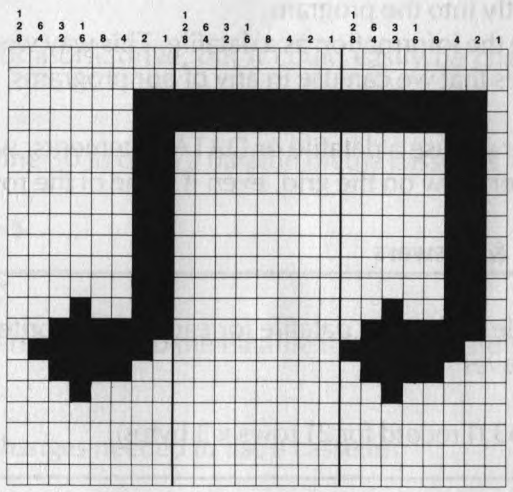
Answer: Here is the completed chart:



Row 1	0	0	0
Row 2	0	0	0
Row 3	0	0	0
Row 4	0	0	0
Row 5	0	32	0
Row 6	0	240	0
Row 7	1	248	0
Row 8	3	240	0
Row 9	3	32	0
Row 10	3	0	0
Row 11	3	0	0
Row 12	3	0	0
Row 13	3	0	0
Row 14	3	0	0
Row 15	3	0	0
Row 16	3	0	0
Row 17	3	0	0
Row 18	3	0	0
Row 19	3	0	0
Row 20	0	0	0
Row 21	0	0	0

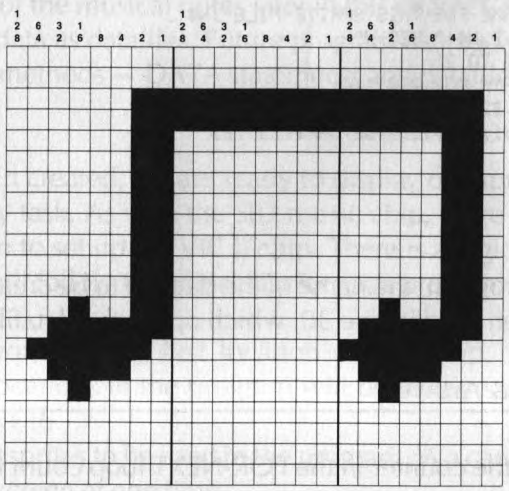
We will try coding one more sprite before learning how to include sprites in our programs. Here is the chart for an eighth note. Can you work out all the row values?

Row 1			
Row 2			
Row 3			
Row 4			
Row 5			
Row 6			
Row 7			
Row 8			
Row 9			
Row 10			
Row 11			
Row 12			
Row 13			
Row 14			
Row 15			
Row 16			
Row 17			
Row 18			
Row 19			
Row 20			
Row 21			



Answer: Here is the completed chart:

Row 1	0	0	0
Row 2	0	0	0
Row 3	3	255	254
Row 4	3	255	254
Row 5	3	0	6
Row 6	3	0	6
Row 7	3	0	6
Row 8	3	0	6
Row 9	3	0	6
Row 10	3	0	6
Row 11	3	0	6
Row 12	3	0	6
Row 13	19	0	38
Row 14	63	0	126
Row 15	126	0	252
Row 16	60	0	120
Row 17	16	0	32
Row 18	0	0	0
Row 19	0	0	0
Row 20	0	0	0
Row 21	0	0	0



Now that we know how to code information to create a sprite, we need to think about ways to include it in a program. There are two ways to do this:

1. Include the information in DATA statements in the program to be READ directly into the program.
2. Store the information as a datafile. This way we can build up a "library" of sprites that we can use in any of our programs.

Whether we use a datafile or DATA statements, we will always include the data for every row on the grid, even if some of the rows are all zeroes.

Questions & Answers

If we decide to create a datafile for each of our sprites, how many records will each file have?

Answer: 63 (1 record for 21 rows x 3 bytes)

If you need help with either creating a datafile or retrieving information from a datafile, you may want to review Chapter 8.

Here is an example of a program to create a datafile for any sprite.

```
10 REM DATAFILE FOR A SPRITE
20 INPUT "NAME FOR THIS SPRITE FILE";N$
30 OPEN 2,8,2,N$+",SEQ,W"
40 FOR C = 1 TO 21
50 PRINT "DATA FOR ROW ";C
60 INPUT B1,B2,B3
70 PRINT#2,B1,CHR$(13),B2,CHR$(13),B3
80 NEXT C
90 CLOSE 2
100 END
```

In this program, the name of the datafile to be created is INPUT in line 20 as N$. It is then used in line 30, which opens the datafile.

Questions & Answers

Why does the counter in the FOR-NEXT loop count from 1 to 21?

Answer: There are 21 rows in the sprite map.

What goes in the variables B1, B2, and B3?

Answer: The value for the three bytes in each row.

This program is written for a disk drive, but it could easily be changed for a cassette.

How would you change line 30 to open a datafile on the cassette?

Answer: 30 OPEN 1,1,1,N$

Don't forget to change all references to the datafile to the same number as the OPEN statement.

What are the remaining changes needed to use a cassette?

Answer:

```
70 PRINT#1, B1; CHR$(13); B2; CHR$(13); B3
90 CLOSE 1
```

We will be using each of the musical notes later in this chapter, so you may want to save these sprite data as datafiles. Future program listings will include references for using both methods — DATA statements and datafiles.

Displaying Sprites

Once the datafile has been created, we are ready to display our sprites on the screen. This is not an easy task. As with the SID music chip, there are certain steps that have to be taken to set up the VIC II chip. There is a logical order to any sprite program. We will follow this check list for each program:

1. Set a variable, V, equal to the first location on the chip, 53248. All references to chip locations in the program will be relative to this position, for example, V+16.
2. Turn on the sprite or sprites to be used in the program. We can have up to eight sprites on the screen at one time.

3. Set the sprite pointer. This tells the computer where the sprite data is located in memory.
4. Load the sprite data into memory.
5. Set the horizontal, Y, and vertical, X, position on the screen.

Once we have performed these five steps, we have several options:

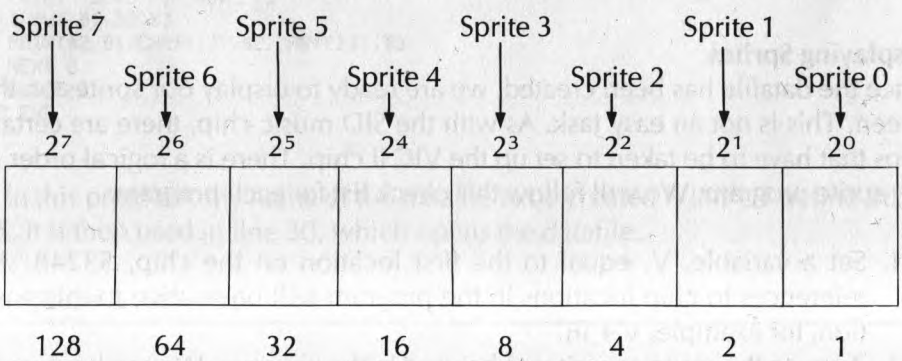
- change the color of the sprite.
- move the sprite around the screen.
- expand the sprite in one or two directions.
- check for sprite-sprite or sprite-background collisions.

We will develop a program to display one of the musical-note sprites that we have just created. The first step in the program is an assignment statement to set the variable, V, equal to the first location on the VIC II chip. The first location on the VIC II chip is 53248, so the first line looks like this:

```
10 V = 53248: REM START VIC II
```

The second step is a little more complicated. The location to enable, or turn on the sprites is 53269, or V+21. This location is used to turn on all eight sprites. Each bit in this location represents one sprite. To turn on a sprite, we need to "turn on" the representative bit.

The eight sprites are numbered from 0 to 7. This chart shows which bits represent them, and the number that must be POKEd to turn on that bit.



To turn on sprite 0, we use this line:

```
20 POKE V + 21, 1
```

Questions & Answers

How would you turn on sprite 4?

Answer: 20 POKE V + 21, 16

To turn off a sprite, turn off the bit representing that sprite. The value POKEd is the difference between 255 and the value used to turn on the sprite.

This line will turn off sprite 0:

```
20 POKE V + 21,254
```

Questions & Answers

Which sprite will this line turn off?

```
20 POKE V + 21, 191?
```

Answer: Sprite 6.

If we want more than one sprite on the screen at a time, then the values for the sprites are added and this value is included in the POKE statement. For example, the values to turn on sprites 7 and 4 are 128 and 16, so the POKE statement to turn them both on would look like this:

```
20 POKE V + 21, 144
```

Questions & Answers

Which of the following statements will turn on sprites 0, 2, 4?

_____ (a) POKE V + 21,7?
_____ (b) POKE V + 21,14?
_____ (c) POKE V + 21, 21?

Answer: (c)

The next step in the program is to read the data for the sprite into memory, either from DATA statements or from a datafile. To keep track of where the sprite data is, and to keep the right data with the right sprite, a pointer system is used. The pointer must be set before the data is loaded into memory. The sprite pointer locations are not on the VIC II chip. Here are the pointer locations:

Sprite #	Location
0	2040
1	2041
2	2042
3	2043
4	2044
5	2045
6	2046
7	2047

Working out the value to poke at these locations is complicated. In this book we will use locations in memory, starting at location 12288, to store sprite data. If you look at the memory map for the Commodore 64 in Chapter 7, you will see that this is a part of the memory that is free for use. This is also the area where our programs may be stored, so we need to be careful that the sprite data is not written over part of the program. The size of the programs we will write in this book will not cause this kind of problem, but you should worry about it if you write large programs.

Any area in the memory that is free can be used to store sprite data. Some programs use locations 832 – 1024. This is the cassette buffer, and I do not recommend that you use this area, especially if you are storing your programs on a cassette.

Each sprite takes up 63 bytes for data storage, plus 1 byte as a marker at the end of the sprite data. Memory locations are divided into groups of 64 bytes. The number stored at the sprite pointer location is the first location for the sprite data divided by 64. Here is a table with the values for the POKE statements calculated for eight sprites, starting with location 12288:

Starting Location for Data Storage	Value to POKE in Pointer Location
12288	12288/64 = 192
12352	12352/64 = 193
12416	12416/64 = 194
12480	12480/64 = 195
12544	12544/64 = 196
12608	12608/64 = 197
12672	12672/64 = 198
12736	12736/64 = 199

In our musical sprite program we will use sprite 0 and store the data starting at location 12288. The statement to set the pointer to the start of the data locations will look like this:

```
30 POKE 2040,192
```

Questions & Answers

How would you rewrite this line to set the pointer for sprite 2 to location 12288?

Answer: 30 POKE 2042,192

If the program had two sprites, 0 and 1, we might code the pointers like this:

```
30  POKE 2040,192
40  POKE 2041,193
```

Line 30 uses pointer location 2040, for sprite 0, and the starting location for the data will be 192*64=12288. Line 40 uses pointer location 2041, for sprite 1, and the starting location will be 193*64=12352.

Questions & Answers

Look at this statement:

```
POKE 2044,196
```

(a) What is the starting location for the sprite data?
(b) Which sprite pointer is this?

Answer:
(a) 12544
(b) Sprite 4

To review, this is the start of our sprite program to display our musical sprite using sprite 0:

```
10 V   = 53248: REM START VIC II CHIP
20 POKE V + 21,2: REM TURN ON SPRITE 0
30 POKE 2040,192: REM SPRITE 0 POINTER TO LOCATION 12288
```

If we have the sprite data as DATA statements, we will need a FOR-NEXT loop to read it into memory.

```
40 FOR C = 0 TO 62
50 READ D
60 POKE 12288 + C, D
70 NEXT C
```

The loop reads all 63 elements. (Don't forget that 0 counts!) Line 60 contains the starting location for the data and is incremented by the counter each time it is executed.

Look at this example:

```
30 POKE 2041, 193
40 FOR C = 0 TO 62
50 READ D
60 POKE . . ., . . . .
70 NEXT C
```

Questions & Answers

How would you complete line 60? (Hint: check the value POKEd in line 30.)

Answer:

60 POKE 12352 + N, D

(12352 is the starting location pointed at in line 30 (193* 64 = 12352))

The last step in displaying a sprite is to set the X and Y location on the screen. To do this we use a screen map. We have worked with screen maps before, which had 40 vertical columns and 24 horizontal rows. The sprite screen map is arranged differently and there is part of the sprite screen map that is actually off the screen. This seems odd, but it is very useful; the sprite can gradually move on to the screen, instead of suddenly appearing at the side of the screen. Figure 11.8 shows the sprite screen map.

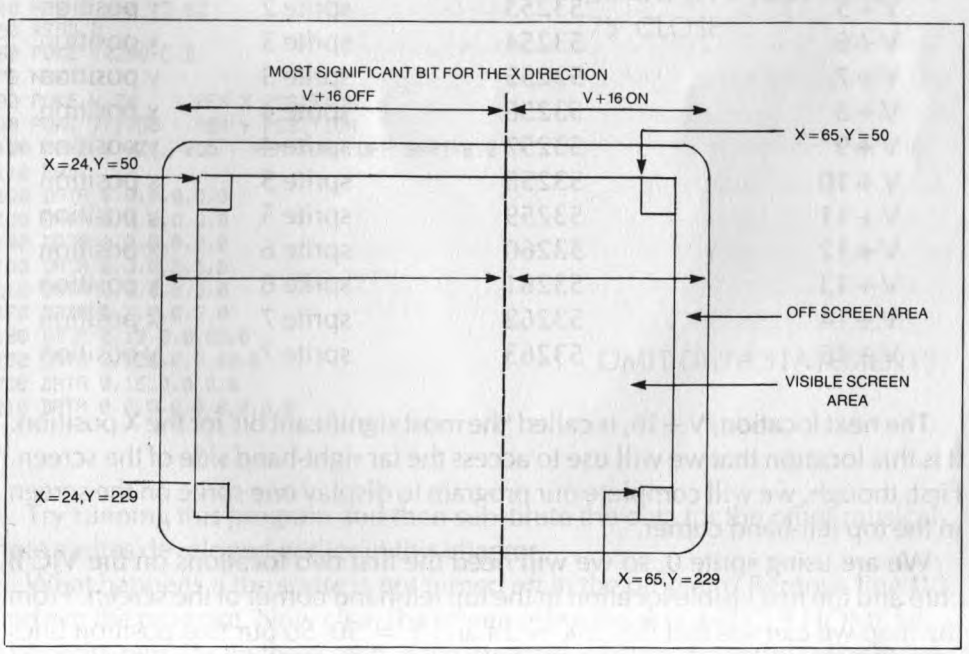


Figure 11.5 Sprite screen map.

The Y direction is the vertical direction and is the equivalent of the columnson the graphics map. Each sprite uses one location on the VIC II chip for its position in the Y direction. The X direction is the horizontal direction and is equivalent to the rows on the graphics map. Each sprite also has one location on the VIC II chip to store its position in the X direction. This direction is complicated by the fact that there are more than 255 positions. As the largest number that can be stored in one byte is 255, positioning sprites on the far right of the screen will require more than 1 byte. We will find out how the VIC II chip arranges this later in this chapter.

The first sixteen locations on the VIC II chip are the X and Y positions for the eight sprites.

V	53248	sprite 0	x position
V+1	53249	sprite 0	y position
V+2	53250	sprite 1	x position
V+3	53251	sprite 1	y position
V+4	53252	sprite 2	x position
V+5	53253	sprite 2	y position
V+6	53254	sprite 3	x position
V+7	53255	sprite 3	y position
V+8	53256	sprite 4	x position
V+9	53257	sprite 4	y position
V+10	53258	sprite 5	x position
V+11	53259	sprite 5	y position
V+12	53260	sprite 6	x position
V+13	53261	sprite 6	y position
V+14	53262	sprite 7	x position
V+15	53263	sprite 7	y position

The next location, V+16, is called 'the most significant bit' for the X position. It is this location that we will use to access the far right-hand side of the screen. First, though, we will complete our program to display one sprite on the screen in the top left-hand corner.

We are using sprite 0, so we will need the first two locations on the VIC II chip and the first visible location in the top left-hand corner of the screen. From the map we can see that this is X = 24 and Y = 50. So our two position lines will look like this:

```
80 POKE V,24: REM SPRITE 0 X POSITION
90 POKE V + 1,50: REM SPRITE 0 Y POSITION
```

Questions & Answers

Which locations on the VIC II chip would you use for the X and Y positions for sprite 7?

Answer: V+14 and V+15

There are two things remaining to be added to our musical-note sprite program. The sprite must be turned off and the DATA statements added at the end. Here is the complete program, including the data for the (down) quarter note.

```
5 PRINT CHR$(147)
10 V= 53248  : REM START VICII
20 POKE V+21,1  : REM TURN ON SPRITE 0
30 POKE 2040,192 : REM SPRITE 0 POINTER
40 FOR C = 0 TO 62
50 READ D
60 POKE 12288+C,D
70 NEXT C
80 POKE V,24   : REM X POSITION
90 POKE V+1,50 : REM Y POSITION
100 POKE V +21, 255 : REM TURN OFF SPRITE 0
110 END
120 DATA 0,0,0,0,0,0
130 DATA 0,3,0,0,3,0
140 DATA 0,3,0,0,3,0
150 DATA 0,3,0,0,3,0
160 DATA 0,3,0,0,3,0
170 DATA 0,3,0,0,3,0
180 DATA 0,19,0,0,63,0
190 DATA 0,126,0,0,60,0
200 DATA 0,16,0,0,0,0
210 DATA 0,0,0,0,0,0,0,0,0
```

35 OPEN STATEMENT
50 INPUT# NOT READ
75 CLOSE

OMIT DATA STATEMENTS

Try running this program and then substitute the data for the other musical-note sprites developed earlier in this chapter.

What happens if the sprite is not turned off in the program? Remove line 110 and run the program. Now clear the screen using the shift and CLR HOME key. The sprite remains on the screen. If the program is listed, the sprite still remains and appears to be in front of the listing. The sprites should always be turned off at the end of a program.

Questions & Answers

Which of these statements will position the sprite 0 at the bottom left-hand corner of the visible screen?

_____ (a) POKE V,24: POKE V+1,249
_____ (b) POKE V,0: POKE V+1, 255
_____ (c) POKE V,50: POKE V+1,250

Answer: (a)

Moving the Sprites

Now we will move the sprite around the screen. To do this, we will use positions taken from the screen map and FOR-NEXT loops. For example, to move the sprite down the screen we would replace line 90 with this loop:

```
90 FOR Y = 0 TO 255
94 POKE V+1,Y
98 NEXT Y
```

Try this substitution and notice how the sprite gradually moves onto the screen at the top and disappears at the bottom. This effect is caused by movement across the areas of the screen map that are not visible on the screen.

We can substitute a FOR-NEXT loop for the x position of sprite 0 this way:

```
80 FOR X = 0 TO 255
84 POKE V, X
88 NEXT X
```

Substitute these lines for line 80 and replace line 90 to its original form:

```
90 POKE V+1,50
```

The sprite moves across the screen and stops before it reaches the right-hand side.

Questions & Answers

Why can't the x counter in line 80 be increased?

Answer: 255 is the highest number that can be stored at one location.

To make the sprite travel all the way across the screen, we must use another location, V+16, the most significant bit for the x position. This location is used for all eight sprites, so we will turn on the relevant bit for sprite 0 by poking a value of 1 at that location.

These lines will move sprite 0 all the way across the screen.

```
80 POKE V+1, 50
85 FOR X = 0 TO 255; POKE V,X: NEXT X
90 POKE V+16,1: FOR X = 0 TO 91: POKE V,X: NEXT X
```

To start again at the left-hand side of the screen or move back to the left-hand side of the screen, the most significant bit must be turned off.

Questions & Answers

How would you turn off the most significant bit for sprite 0?

Answer: POKE V+16, 254.

Here is a listing for a program that moves our quarter note back and forth across the screen.

```
5 PRINT CHR$(147)
10 V= 53248  : REM START VICII
20 POKE V+21,1  : REM TURN ON SPRITE 0
30 POKE 2040,192 : REM POINTER SPRITE 0
40 FOR C = 0 TO 62
50 READ D
60 POKE 12288+C,D
70 NEXT C
72 FOR P = 1 TO 10 : REM TEN TIMES BACK & FORTH
74 POKE V+1,50  : REM Y POSITION
76 FOR X = 0 TO 255 : REM LOOP FOR X POSITION MOVING TO THE RIGHT
80 POKE V,X
82 NEXT X
```

DATAFILE CHANGES

35 OPEN STATEMENT
50 INPUT# NOT READ
75 CLOSE

(continued)

```
84 POKE V+16,1 : REM MSB ON
86 FOR X = 0 TO 91
90 POKE V,X
92 NEXT X
94 FOR X = 91 TO 0 STEP -1 : REM LOOP X POSITION TO MOVING TO THE LEFT
96 POKE V,X
98 NEXT X
100 POKE V+16,254  : REM MSB OFF
102 FOR X = 255 TO 0 STEP -1
104 POKE V,X
106 NEXT X
108 NEXT P
110 POKE V+21,254  : REM SPRITE 0 OFF
120 END
130 DATA 0,0,0,0,0,0
140 DATA 0,3,0,0,3,0
150 DATA 0,3,0,0,3,0
160 DATA 0,3,0,0,3,0
170 DATA 0,3,0,0,3,0
180 DATA 0,3,0,0,3,0
190 DATA 0,19,0,0,63,0
200 DATA 0,126,0,0,60,0
210 DATA 0,16,0,0,0,0
220 DATA 0,0,0,0,0,0,0,0,0
```

OMIT DATA STATEMENTS IF USING A DATAFILE

This program is listed so that you can clearly see what is happening at each line. However, sprite programs are ideal programs for streamlining, or "crunching." In Chapter 7 we discussed ways of compacting and speeding up programs once they were debugged.

Questions & Answers

Can you think of two things you could do to crunch this program?

Answer:

1. Remove the REM statements.
2. Put multiple statements on a line.

Here is a crunched version of the program:

```
5 PRINTCHR$(147)
10 V= 53248: REM START VIC II
20 POKE V+21,7
30 POKE 2040,192: POKE 2041,192: POKE 2042,192
40 FOR C = 0 TO 62
```

(continued)

```
50 READ D
60 POKE 12288+C,D
70 NEXT C
80 POKE V,24 : POKE V+2,24 : POKE V+4,24
90 POKE V+1,50 : POKE V+3,70 : POKE V+5,90
110 END
130 DATA 0,0,0,0,0,0
140 DATA 0,3,0,0,3,0
150 DATA 0,3,0,0,3,0
160 DATA 0,3,0,0,3,0
170 DATA 0,3,0,0,3,0
180 DATA 0,3,0,0,3,0
190 DATA 0,19,0,0,63,0
200 DATA 0,126,0,0,60,0
210 DATA 0,16,0,0,0,0
220 DATA 0,0,0,0,0,0,0,0,0
```

While we are learning about sprites and developing programs, I will include the remark statements and keep one statement to a line. You can crunch the programs later, if you want to do so.

Color and Sprites

Before we leave our musical sprite, let's use it to explore some of our options. We can:

- change it's color
- expand it.
- put more than one sprite on the screen.

Each sprite has a location on the VIC II chip that controls its color. The color is changed by using the same numbers, 0 – 16, that we used with the screen map in Chapter 9.

Here are the color locations on the VIC II chip:

Location		Sprite
V+39	53287	Sprite 0
V+40	53288	Sprite 1
V+41	53289	Sprite 2
V+42	53290	Sprite 3
V+43	53291	Sprite 4
V+44	53292	Sprite 5
V+45	53293	Sprite 6
V+46	53294	Sprite 7

At the moment, our program is using sprite 0, so to change its color, use location V+39 and change the color of the sprite before it is positioned on the screen. This line will color the sprite black:

```
75 POKE V+39,0
```

Questions & Answers

What would be the color of sprite 0 after this statement is executed?

```
POKE V+39,7
```

Answer: Yellow.

It is important not to confuse the sprite number with the color code. There are so many things going on in a sprite program that this can happen easily.

Questions & Answers

Try matching these statements with their effects:

(1) POKE V+39,1	(a) sprite 2 red
(2) POKE V+41,2	(b) sprite 7 light blue
(3) POKE V+44,8	(c) sprite 0 white
(4) POKE V+46,14	(d) sprite 1 purple
(5) POKE V+40,4	(e) sprite 5 orange

Answer: (a) 3, (b) 1, (c) 5, (d) 2, (e) 4

Run the quarter-note sprite program several times, changing the color in line 75 each time.

Expanding Sprites

The next option we will try is expanding the sprite. There are three ways to expand a sprite:

1. Double its width.
2. Double its height.
3. Double its width *and* its height.

To expand a sprite, we need two locations on the VIC II chip:

V+23	53271	expands sprite in y directions
V+29	53277	expands sprite in x directions

These two locations are used by all eight sprites, so they are used in the usual way by turning on the relevant bit.

Here is a quick quiz on the values to poke for each sprite at these locations.

Questions & Answers

Complete this table:

Sprite #	7	6	5	4	3	2	1	0
Turn on								
Turn off								

Answer:

Sprite #	7	6	5	4	3	2	1	0
Turn on	128	64	32	16	8	4	2	1
Turn off	127	191	223	239	247	251	253	254

If the sprite is going to be exanded, the expansion instruction must be included in the program before the sprite appears on the screen. Therefore, the expansion lines should be among the first lines of the program. In our program

that sends sprite 0 back and forth across the screen, the expansion instructions will come immediately after the sprite pointer line:

```
45 POKE V+29,1
```

This will expand the sprite in the x direction. Sprite expansion remains in effect if the program is run repeatedly without using RUN STOP and RESTORE between runs. The sprite expansion must be turned off in the program if you want the sprite to return to normal size.

Questions & Answers

Which of these values will turn off sprite 1?

_____ (a) 255
_____ (b) 254
_____ (c) 253

Answer: (c)

Try these substitutions in the program:

(a) POKE V+29,254: POKE V+23,1
(b) POKE V+29,1: POKE V+23,1
(c) POKE V+29,254: POKE V+23,254
(d) POKE V+29,1: POKE V+23,254

What is the effect of each statement?

Answer:

(a) Sprite expanded in y direction.
(b) Sprite expanded in both directions.
(c) Normal size sprite.
(d) Sprite expanded in x directions.

We can also use the quarter-note sprite program to put more than one sprite on the screen. We will put up three sprites, by making these changes to the program:

- Turn on three sprites, 0, 1 and 2.
- Create one pointer for each sprite.
- Choose a screen location for each sprite.

We will modify the program that put the sprite in the top left-hand corner of the screen. The first line change is line 20, to turn on or enable, three sprites. The enable location V + 21 was discussed previously. This location is used for all sprites, so the value of the sprites we select are added and the sum used in the POKE statement.

Questions & Answers

Line 20 turns on sprite 0. How would you change line 20 so that it will turn on sprites 0, 1 and 2?

Answer: 20 POKE V + 21,7

The next line that needs changing is line 30. This contains the sprite pointer statement. Each sprite has its own pointer location. The number that is poked indicates or "points" to the location in memory where the sprite data is stored. The number in the POKE statement is the result of this calculation:

pointer value = location/64

We will use the same data for each of the three sprites so that the number poked for each pointer will be the same. We will then have three sprites, each of which is identical to the other two. To set up three pointers we can put multiple statements into line 30:

30 POKE 2040,192: POKE 2041,192: POKE 2042,192

The last modification to be made to the program is to add x and y positions for the two new sprites. Each sprite has its own x and y locations on the VIC II chip. The values poked at these locations are taken from the screen map. We

have sprite 0 positioned in the top left-hand corner. We need to choose different locations for sprite 1 and sprite 2. To start, we will give sprite 1 and 2 the same x position as sprite 0, and vary the y position.

Questions & Answers

If we do this, will the sprites be lined up next to each other or under each other?

Answer: The y direction is the vertical direction, so they will be under each other.

Here is the program with the additions for sprites 1 and 2.

```
5 PRINTCHR$(147)
10 V= 53248: REM START VIC II
20 POKE V+21,7
30 POKE 2040,192: POKE 2041,192: POKE 2042,192
40 FOR C = 0 TO 62
50 READ D
60 POKE 12288+C,D
70 NEXT C
80 POKE V,24 : POKE V+2,24 : POKE V+4,24
90 POKE V+1,50 : POKE V+3,70 : POKE V+5,90
110 END
130 DATA 0,0,0,0,0,0
140 DATA 0,3,0,0,3,0
150 DATA 0,3,0,0,3,0
160 DATA 0,3,0,0,3,0
170 DATA 0,3,0,0,3,0
180 DATA 0,3,0,0,3,0
190 DATA 0,19,0,0,63,0
200 DATA 0,126,0,0,60,0
210 DATA 0,16,0,0,0,0
220 DATA 0,0,0,0,0,0,0,0,0,0
```

DATAFILE CHANGES

35 OPEN STATEMENT
50 INPUT# NOT READ
75 CLOSE

OMIT DATA STATEMENTS

Piano Program with Sprites

Earlier we created three different sprites:

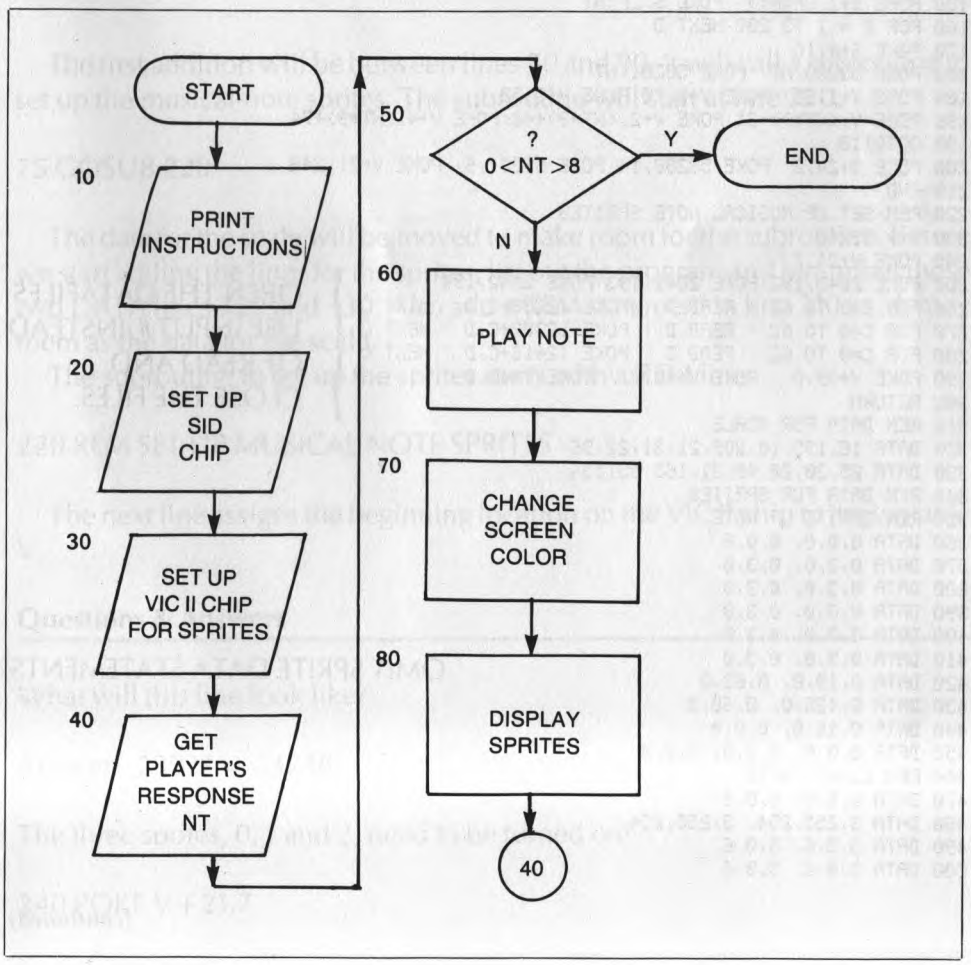
A quarter-note with the tail down.
A quarter-note with the tail up.
An eighth note.

In the next exercise we will include all three of these sprites in the "piano" program from Chapter 11.

Here is a list of the steps in the program that will include the sprites.

- Numbers 1-8 will play a note in the scale.
- Player chooses note.
- Note plays.
- Screen changes color.
- Three black sprites move across the screen, one after the other.
- Sequence is repeated for each note played.
- Anything other than the numbers 1-8 ends the program.

The flowchart looks like this:



Here is the listing for the program as it is now written, with placement of the new steps for the sprites indicated by the arrows.

```
10 S = 54272                                          DATAFILE CHANGES
20 FOR C= S TO S+24:POKE S,0:NEXTC
30 POKE S+24,15
40 POKE S+5,9:POKE S+6,0
50 FOR C= 1 TO 8
60 READ HF(C),LF(C)
70 NEXT C
75 GOSUB220
80 PRINT CHR$(147)
90 PRINT"NOW I WILL PLAY  YOUR TUNE FOR YOU"
100 PRINT"USE THE NUMBERS 1 - 8 ANYTHING ELSE WILL END THE PROGRAM"
110 GET N$:IF N$="" THEN110
120 NT = VAL(N$):IF NT<1 OR NT>8 THEN200
130 PRINT CHR$(147)
140 POKE S+4,17
150 POKE S+1,HF(NT): POKE S,LF(NT)
160 FOR D = 1 TO 200:NEXT D
170 POKE S+4,16
180 POKE 53280,NT: POKE 53281,NT
184 POKE V+1,60: POKE V+3,70:POKE V+5,50
186 POKE V,(NT*9)+31:POKE V+2,(NT*9)+48:POKE V+4,(NT*9)+74
190 GOTO110
200 POKE S+24,0 :POKE 53280,14:POKE 53281,6: POKE V+21,248
210 END
220 REM SET UP MUSICAL NOTE SPRITES
230 V = 53248
240 POKE V+21,7
250 POKE 2040,192:POKE 2041,193:POKE 2042,194
260 FOR C=0 TO 62 : READ D : POKE 12288+C,D : NEXT C   } OPEN THE DATAFILES
270 FOR C=0 TO 62 : READ D : POKE 12352+C,D : NEXT C   } USE INPUT # INSTEAD
280 FOR C=0 TO 62 : READ D : POKE 12416+C,D : NEXT C   } OF READ AND
290 POKE V+39,0 : POKE V+40,0 : POKE V+41,0            } CLOSE THE FILES.
300 RETURN
310 REM DATA FOR SCALE
320 DATA 16,195,18,209,21,31,22,96
330 DATA 25,30,28,49,31,165,33,134
340 REM DATA FOR SPRITES
350 REM SPRITE UP NOTE
360 DATA 0,0,0, 0,0,0
370 DATA 0,3,0, 0,3,0
380 DATA 0,3,0, 0,3,0
390 DATA 0,3,0, 0,3,0
400 DATA 0,3,0, 0,3,0
410 DATA 0,3,0, 0,3,0                                 OMIT SPRITE DATA STATEMENTS
420 DATA 0,19,0, 0,63,0
430 DATA 0,126,0, 0,60,0
440 DATA 0,16,0, 0,0,0
450 DATA 0,0,0, 0,0,0, 0,0,0
460 REM EIGHTH NOTE
470 DATA 0,0,0, 0,0,0
480 DATA 3,255,254, 3,255,254
490 DATA 3,0,6, 3,0,6
500 DATA 3,0,6, 3,0,6
```

(continued)

```
510 DATA 3,0,6, 3,0,6
520 DATA 3,0,6, 3,0,6
530 DATA 19,0,38, 63,0,126
540 DATA 126,0,252, 60,0,120
550 DATA 16,0,32, 0,0,0
560 DATA 0,0,0, 0,0,0, 0,0,0
570 REM DOWN NOTE
580 DATA 0,0,0, 0,0,0
590 DATA 0,0,0, 0,0,0
600 DATA 0,32,0, 0,240,0
610 DATA 1,248,0, 3,240,0
620 DATA 3,32,0, 3,0,0
630 DATA 3,0,0, 3,0,0
640 DATA 3,0,0, 3,0,0
650 DATA 3,0,0, 3,0,0
660 DATA 3,0,0, 3,0,0
670 DATA 3,0,0, 3,0,0
680 DATA 3,0,0, 0,0,0, 0,0,0
```

The first addition will be between lines 80 and 90. It will call a subroutine to set up the musical-note sprites. The subroutine will start at line 220.

```
75 GOSUB 220
```

The data for the scale will be moved to make room for the subroutine. Before we start adding the lines for the sprites, list out the program and renumber these two DATA lines 320 and 330. Also add a REM statement in line 310 to identify them as the data for the scale.

The subroutine to set up the sprites starts with a REM line:

```
220 REM SET UP MUSICAL NOTE SPRITES
```

The next line assigns the beginning location on the VIC II chip to the variable V.

Questions & Answers

What will this line look like?

Answer: 230 V = 53248

The three sprites, 0, 1 and 2, need to be turned on:

```
240 POKE V+21,7
```

All three sprites will need a pointer, and this time each sprite will be different. Can you write a line with multiple statements for the three sprite pointers?

Answer: 250 POKE 2040,192 : POKE 2041,193 : POKE 2042, 194

The next step in setting up the sprites is to read in the data for each sprite. To do this, use a FOR-NEXT loop and POKE the data into memory. What will be the starting location for sprite 0 data?

Answer: 12288

The three loops to read in the sprite data will look like this:

```
260 FOR C = 0 TO 62: READ D: POKE 12288 +C,D :NEXT C
270 FOR C = 0 TO 62: READ D : POKE 12352 +C,D :NEXT C
280 FOR C = 0 TO 62 : READ D : POKE 12416 +C,D :NEXT C
```

We will use three loops, so that one location is left at the end of each sprite data set for the marker supplied by the computer.

The last line in the subroutine sets the color of the sprites. The program already changes the screen color with every note, so we will keep the notes black.

Questions & Answers

Which of these multiple statements will set the color of all three sprites to black?

(a) 290 POKE V+39,1 : POKE V+40,1 : POKE V+41,1
(b) 290 POKE V+39,0 : POKE V+40,0 : POKE V+41,0

Answer: (b) The color code for black is 0. The color code for white is 1.

What is the last statement for any subroutine?

Answer: RETURN

So, the last line for the subroutine will be:

Answer: 300 RETURN

The DATA statements for the three sprites follow immediately after the data for the scale, starting at line 340. We will use the three sprites we created at the beginning of the chapter. Type the numbers from the grids into DATA statements starting at line 300. If you put the data from two rows on the grid in one DATA statement, it will be easy to proofread.

The last change is an addition to the main program to position the sprites on the screen. This will happen after the screen changes color in line 180. The lines look like this:

```
184 POKE V+1,60 : POKE V+3,70 : POKE V+5,90
186 POKE V,31+NT*9 : POKE V+2,48+NT*9 : POKE V+4,74+NT*9
```

Questions & Answers

Which of these lines sets the X position?

Answer: 186

The X position will vary with the note played, because the variable NT, which selects the note, is included in the POKE statement for the X position.

Questions & Answers

Is the Y position constant for each sprite?

Answer: Yes, line 184 does not include a variable.

Animation and Collisions

There are two other sprite features that have not been mentioned yet. The first is animation. We have moved sprites around the screen, but we have not made an individual sprite give the appearance of movement, for example, a person walking or a bird flying.

The second feature is the ability to detect collisions. The VIC II chip has two locations that are used to register collisions, either sprite-sprite collisions or sprite-background collisions. It is this feature that allows us to program arcade-type video games without too much difficulty.

Now we are going to try our skill at programming a simple video game. Once again we will build the program step by step. This time we will include all the sprite features we have used up to this point, with some of the graphics features covered in Chapter 9, and the new collision feature.

This is the plot for a game I call "The Revenge of the Killer Duck." It is duck-hunting season, and the player is the hunter in a duck-blind at the bottom of the screen. A cyan duck flies across the screen. The hunter can fire arrows at the duck in hopes of obtaining a duck dinner. However, the duck can defend itself by dropping exploding eggs. If an egg hits the hunter, his hunting days are over.

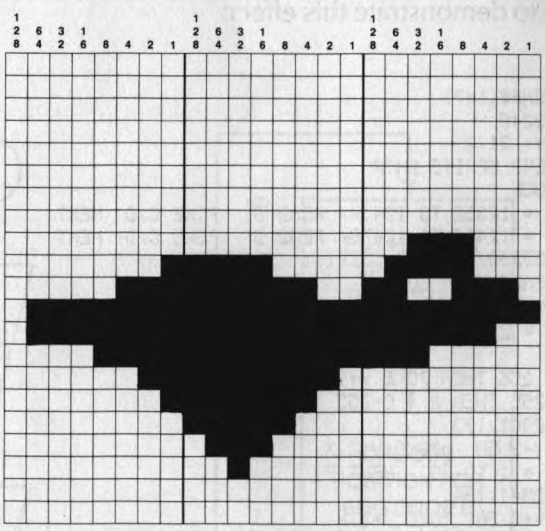
- There are three sprites:
  a cyan duck,
  a yellow egg,
  a brown hunter.
- The arrows are black graphic characters, screen code 30.
- The duck moves across the top of the screen.
- The hunter is fixed at the bottom of the screen.
- The egg moves down the screen, starting from a randomly chosen point at the top of the screen.
- The hunter may fire his arrows at anytime. They move straight up the screen.

Collisions between two sprites (the egg and the hunter) and a sprite and the background (the arrow and the duck) are tested after the release of an egg or an arrow. These are the main points for the program, although we will also need an introduction to explain the game and suitable ending messages.

Before we start programming the game, let's take a look at how the duck will fly across the screen.

We will use one sprite and two sets of sprite data, like this:

Row 1	0	0	0
Row 2	0	0	0
Row 3	0	0	0
Row 4	0	0	0
Row 5	0	0	0
Row 6	0	0	0
Row 7	0	0	0
Row 8	0	0	0
Row 9	0	0	56
Row 10	1	240	20
Row 11	7	252	206
Row 12	127	255	255
Row 13	127	255	255
Row 14	15	255	224
Row 15	3	254	0
Row 16	1	252	0
Row 17	0	248	0
Row 18	0	114	0
Row 19	0	32	0
Row 20	0	0	0
Row 21	0	0	0



Row 1	0	0	0
Row 2	0	0	0
Row 3	0	0	0
Row 4	0	0	0
Row 5	0	0	0
Row 6	0	64	0
Row 7	0	224	0
Row 8	1	240	0
Row 9	3	248	56
Row 10	3	248	120
Row 11	7	252	206
Row 12	127	255	255
Row 13	127	255	252
Row 14	15	255	224
Row 15	0	120	0
Row 16	0	0	0
Row 17	0	0	0
Row 18	0	0	0
Row 19	0	0	0
Row 20	0	0	0
Row 21	0	0	0

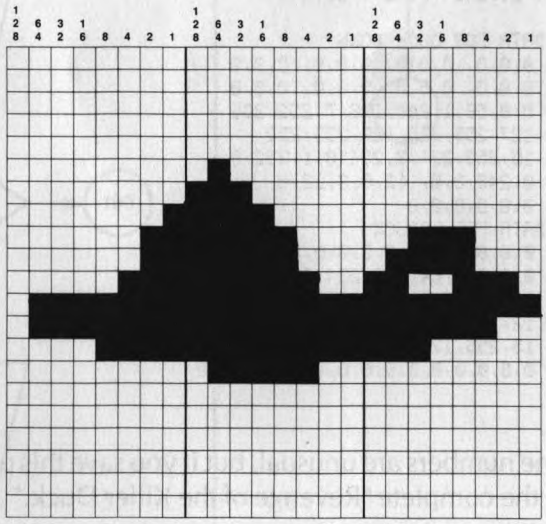


Figure 11.6 The two sprite charts to create the flying duck.

The animated effect occurs when the two sets of data are used alternately to create a sprite in the same position on the screen. First, we will use the duck with its wings down, then with its wings up. After showing the two sprites in the

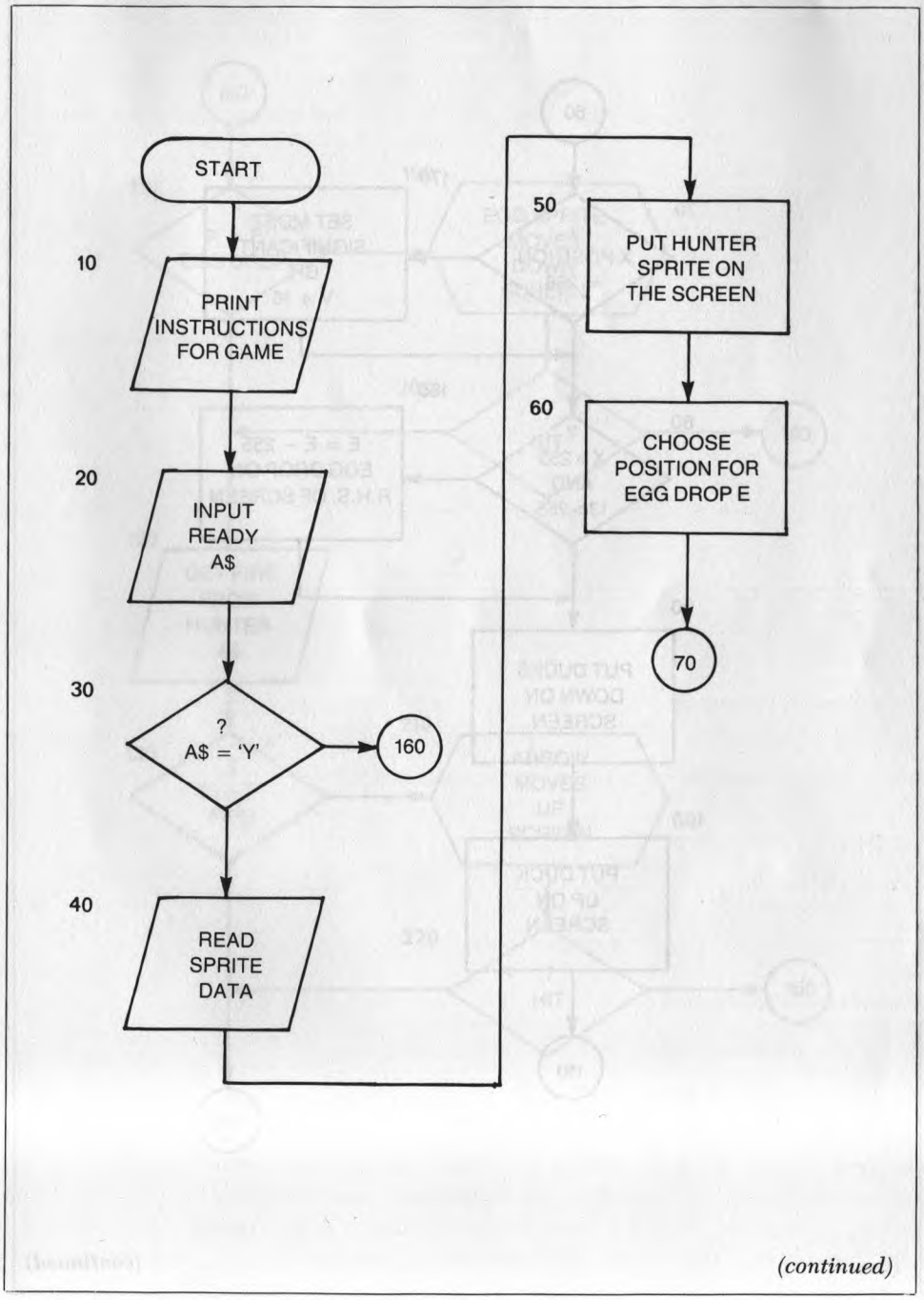
same position, we move one pixel to the right and show them both again. This sequence is repeated all the way across the screen. Here is the listing for a short program to demonstrate this effect:

```
110 PRINTCHR$(147)
115 V = 53248
117 POKE V+ 21,2
120 REM READ SPRITE DATA
170 REM DUCK
190 FOR C = 12352 TO 12414: READ D : POKE C,D: NEXT
220 FOR C = 12416 TO 12478: READ D : POKE C,D: NEXT
310 POKE 53280,1: POKE 53281,1
400 REM START MAIN LOOP
410 POKE V+40,3
420 FOR C = 0 TO 347
430 X= C
435 IF X= 256 THEN POKE V+16,2
440 IF X>255 THEN X = C-255
460 POKE 2041,193
470 POKE V+3,50 : POKE V+2,X
480 FOR P = 1 TO 50 : NEXT P
490 POKE 2041,194
500 POKE V+3,50: POKE V+2,X
540 NEXT C
550 POKE V+21,248 :POKE V+16,249
580 END
1100 REM DATA FOR DUCK DOWN
1110 DATA 0,0,0, 0,0,0, 0,0,0, 0,0,0
1120 DATA 0,0,0, 0,0,0, 0,0,0, 0,0,0
1130 DATA 0,0,56,1,240,120,7,252,206
1140 DATA 127,255,255,127,255,252
1150 DATA 15,255,224,3,254,0,1,252,0
1160 DATA 0,248,0,0,112,0,0,32,0
1170 DATA 0,0,0,0,0,0
1200 REM DATA FOR UPDUCK
1210 DATA 0,0,0,0,0,0,0,0,0,0,3,0
1220 DATA 0,0,0,0,64,0,0,224,0,1,240,0
1230 DATA 3,248,56,3,248,120,7,252,206
1240 DATA 127,255,255,127,255,252
1250 DATA 15,255,224,0,120,0,0,0,0
1260 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

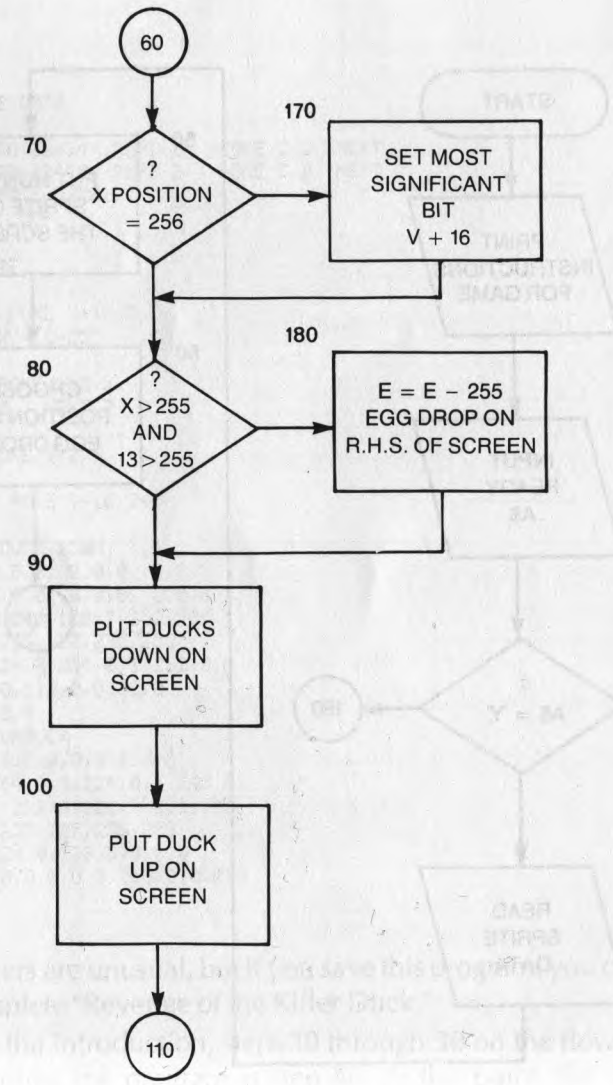
The line numbers are unusual, but if you save this program, you can add to it to create the complete "Revenge of the Killer Duck."

We will leave the introduction, steps 10 through 30 on the flowchart, until later and start writing the program at step 40. At this point, the data for the sprites is read in and poked into memory. There are three sprites, but the duck sprite will require two sets of data, one with the wing up and one with the wing down. These two sets will be used alternately by the same sprite to give the appearance of flying. So, there will be a total of four sets of data.

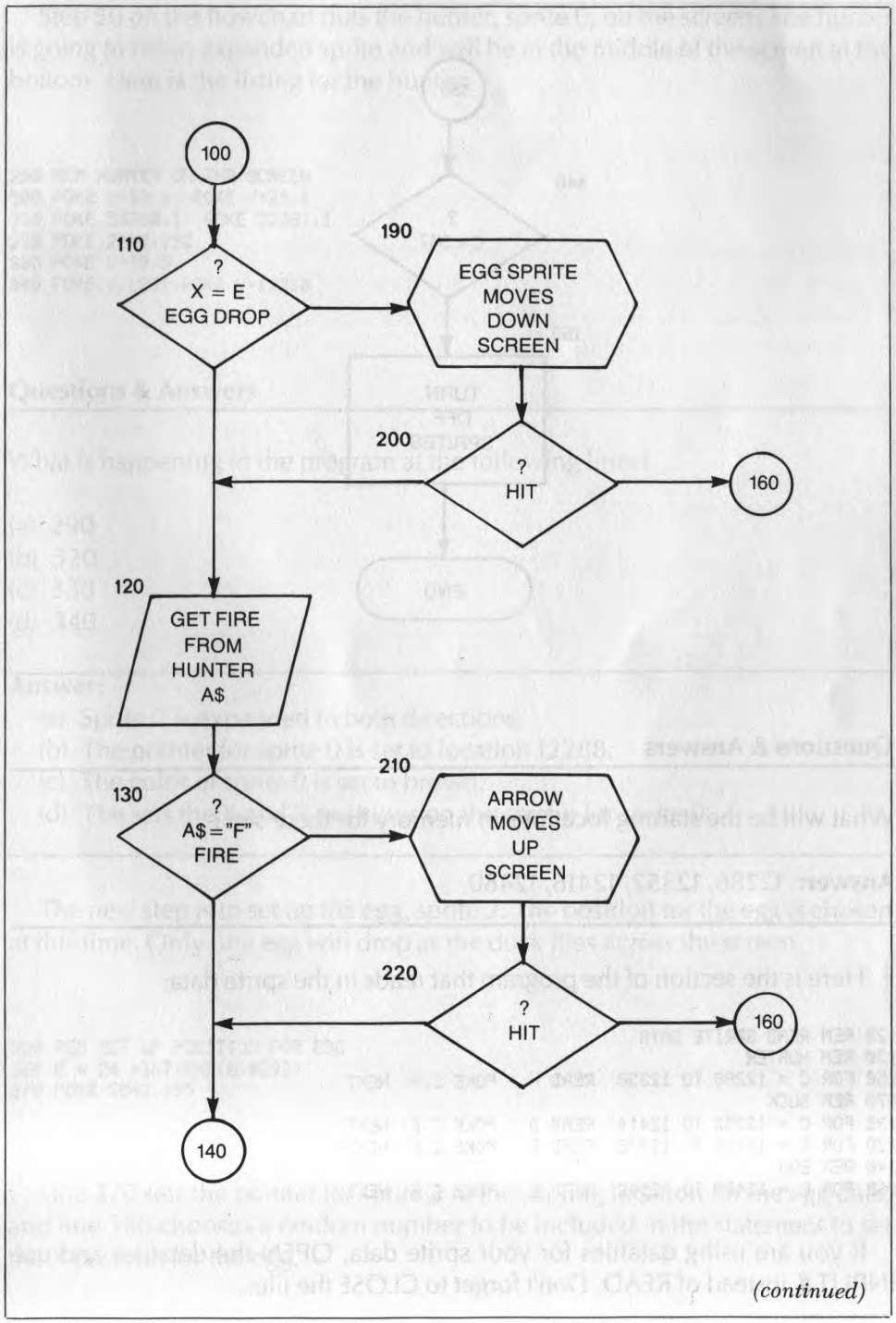
Here is the flowchart for the entire game:



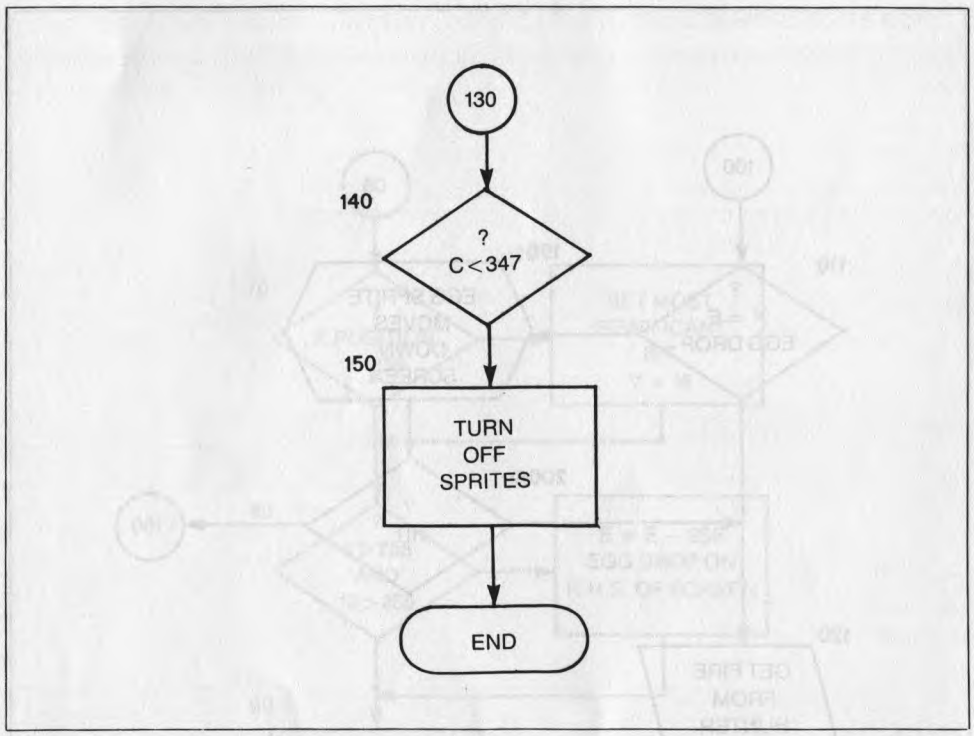
(continued)


60
70
?
X POSITION
= 256
170
SET MOST
SIGNIFICANT
BIT
V + 16
80
?
X > 255
AND
13 > 255
180
E = E − 255
EGG DROP ON
R.H.S. OF SCREEN
90
PUT DUCKS
DOWN ON
SCREEN
100
PUT DUCK
UP ON
SCREEN
110

(continued)


100
110
?
X = E
EGG DROP
190
EGG SPRITE
MOVES
DOWN
SCREEN
200
?
HIT
160
120
GET FIRE
FROM
HUNTER
A$
130
?
A$ = "F"
FIRE
210
ARROW
MOVES
UP
SCREEN
220
?
HIT
160
140

(continued)



Questions & Answers

What will be the starting locations in memory for these sets?

Answer: 12288, 12352, 12416, 12480

Here is the section of the program that reads in the sprite data:

```
120 REM READ SPRITE DATA
130 REM HUNTER
150 FOR C = 12288 TO 12350: READ H : POKE C,H: NEXT
170 REM DUCK
190 FOR C = 12352 TO 12414: READ D : POKE C,D: NEXT
220 FOR C = 12416 TO 12478: READ D : POKE C,D: NEXT
240 REM EGG
260 FOR C = 12480 TO 12542: READ E : POKE C,E: NEXT
```

If you are using datafiles for your sprite data, OPEN the datafiles and use INPUT # instead of READ. Don't forget to CLOSE the files.

Step 50 on the flowchart puts the hunter, sprite 0, on the screen. The hunter is going to be an expanded sprite and will be in the middle of the screen at the bottom. Here is the listing for the hunter:

```
290 REM HUNTER ON THE SCREEN
300 POKE V+23,1: POKE V+29,1
310 POKE 53280,1: POKE 53281,1
320 POKE 2040,192
330 POKE V+39,9
340 POKE V,185: POKE V+1,210
```

Questions & Answers

What is happening in the program at the following lines?

(a) 290
(b) 320
(c) 330
(d) 340

Answer:

(a) Sprite 0 is expanded in both directions.
(b) The pointer for sprite 0 is set to location 12288.
(c) The color of sprite 0 is set to brown.
(d) The sets the X and Y positions on the screen for sprite 0.

The next step is to set up the egg, sprite 2. The position for the egg is chosen at this time. Only one egg will drop as the duck flies across the screen.

```
350 REM SET UP POSITION FOR EGG
360 E = 24 +INT(RND(0)*292)
370 POKE 2042,195
```

Line 370 sets the pointer for sprite 2 to the starting location for the egg data, and line 360 chooses a random number to be included in the statement to set the X position for the egg.

Questions & Answers

What are the minimum and maximum values for the X position for the egg?

Answer: Minimum = 24 Maximum = 316

The rest of the action for the game is contained in a loop, flowchart steps 70 through 140, as the duck flies across the screen. The loop is set to move the duck from position X + 0 to position X + 347. The X position will exceed 255, so the most significant bit for the duck sprite 1 will have to be turned on at the right time in the loop. Here is the listing for the loop:

```
400 REM START MAIN LOOP
410 POKE V+40,3: POKE V+41,7
420 FOR C = 0 TO 347
430 X= C
435 IF X= 256 THEN POKE V+16,6
440 IF X>255 THEN X = C-255
450 IF E>255 AND C>255 THEN E = E-255
460 POKE 2041,193
470 POKE V+3,50 : POKE V+2,X
480 FOR P = 1 TO 20 : NEXT P
490 POKE 2041,194
500 POKE V+3,50: POKE V+2,X
510 IF X=E THEN GOSUB 700
520 GET A$
530 IF A$="F" THEN GOSUB 900
540 NEXT C
```

Now here is a line-by-line explanation:

420 Start loop.

430 X will be used to store the position in the X direction on the screen.

440 Reset X for the right-hand side of the screen.

450 Reset position for the egg on the right-hand side of the screen, if necessary.

460 Set pointer for sprite 1 to duck with wing down.

470 Set X and Y position for duck sprite 1.

480 Pause.

490 Set pointer for duck sprite to duck with wing up.

500 Set same X and Y positions for duck sprite.

510 Check for dropping egg and branch to subroutine, if necessary.

520 Get action from player, if any.

530 If player has acted, check for fire response and branch to subroutine to shoot arrow.

540 End loop.

After the loop ends, sprites 0, 1, and 2 will need to be turned off and the most significant bit for sprites 1 and 2 must also be turned off.

Questions & Answers

Can you write a multiple statement line to take care of these two actions?

Answer: 550 POKE V+21,248 : POKE V+16,249

This is the end of the main program. There are two subroutines, one for the falling egg and one for shooting arrows. First, we will program the falling egg.

The egg moves down the screen, and if it hits the hunter, that is the end of the hunter. Otherwise, the game continues. If the egg *does* hit the hunter, that is a sprite-sprite collision, detected by location V+30 on the VIC II chip. This location is used by all the sprites, so there will have to be a bit check at this location. Here is the listing for the subroutine:

```
700 REM FALLING EGG SUBROUTINE
705 POKE V+30,0
710 FOR F = 92 TO 229
720 POKE V+4,E:POKE V+5,F
730 NEXT F
740 IF PEEK(V+30) =0 THEN 800
750 POKE V+21,248: POKE V+16,249
760 PRINT CHR$(147):PRINT:PRINT:PRINT
770 PRINT TAB(5) "  S  P  L  A  T  !  !  !"
780 PRINT:PRINT:PRINT TAB(5)"  FAREWELL BRAVE HUNTER "
790 END
800 RETURN
```

At the beginning of the subroutine the collision location V+30 is set to 0. Then the egg moves down the screen. When it stops, location V+30 is checked for a collision at line 740. If no collision has occurred, the program returns to the main program. If not, lines 790-800 are executed, a suitable ending message is printed, and the sprites are turned off.

In this program there is only one possible sprite-sprite collision, the egg and the hunter. So, the test to see if the collision location was not zero worked well. If there is the possibility of more than one sprite-sprite collision, it is possible to see if a particular sprite has collided by checking that sprite's bit at location V+30. For example, this statement will check to see if sprite 2 has collided with another sprite:

IF PEEK (V+30)AND 4 = 4 THEN . . . .

This is an example of a logical AND operation similar to those discussed in Chapter 7. This statement will only be true if the bit representing sprite 2 is on.

Questions & Answers

Which of the following statements will check to see if sprite 5 has collided with another sprite?

(a) IF PEEK(V+30)AND 64 = 64 THEN . . .
(b) IF PEEK (V+30) AND 5 = 5 THEN . . .
(c) IF PEEK(V+30) AND 32 = 32 THEN . . .

Answer: (c)

The second subroutine is called when the player fires the hunter's arrow by pressing the 'F' key. Here is the listing for this subroutine:

```
900 REM SHOOTING ARROW SUBROUTINE
905 POKE V+31,0
910 FOR A = 0 TO 20
920 POKE 1848-(40*A),30 : POKE 56120-(40*A),0
930 NEXT A
940 IF PEEK(V+31)AND 2=2 THEN950
942 PRINT CHR$(147)
945 RETURN
```

(continued)

```
950 POKE V+21,248: POKE V+16,249
960 PRINT CHR$(147):PRINT:PRINT:PRINT
970 PRINT TAB(5)" DUCK FOR DINNER TONIGHT"
980 END
```

The arrow is a black graphics character. It moves up the screen, and its position is determined by using the screen and color memory maps given in Chapter 9.

After the arrow has travelled up the screen, the sprite-background collision location is checked to see if the arrow hit the duck. Because the arrow is a graphics character, it is considered to be "background." Background is anything that is not a sprite. Line 940 checks to see if sprite 1 has collided with the background. Of course, as there is only one possible collision, we also could have checked to see if the location was still zero.

After the subroutines are written, the DATA statements are added. Figures 10 and 11 on page 410, show my sprite designs with the coding for the DATA statements for the hunter and the egg.

Questions & Answers

Try adding the data statements and a suitable introduction to the program before you check my listing.

Answer: Here is the complete listing for my program. Two methods of handling sprite data — DATA statements and datafiles — are shown. The program runs faster with DATA statements.

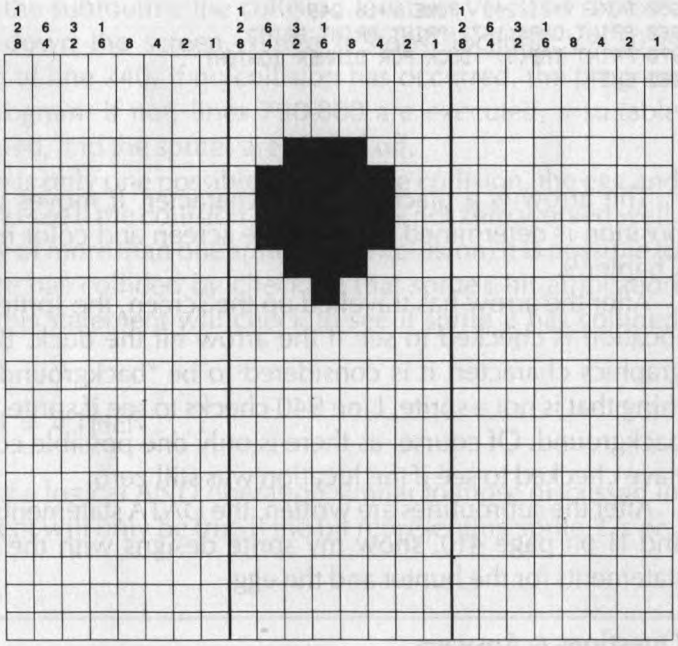
```
10 PRINT CHR$(147)
20 POKE 53280,5:POKE 53281,1
30 FOR C = 1 TO 8:PRINT:NEXTC
40 PRINT TAB(8)"REVENGE OF THE KILLER DUCK"
50 PRINT:PRINT:PRINT
60 PRINT"YOU ARE THE BRAVE DUCK HUNTER"
70 PRINT"USE THE F KEY TO SHOOT YOUR ARROWS"
80 PRINT "BEWARE OF FALLING EGGS!"
90 PRINT:PRINT:PRINT  "ARE YOU READY (Y/N)";A$
95 GET A$ : IF A$="" THEN 95
100 IF A$<>"Y" THEN END
110 PRINTCHR$(147)
115 V = 53248
117 POKE V+ 21,7
120 REM READ SPRITE DATA
130 REM HUNTER
150 FOR C = 12288 TO 12350: READ H : POKE C,H: NEXT
```

DATA FILE CHANGES

```
140 OPEN STATEMENT
150 INPUT# NOT READ
160 CLOSE NOT READ
180 OPEN STATEMENT
```

(continued)

Row 1	0	0	0
Row 2	0	0	0
Row 3	0	0	0
Row 4	0	56	0
Row 5	0	124	0
Row 6	0	124	0
Row 7	0	124	0
Row 8	0	56	0
Row 9	0	16	0
Row 10	0	0	0
Row 11	0	0	0
Row 12	0	0	0
Row 13	0	0	0
Row 14	0	0	0
Row 15	0	0	0
Row 16	0	0	0
Row 17	0	0	0
Row 18	0	0	0
Row 19	0	0	0
Row 20	0	0	0
Row 21	0	0	0



Row 1	0	0	0
Row 2	0	0	0
Row 3	0	0	0
Row 4	0	1	128
Row 5	4	121	128
Row 6	14	134	0
Row 7	14	70	0
Row 8	4	71	0
Row 9	31	225	128
Row 10	47	8	128
Row 11	47	4	128
Row 12	46	3	0
Row 13	31	0	0
Row 14	27	0	0
Row 15	27	0	0
Row 16	27	0	0
Row 17	27	128	0
Row 18	27	128	0
Row 19	0	0	0
Row 20	0	0	0
Row 21	0	0	0

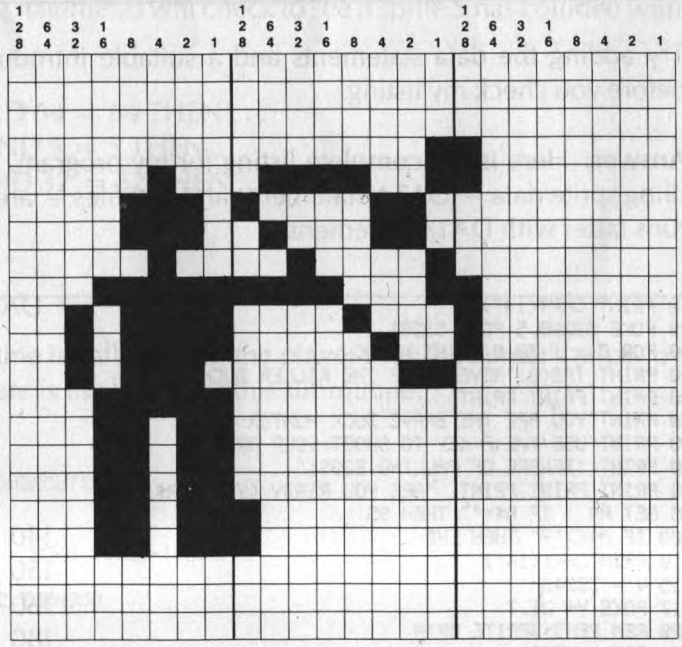


Figure 11.7a & **7b** Egg and hunter sprites.

```
170 REM DUCK
190 FOR C = 12352 TO 12414: READ D : POKE C,D: NEXT
220 FOR C = 12416 TO 12478: READ D : POKE C,D: NEXT
240 REM EGG
260 FOR C = 12480 TO 12542: READ E : POKE C,E: NEXT
290 REM HUNTER ON THE SCREEN
300 POKE V+23,1: POKE V+29,1
310 POKE 53280,1: POKE 53281,1
320 POKE 2040,192
330 POKE V+39,9
340 POKE V,185: POKE V+1,210
350 REM SET UP POSITION FOR EGG
360 E = 24 +INT(RND(0)*292)
370 POKE 2042,195
400 REM START MAIN LOOP
410 POKE V+40,3: POKE V+41,7
420 FOR C = 0 TO 347
430 X= C
435 IF X= 256 THEN POKE V+16,6
440 IF X>255 THEN X = C-255
450 IF E>255 AND C>255 THEN E = E-255
460 POKE 2041,193
470 POKE V+3,50 : POKE V+2,X
480 FOR P = 1 TO 20 : NEXT P
490 POKE 2041,194
500 POKE V+3,50: POKE V+2,X
510 IF X=E THEN GOSUB 700
520 GET A$
530 IF A$="F" THEN GOSUB 900
540 NEXT C
550 POKE V+21,248 :POKE V+16,249
560 PRINT CHR$(147):PRINT:PRINT:PRINT
570 PRINTTAB(5)" BETTER LUCK NEXT TIME"
580 END
700 REM FALLING EGG SUBROUTINE
705 POKE V+30,0
710 FOR F = 92 TO 229
720 POKE V+4,E:POKE V+5,F
730 NEXT F
740 IF PEEK(V+30) =0 THEN 800
750 POKE V+21,248: POKE V+16,249
760 PRINT CHR$(147):PRINT:PRINT:PRINT
770 PRINT TAB(5) "  S  P  L  A  T  !  !  !"
780 PRINT:PRINT:PRINT TAB(5)"  FAREWELL BRAVE HUNTER "
790 END
800 RETURN
900 REM SHOOTING ARROW SUBROUTINE
905 POKE V+31,0
910 FOR A = 0 TO 20
920 POKE 1848-(40*A),30 : POKE 56120-(40*A),0
930 NEXT A
940 IF PEEK(V+31)AND 2=2 THEN950
942 PRINT CHR$(147)
945 RETURN
950 POKE V+21,248: POKE V+16,249
960 PRINT CHR$(147):PRINT:PRINT:PRINT
970 PRINT TAB(5)" DUCK FOR DINNER TONIGHT"
```

DATAFILE CHANGES

190 INPUT # NOT READ
220 INPUT # NOT READ
230 CLOSE
250 OPEN STATEMENT
260 INPUT # NOT READ

(continued)

DATAFILE

```
980 END
1000 REM DATA FOR HUNTER
1010 DATA 0,0,0, 0,0,0, 0,0,0, 0,0,128
1020 DATA 4,121,128,14,134,0,14,70,0
1030 DATA 4,41,0,31,255,128,47,8,128
1040 DATA 47,4,128,46,3,0,31,0,0
1050 DATA 27,0,0,27,0,0,27,0,0,27,128,0
1060 DATA 27,128,0,0,0,0,0,0,0,0,0,0
1100 REM DATA FOR DUCK DOWN
1110 DATA 0,0,0, 0,0,0, 0,0,0, 0,0,0
1120 DATA 0,0,0, 0,0,0, 0,0,0, 0,0,0
1130 DATA 0,0,56,1,240,120,7,252,206
1140 DATA 127,255,255,127,255,252
1150 DATA 15,255,224,3,254,0,1,252,0
1160 DATA 0,248,0,0,112,0,0,32,0
1170 DATA 0,0,0,0,0,0
1200 REM DATA FOR UPDUCK
1210 DATA 0,0,0,0,0,0,0,0,0,0,0,0
1220 DATA 0,0,0,0,64,0,0,224,0,1,240,0
1230 DATA 3,248,56,3,248,120,7,252,206
1240 DATA 127,255,255,127,255,252
1250 DATA 15,255,224,0,120,0,0,0,0
1260 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1300 REM DATA FOR EGG
1310 DATA 0,0,0,0,0,0,0,0,0,0
1320 DATA 0,56,0,0,124,0,0,124,0
1330 DATA 0,124,0,0,56,0,0,16,0
1340 DATA 0,0,0,0,0,0,0,0,0,0,0,0
1350 DATA 0,0,0,0,0,0,0,0,0,0,0,0
1360 DATA 0,0,0,0,0,0,0,0,0,0,0,0
```

OMIT IF USING DATAFILES

I think that you will agree that "Revenge of the Killer Duck" is not going to make us a fortune at the local video arcade, but, it is a beginning and now it is up to you to add your own enhancement, such as:

- Make the duck fly faster.
- Have more than 1 egg falling.
- Let the hunter move around.
- Have the duck fly across the screen more than once.
- Add sound effects.

Some of these ideas will be easier to program than others, and I am sure that you can invent your own additions to the program.

If you check the VIC II map in Appendix J, you will see that we have used most of the sprite locations in this chapter. We have not discussed multicolored sprites, and I suggest that you save this option until you have become familiar with the material included in this chapter. Writing programs that include

sprites can become very complicated, but we have made a start and I hope that you will enjoy including them in future programs.

Also, there are several excellent "sprite-maker" programs available commercially for the Commodore 64. These greatly simplify the creation of sprites, and you might want to explore one if you plan to do a lot of sprite programming.

Before you close this book, please take the final Self-Check.

SELF-CHECK

1. (a) How many pixels are there on a sprite grid?
   (b) How many bytes are used to code a sprite?
   (c) What is the 64th byte at the end of the sprite data in memory used for?
2. (a) What is the sprite pointer location for sprite 4?
   (b) How would the sprite pointer for sprite 4 be set to read data starting at location 12672?
3. (a) When do you use location V+16 on the VIC II chip?
   (b) How would you turn on the most significant bit in the X direction for sprite 6?
4. Which of these statements sets the color for sprite 3?

   (a) 40 POKE V+41,0
   (b) 40 POKE V+42,0
   (c) 40 POKE V+43,0
5. (a) In the following statement, what color will the sprite be?
   (b) Which sprite is being used?

   10 POKE V+41,11
6. Where on the screen will the sprite appear if the following statements are executed?

   40 POKE V+2,24
   50 POKE V+3,229
7. What are the three possibilities for expanding a sprite?
8. (a) Write a statement to turn on sprites 1,2, and 5.
   (b) Write a statement to turn off sprites 1,2 and 5.

9. Which locations on the VIC II chip are used for the following functions?
   (a) Sprite-sprite collisions.
   (b) Sprite-background collisions.
   (c) Expand a sprite in the X direction.
   (d) Expand a sprite in the Y direction.

Answers to SELF-CHECK

1. (a) 504 (24 columns * 21 rows)
   (b) 63
   (c) A marker to mark the end of the sprite data.
2. (a) 2044
   (b) POKE 2044,198
3. (a) The location V+16 is used to position a sprite on the right-hand side of the screen.
   (b) POKE V+16,64
4. (b)
5. (a) Dark gray
   (b) Sprite 2
6. Bottom left-hand corner of the screen.
7. Double width, double height, double width and double height.
8. (a) POKE V+21,38
   (b) POKE V+21,217
9. (a) V+30
   (b) V+31
   (c) V+29
   (d) V+23

Ideas for Further Exploration with Sprites

Sprites are the most versatile and creative graphics feature of the Commodore 64. I hope that you will improve on the "Revenge of the Killer Duck" program and go on to program your own video games.

Sprites can also be used for programs other than game programs. They can be used to draw the user's attention to items on a menu, or they can be included in educational programs, either as an aid to learning or as a reward for good work.

In this book we have only just begun to use sprites, but I hope that you have learned enough to want to continue using them in your programs.

I hope that you will continue to write programs for your Commodore 64 and to explore further the topics we have covered in this book.

APPENDIX A

Error Messages

These are some of the more frequent error messages that you will encounter on the Commodore 64. Not all errors generate an error message, so be sure to always check the red LED light on the disk drive and test the program with some trial data that has a known output.

BAD DATA — The program expected string data and found numbers, or it expected numeric data and found strings. Check that the variable name in the READ statement matches the data in the DATA statement. Also, if there is more than one variable being read, check the order of the data in the DATA statements.

BAD SUBSCRIPT—This refers to an array variable and means that the subscript number is too large. Check the DIM statement and the size of the array.

CAN'T CONTINUE — The CONT command will not work. This happens if the program cannot continue because it was not RUN, or there have been some changes made with the screen editor, or it stopped because of an error. Use RUN instead.

DEVICE NOT PRESENT — The input/output device, (datassette, disk drive or printer) asked for in an OPEN, CLOSE, CMD, PRINT #, INPUT # or GET # statement was not available. Check that the device is connected and turned on.

DIVISION BY ZERO — Division by zero is a math error. Check for calculations that could possibly result in something like this — 5/0. Most often there is a

variable that can have the value 0; it should be tested for and rejected before the division occurs in the program.

EXTRA IGNORED — Too many items were entered in response to an INPUT statement, or a comma was included as part of the information entered. Remember that a comma separates items in an INPUT statement.

FILE NOT FOUND — The file asked for in a LOAD command could not be found. On a tape, this happens when an END-OF-TAPE marker is found. On a disk, the file was not there. Check the spelling of the file name and that you have the correct tape or disk loaded in the device.

FILE NOT OPEN — This happens with CLOSE, CMD, PRINT #, INPUT #, and GET # statements when the file mentioned is not opened. Check the OPEN statements in the program or enter an OPEN command.

FORMULA TOO COMPLEX — A string expression is too long or there are more than ten sets of parentheses. If this happens, the formula needs to be broken down into smaller steps.

ILLEGAL DIRECT — The INPUT statement was used in the direct mode. It can only be used in a program.

ILLEGAL QUANTITY — A number used as the argument of a function or statement is out of the allowable range.

LOAD — There is a problem with the program on the tape or disk, and it could not be loaded.

NEXT WITHOUT FOR — This is caused by either incorrectly nesting loops or having a variable name in a NEXT statement that doesn't match the one in a FOR statement.

NOT INPUT FILE — An attempt was made to INPUT or GET data from a file which was specified to be for output only.

NOT OUTPUT FILE — An attempt was made to PRINT data to a file which was specified as input only.

OUT OF DATA — A READ statement was executed, but there is no data left. Check the number of items in the DATA statements, the number of variables in the READ statements, and the number of times the READ statement is executed.

OUT OF MEMORY — There is no more RAM available for the program or variables. This may also occur when too many FOR loops have been nested, or when there are too many GOSUBs in effect.

OVERFLOW — The result of a combination is larger than the largest number allowed, which is 1.70141884E+38.

REDIM'D ARRAY — The array may only be DIMensioned once. This error will occur if the array is dimensioned more than once, or if it is dimensioned after it has been used. In the second case, the maximum undimensioned size, 0-10, will automatically have been assigned, thus causing the error. Make a habit of always dimensioning arrays at the beginning of the program.

REDO FROM START — Character data was typed in during an INPUT statement when numeric data was expected, or vice versa. Just retype the entry so that it is correct, and the program will continue by itself. Check the INPUT statements in the program to make sure that the variable name is correct.

RETURN WITHOUT GOSUB — A RETURN statement was encountered, and no GOSUB command has been issued. This often occurs if there is not an END statement between the main program and the subroutines.

STRING TOO LONG — A string can contain up to 255 characters.

?SYNTAX ERROR — A statement is unrecognizable by the Commodore 64. There is a missing or extra parenthesis, misspelled keyword, etc.

TYPE MISMATCH — This error occurs when a number is used in place of a string, or vice-versa.

UNDEF'D FUNCTION — A user-defined function was referenced, but has never been defined using the DEF FN statement.

UNDEF'D STATEMENT — An attempt was made to GOTO, GOSUB or RUN a line number that doesn't exist.

VERIFY — The program on tape or disk does not match the program currently in memory.

APPENDIX B

Commodore 64 BASIC Keywords

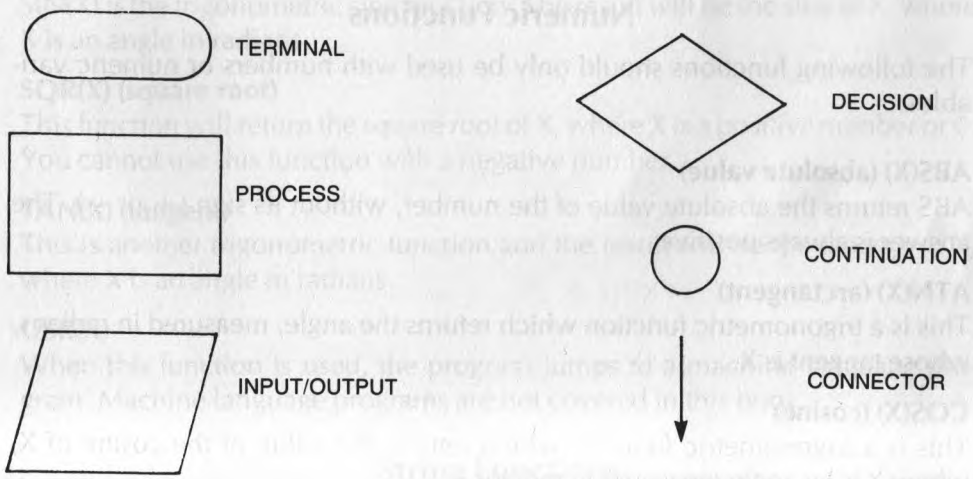
ABS	AND	ASC	ATN
CHR$	CLOSE	CLR	CMD
CONT	COS	DATA	DEF
DIM	END	EXP	FN
FOR	FRE	GET	GET #
GOSUB	GO	GOTO	IF
INPUT	INPUT #	INT	LEFT$
INT	LEN	LET	LIST
LOAD	LOG	MID$	NEW
NEXT	NOT	ON	OPEN
OR	PEEK	POKE	POS
PRINT	PRINT #	READ	REM
RESTORE	RETURN	RIGHT$	RND
RUN	SAVE	SING	SIN
SPC<	SQR	STATUS	STEP
STOP	STR$	STS	TAB(
TAN	THEN	TIME	TI
TIMES$	TI$	TO	USR
VAL	VERIFY	WAIT	

These words and abbreviations should not be used in programs as variable names or as part of a variable name.

APPENDIX C

Flowcharting Symbols

These are the flowcharting symbols that have been used throughout this book. You can buy a template with all these symbols on it.



APPENDIX D

BASIC Functions for the Commodore 64

Numeric Functions

The following functions should only be used with numbers or numeric variables.

ABS(X) (absolute value)
ABS returns the absolute value of the number, without its sign (+ or −). The answer is always positive.

ATN(X) (arctangent)
This is a trigonometric function which returns the angle, measured in radians, whose tangent is X.

COS(X) (cosine)
This is a trigonometric function which returns the value of the cosine of X, where X is an angle measured in radians.

EXP(X) (exponential)
Returns the value of the mathematical constant e (2.71827183) raised to the power of X.

FNxx(X) (user defined function)
Returns the value of the user-defined function xx created in a DEF FNxx(X) statement.

INT(X) (integer)
Returns a whole number. All the decimal places to the right of the decimal point are removed.

LOG(X) (logarithm)
Will return the natural log of X. The natural log to the base e (see EXP(X)). To convert to log base 10, divide by LOG(10).

PEEK(X)
Used to find out contents of memory location X, in the range 0– 65535. The result will always be a number between 0 and 255.

RND(X) (random number)
RND(X) returns a random number in the range 0-1. It is frequently used with the INT function, and the range can be increased by multiplying by the maximum number desired.

SGN(X) (sign)
This function returns the sign (positive, negative, or zero) of X. The result will be +1 if positive, 0 if zero, and −1 if negative.

SIN(X) (sine)
SIN(X) is the trigonometric sine function. The result will be the sine of X, where X is an angle in radians.

SQR(X) (square root)
This function will return the square root of X, where X is a positive number or 0. You cannot use this function with a negative number.

TAN(X) (tangent)
This is another trigonometric function and the result will be the tangent of X, where X is an angle in radians.

USR(X)
When this function is used, the program jumps to a machine language program. Machine language programs are not covered in this book.

String Functions

These functions should only be used with strings or string variables.

ASC(x$)
This function will return the ASCII codes of the first character of X$. The ASCII codes are shown in Appendix VI.

CHR$(X)
This is the opposite of ASC, and returns a string character whose ASCII code is X. Check Appendix VI for the ASCII codes.

LEFT$(X$,X)
Returns a string containing the leftmost X characters of X$.

LEN(X$)
Returned will be the number of characters (including spaces and other symbols) in the string X$.

MID$(X$,X,S)
This will return a string containing X characters, starting from the Sth character in X$.

MID$(X$,S)
This will return part of a string starting at the Sth character.

RIGHT$(X$,X)
Returns the rightmost X characters in X$.

STR$(X)
This will return a string which is identical to the PRINTed version of X.

VAL(X$)
This function converts X$ into a number.

Other Functions

FRE(X)
This function returns the number of unused bytes available in memory. X may have any value and is not used. Note that the FRE(X) will read out in negative numbers, if the number of unused bytes is over 32K.

POS(X)
This function returns the number of the column (0-39) where the next PRINT statement will begin on the screen. X may have any value and is not used.

SPC(X)
This is used in a PRINT statement to skip X spaces forward.

TAB(X)
This is similar to the TAB function on a typewriter. TAB is used in a PRINT statement; the next character will be printed in column x. The value x must be greater than the current position on the line. (Chapter 2)

APPENDIX E

BASIC Commands and Statements

This appendix includes a brief summary of all the BASIC commands and statements available on the Commodore 64. If the command or statement has been used in this book, the chapter where it was introduced is included in parentheses.

CLOSE
This command closes any files used by OPEN statements. The number following CLOSE is the file number to be closed. (Chapters 2,8)

CLR
This command will erase any variable in memory, but leaves the program itself intact. This command is automatically executed when a RUN command is given or a program is loaded.

CMD
CMD sends the output which normally would go to the screen (i.e., PRINT statements, LISTs, but not POKEs onto the screen) to another device instead. This device or file must be OPENed first. The CMD command must be followed by a number or numeric variable referring to the file. (Chapter 2)

CONT (Continue)
This command is used to restart the execution of a program which has been stopped by using the STOP key, a STOP statement, or an END statement within

the program. The program will restart at the exact place where it left off.

CONT will not work if you have changed or added lines to the program, if the program halted due to an error, or if you caused an error before trying to restart the program. In these cases, you will get a CAN'T CONTINUE ERROR. (Chapter 6)

DATA

The statement is followed by a list of items to be used by READ statements. Items may be numeric values or text strings, and items are separated by commas. String items need not be inside quote marks unless they contain a space, a colon, or a comma. If two commas have nothing between them, the value will be READ as a zero or an empty string. (Chapter 5)

DEF FN

This command allows you to define a complex calculation or formula as a function with a short name. In the case of a long formula that is used many times within the program, this can save time and space. The user-defined function is called in a program by using the FN (<variable>) function. (Chapter 5)

DIM

The DIM statement sets aside space in the computer memory for an array. It must be used for arrays with more than 11 elements. (Chapter 5)

END

When a program encounters an END statement, the program halts. You may use CONT to restart the program. (Chapter 2)

FOR-TO STEP

This statement works with the NEXT statement to repeat a section of the program a set number of times. (Chapter 5)

GET

The GET statement allows you to get data from the keyboard, one character at a time. When GET is executed, the character that is typed is assigned to the variable. If no character is typed, then a null (empty) character is assigned. (Chapter 9)

GET #

The GET # statement is used with a previously OPENed device, or file, to input one character at a time from that device or file.

GOSUB

This statement branches to a subroutine. The computer remembers which

program line it last executed before the GOSUB. When a line with a RETURN statement is encountered, the program jumps back to the statement immediately following the GOSUB. (Chapter 5)

GOTO or GO TO

When a statement with the GOTO command is reached, the next line to be executed will be the one with the line number following the word GOTO. (Chapter 5)

IF-THEN

IF-THEN lets the computer analyze a situation and take two possible courses of action, depending on the outcome. If the expression is true, the statement following THEN is executed. This may be any BASIC statement or a line number. If the expression is false, the program goes directly to the next line. (Chapter 4)

INPUT

The INPUT statement allows the program to get data from the user, assigning that data to a variable. The program will stop, print a question mark (?) on the screen, and wait for the user to type in the answer and hit RETURN. (Chapter 2)

INPUT #

INPUT # is similar to INPUT, but takes data from a previously OPENed file or device. (Chapter 8)

LET

LET is rarely used in programs, because it is optional. It can be included before the variable name that is to be assigned the result of a calculation. (Chapter 2)

LIST

The LIST command allows you to look at lines of a BASIC program in memory. You can ask for the entire program to be displayed, or only certain line numbers. (Chapter 2)

LOAD

The LOAD command moves a program into memory from a storage device, such as a cassette recorder or a disk drive. LOAD is usually followed by the program name in quotation marks and the device number. If the device number is omitted, the computer assumes that a cassette recorder is being used. (Chapter 2)

NEW
This command deletes the program in memory, and also clears out any variables that may have been used. Unless the program was SAVEd, it is lost. BE CAREFUL WHEN YOU USE THIS COMMAND. (Chapter 2)

NEXT
NEXT is always used in conjunction with the FOR statement. When the program reaches a NEXT statement, it checks the FOR statement to see if the limit of the loop has been reached. If the loop is not finished, the loop variable is increased by 1 or the specified STEP value. If the loop is finished, execution proceeds with the statement following NEXT. (Chapter 5)

ON
This command turns the GOTO and GOSUB commands into a special form of an IF statement called a 'case' statement. It allows the program to branch to more than one place in the program. (Chapter 4)

OPEN
The OPEN statement allows the Commodore 64 to access devices such as the cassette recorder, disk, keyboard, screen or a printer. (Chapter 2,8)

POKE
POKE puts information, in numeric form, at a specified location in memory. (Chapter 9)

PRINT
The PRINT statement, the first one most people learn to use, is a very versatile statement. PRINT can be followed by:

- text string with quotes
- variable names
- functions
- punctuation marks

Punctuation marks are used to format the data on the screen. The comma divides the screen into four columns, the semicolon suppresses all spacing, and a return starts a new line. (Chapter 2)

PRINT #
PRINT # is followed by a number, which refers to the device or datafile previously OPENed. This number is followed by a comma and a list to be printed. The comma and semicolon have the same effect as they do in PRINT. (Chapter 2,8)

READ
READ is used to assign information from DATA statements to variables, so the information may be used in a program. (Chapter 5)

REM (Remark)
The REMark statement is a note to the user. REM statements in no way affect the operation of the program, except to add to its length. (Chapter 4)

RESTORE
RESTORE resets the pointer in a DATA statement to the first DATA item in the PROGRAM. This allows the information in a DATA statement to be READ more than once. RESTORE stands by itself on a line. (Chapter 5)

RETURN
This statement is always used with the GOSUB statement. When the program encounters a RETURN, it will go to the statement immediately following the GOSUB command. If no GOSUB was previously issued, a RETURN WITHOUT GOSUB ERROR will occur. (Chapter 5)

RUN
This command causes execution of a program when the program is loaded into memory. If there is no line number following RUN, the computer will start with the lowest line number. If a line number is included, the program will start executing from the specified line. (Chapters 2,6)

SAVE
This command will store the program currently in memory on a cassette or disk. The word SAVE is followed by the program name in quotation marks and the device number. (Chapter 2)

STOP
This statement will halt the execution of the program. The message BREAK IN xxx will be displayed, where xxx is the line number containing STOP. The program may be restarted by using the CONT command. STOP is normally used in debugging a program. (Chapter 6)

SYS
SYS is followed by a decimal number in the range 0 - 65535. It is used to call a machine language program or subroutine.

VERIFY
This command causes the computer to check the program on disk or tape against the one currently in the computer memory. This is proof that the program is actually SAVEd, in case the tape or disk is bad or something went wrong during the SAVE command. (Chapter 2)

WAIT
WAIT is used to halt a program until the contents of a memory location change in a specific way. WAIT is followed by a memory location and up to two variables.

APPENDIX F

Abbreviations for BASIC Keywords

This table shows the abbreviations for BASIC keywords for the Commodore 64. Their use is discussed in Chapter 6.

Command	Abbreviation	Looks like this on screen
ABS	A SHIFT B	A
AND	A SHIFT N	A
ASC	A SHIFT S	A
ATN	A SHIFT T	A
CHR$	C SHIFT H	C
CLOSE	CL SHIFT O	CL
CLR	C SHIFT L	C
CMD	C SHIFT M	C
CONT	C SHIFT O	C
COS	NONE	COS
DATA	D SHIFT A	D
DEF	D SHIFT E	D
DIM	D SHIFT I	D
END	E SHIFT N	E
EXP	E SHIFT X	E
FN	NONE	FN
FOR	F SHIFT O	F
FRE	F SHIFT R	F
GET	G SHIFT E	G
GET #	NONE	GET #
GOSUB	GO SHIFT S	GO
GOTO	G SHIFT O	G
IF	NONE	IF
INPUT	NONE	INPUT

Command	Abbreviation	Looks like this on screen
INPUT #	I SHIFT N	I
INT	NONE	INT
LEFT$	LE SHIFT F	LE
LEN	NONE	LEN
LET	L SHIFT E	L
LIST	L SHIFT I	L
LOAD	L SHIFT O	L
LOG	NONE	LOG
MID$	M SHIFT I	M
NEW	NONE	NEW
NEXT	N SHIFT E	N
NOT	N SHIFT O	N
ON	NONE	ON
OPEN	O SHIFT P	O
OR	NONE	OR
PEEK	P SHIFT E	P
POKE	P SHIFT O	P
POS	NONE	POS
PRINT	?	?
PRINT #	P SHIFT R	P
READ	R SHIFT E	R
REM	NONE	REM
RESTORE	RE SHIFT S	RE
RETURN	RE SHIFT T	RE
RIGHT$	R SHIFT I	R
RND	R SHIFT N	R
RUN	R SHIFT U	R
SAVE	S SHIFT A	S
SGN	S SHIFT G	S
SIN	S SHIFT I	S
SPC(	S SHIFT P	S
SQR	S SHIFT Q	S
STATUS	ST	ST
STEP	ST SHIFT E	ST
STOP	S SHIFT T	S
STR$	ST SHIFT R	ST
SYS	S SHIFT Y	S
TAB(	T SHIFT A	T
TAN	NONE	TAN
THEN	T SHIFT H	T
TIME	TI	TI
TIME$	TI$	TI$
USR	U SHIFT S	U
VAL	V SHIFT A	V
VERIFY	V SHIFT E	V
WAIT	W SHIFT A	W

APPENDIX G

CHR$ and ASCII Codes

CHR$ and ASCII codes are used with the function CHR$ and ASC. These functions have the form:

CHR$(x)
x is an integer number between 0 and 255.

ASC("n")
n may be any character.

Some of the characters are duplicated:

Codes 192-223 are the same as codes 96-127.
Codes 224-254 are the same as codes 160-190.
Code 255 is the same as code 126.

Chr. Code	Chr.	Code	Chr.	Code	Chr.	Code	Chr.	Code
0		4	DISABLES SHIFT C=	8		12		16
1	WHT	5	ENABLES SHIFT C=	9	RETURN	13	CRSR ↑	17
2		6		10	SWITCH TO LOWER CASE	14	RVS ON	18
3		7		11		15	CLR HOME	19

Chr.	Code	Chr.	Code	Chr.	Code	Chr.	Code	Chr.	Code	Chr.	Code
INST DEL	20	3	51	R	82		113	BLK	144		175
	21	4	52	S	83		114	↓ CRSR	145		176
	22	5	53	T	84		115	RVS OFF	146		177
	23	6	54	U	85		116	CLR HOME	147		178
	24	7	55	V	86		117	INST DEL	148		179
	25	8	56	W	87		118		149		180
	26	9	57	X	88		119		150		181
	27	:	58	Y	89		120		151		182
RED	28	;	59	Z	90		121		152		183
CRSR →	29	<	60	[	91		122		153		184
GRN	30	=	61	£	92		123		154		185
BLU	31	>	62	]	93		124		155		186
SPACE	32	?	63	↑	94		125	PUR	156		187
!	33	@	64	←	95		126	← CRSR	157		188
"	34	A	65		96		127	YEL	158		189
#	35	B	66		97		128	CYN	159		
$	36	C	67		98		129	SPACE	160		
%	37	D	68		99		130		161		
&	38	E	69		100		131		162		
.	39	F	70		101		132		163		
(	40	G	71		102	f1	133		164		
)	41	H	72		103	f3	134		165		
*	42	I	73		104	f5	135		166		
+	43	J	74		105	f7	136		167		
,	44	K	75		106	f2	137		168		
–	45	L	76		107	f4	138		169		
.	46	M	77		108	f6	139		170		
/	47	N	78		109	f8	140		171		
0	48	O	79		110	SHIFT RETURN	141		172		
1	49	P	80		111	SWITCH TO UPPER CASE	142		173		
2	50	Q	81		112		143		174		

APPENDIX H

Screen Display Codes

The following table lists all of the characters available on the Commodore 64.

There are two sets of characters, but you may use only ONE set of characters on the screen at any given time.

To select Set 1 (uppercase) use:

POKE 53272,21

To select Set 2 (lowercase) use:

POKE 53272,23

Note: If not character is shown for Set 2, then the character for Set 1 will be displayed for both sets.

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	G	g	7	N	n	14	U	u	21
A	a	1	H	h	8	O	o	15	V	v	22
B	b	2	I	i	9	P	p	16	W	w	23
C	c	3	J	j	10	Q	q	17	X	x	24
D	d	4	K	k	11	R	r	18	Y	y	25
E	e	5	L	l	12	S	s	19	Z	z	26
F	f	6	M	m	13	T	t	20	[		27

SET 1	SET 2	POKE
£		28
]		29
↑		30
←		31
SPACE		32
!		33
"		34
#		35
$		36
%		37
&		38
'		39
(		40
)		41
*		42
+		43
,		44
−		45
.		46
/		47
0		48
1		49
2		50
3		51
4		52

SET 1	SET 2	POKE
5		53
6		54
7		55
8		56
9		57
:		58
;		59
<		60
=		61
>		62
?		63
		64
	A	65
	B	66
	C	67
	D	68
	E	69
	F	70
	G	71
	H	72
	I	73
	J	74
	K	75
	L	76
	M	77

SET 1	SET 2	POKE
	N	78
	O	79
	P	80
	Q	81
	R	82
	S	83
	T	84
	U	85
	V	86
	W	87
	X	88
	Y	89
		90
		91
		92
		93
		94
		95
SPACE		96
		97
		98
		99
		100
		101
		102

SET 1	SET 2	POKE
		103
		104
		105
		106
		107
		108
		109
		110
		111
		112
		113
		114
		115
		116
		117
		118
		119
		120
		121
		122
		123
		124
		125
		126
		127

APPENDIX I

Memory Map For VIC II Graphics Chip

Start	Location	Activity
v	53248	Sprite 0 x position
v+1	53249	Sprite 0 y position
v+2	53250	Sprite 1 x position
v+3	53251	Sprite 1 y position
v+4	53252	Sprite 2 x position
v+5	53253	Sprite 2 y position
v+6	53254	Sprite 3 x position
v+7	53255	Sprite 3 y position
v+8	53256	Sprite 4 x position
v+9	53257	Sprite 4 y position
v+10	53258	Sprite 5 x position
v+11	53259	Sprite 5 y position
v+12	53260	Sprite 6 x position
v+13	53261	Sprite 6 y position
v+14	53262	Sprite 7 x position
v+15	53263	Sprite 7 y position
v+16	53264	*most significant bit, x position for sprites 0-7
v+17	53265	VIC II control register
v+18	53266	Raster value register
v+19	53267	Light pen x

v+20	53268	Light pen y
v+21	53269	*Sprite display control
v+22	53270	VIC II control register
v+23	53271	*Sprite expansion and direction
v+24	53272	VIC II address control
v+25	53273	VIC II interrupt control
v+26	53274	VIC II interrupt enable
v+27	53275	*Sprite background priority
v+28	53276	*Sprite multicolor control
v+29	53277	*Sprite expansion x direction
v+30	53278	*Sprite-sprite collision
v+31	53279	*Sprite-background collision
v+32	53280	Screen border color (0-15)
v+33	53281	Background color 0 (0-15)
v+34	53282	Background color 1 (0-15)
v+35	53283	Background color 2 (0-15)
v+36	53284	Background color 3 (0-15)
v+37	53285	*Sprite multicolor register 0 colors (0-7)
v+38	53286	*Sprite multicolor register 1 colors (0-7)
v+39	53287	Sprite 0 color (0-15)
V+40	53288	Sprite 1 color (0-15)
v+41	53289	Sprite 2 color (0-15)
v+42	53290	Sprite 3 color (0-15)

*These locations are used for sprites 0-7. The following table shows the values to be poked at these locations to turn the sprites on and off.

Sprite #	7	6	5	4	3	2	1	0
Turn on	128	64	32	16	8	4	2	1
Turn off	127	191	223	239	247	251	253	255

APPENDIX J

Frequencies for Musical Notes

This table contains the frequencies used in Chapter 10 to write music programs.

Note and Octave	Frequency		Note and Octave	Frequency	
	High	Low		High	Low
C–0	1	12	A#-1	3	187
C#–0	1	12	B-1	3	244
D–0	1	45	C-2	4	48
D#–0	1	62	C#-2	4	112
D-0	1	81	D-2	4	180
F-0	1	102	D#-2	4	251
F#-0	1	123	E-2	5	71
G-0	1	145	F-2	5	152
G#-0	1	169	F#-2	5	237
A-0	1	195	G-2	6	71
A#-0	1	221	G#-2	6	167
B-0	1	250	A-2	7	12
C-1	2	24	A#-2	7	119
C#-1	2	56	B-2	7	233
D-1	2	90	C-3	8	97
D#-1	2	125	C#-3	8	225
E-1	2	163	D-3	9	104
F-1	2	204	D#-3	9	247
F#-1	2	246	E-3	10	143
G-1	3	35	F-3	11	48
G#-1	3	83	F#-3	11	218
A-1	3	134	G-3	12	143

Note and Octave	Frequency		Note and Octave	Frequency	
	High	Low		High	Low
G#-3	13	78	A#-5	59	190
A-3	14	24	B-5	63	75
A#-3	14	239	C-6	67	15
B-3	15	210	C#-6	71	12
C-4	16	195	D-6	75	69
C#-4	17	195	D#-6	79	191
D-4	18	209	E-6	84	125
D#-4	19	239	F-6	89	131
E-4	20	31	F#-6	94	214
F-4	22	96	G-6	100	121
F#−4	23	181	G#-6	106	115
F#−4	25	30	A-6	112	199
G#−4	26	156	A#-6	119	124
A−4	28	49	B-6	126	151
A#−4	29	223	C-7	134	30
B−4	31	165	C#-7	142	24
C-5	33	135	D-7	150	139
C#−5	35	134	D#-7	159	126
D−5	37	162	E-7	168	250
D#-6	39	223	F-7	179	6
E-5	42	62	F#-7	189	172
F-5	44	193	G-7	200	243
F#−5	47	107	G#-7	212	230
G-5	50	60	A-7	225	143
G#-5	53	57	A#-7	238	248
A-5	56	99	B-7	253	46

Frequencies for Musical Notes

Index

ADSR 325, 329, 334, 340, 344-345
ASC II 287
Abbreviations 226-227
Algorithm 49, 361
Alphanumeric information 39,44
Analytical errors 220-223
Animation 398-400
Array 173-183, 342
Attack/Delay 322, 325, 332

BASIC 33, 45, 240
BASIC interpreter 241
BITS 231, 367-372
Back-up copies 213, 225
Binary math 230
Block allocation map (BAM) 56, 58, 59, 62
Border and screen color 289, 338, 366
Branching programs 107
Byte 367-372, 379

CHR$ function 287, 366
CLOSE statement 69, 250, 255, 375
CLR HOME key 25, 26, 287, 383
CMD statement 70
CONT 222
COS function 166
CPU 4, 6
CTRL key 26-29, 82, 284-287
Calculator functions 34-39
Cassette recorder 11-12, 53-54, 248
Cassette tape datafiles 248-252
Cassette tapes 53
Chord 354, 356
Clearing the screen 68, 288, 383
Color keys 26-28
Color memory map 291-300, 409
Commodore key 23, 26-29, 82, 284-287
Constants 40
Continuation (flowcharts) 95-100
Control ports 5, 314
Counters 159-163
Cursor 24
Cursor keys 24, 283, 286

DATA statements 168-169
DATA statements (Sprites) 368, 374, 375, 409
DATA statements (music) 341, 346
DEF FN statement 183
DIM statement 175, 224
DO-UNTIL loop 92-93, 151
DO-WHILE loop 93-94, 163-165, 263
DOS errors 60
Datafile 374-375, 409
Debugging 74, 217
Decimal numbering system 228-230
Decisions (flowcharts) 86-90
Device number 21, 22, 56, 60, 69, 249, 253
Disk Drive 11, 13-19, 248
Disk Operating System (DOS) 55, 60
Disk datafiles 252-257
Disk directory 56-57, 63
Diskettes 15-16, 56
Diskfile error handling routine 255
Documentation 100, 212, 225
Dummy variable 276-280

END-OF-FILE marker 263
Exponentiation 34
FOR-NEXT loop 152-163, 220, 250, 254
FOR-NEXT loop (Music) 329, 346, 361
FOR-NEXT loop (Sprites) 374, 380, 384-385, 394
Fields 246

Flowchart symbols 75
Formatting a diskette 56
Frequency 322, 324, 330, 361
Function keys 28

GET statement 284, 303, 338
GOSUB statement 186
GOTO statement 137-145, 148-152, 220

Harmony 349-361

IF statement 112-120, 220
INPUT statement 49, 64-68
INPUT # statement 251, 256
INST DEL key 25-26, 44
INT function 121-126, 165-166
Indentation 214
Initialize (diskette command) 58-59
Input, process, output 47-48
Insert mode 44
Integer variables 40

Joystick 2, 314-318

Kernal 241
Keyboard 22-28, 284

LEFT$ function 166-168
LET statement 50
LIST command 54, 62, 63, 70
LOAD command 54-55, 62, 63
Logical operators 87, 107, 109, 110, 120, 316, 408
MID$ function 166-168
Machine language 228, 240
Mathematical operators 34-36
Memory map 241, 378
Monitor 6, 8, 330
Most Significant BIT (MSB) 382-383, 385
Multiple statements 225
Musical notes 339, 340, 342, 344, 345, 350-351, 354-356

NEW (disk command) 56-57
NEW command 54
Null string 302, 312
Numeric variable 49

ON-GOSUB statement 191
ON-GOTO statements 132-136
OPEN statement 56, 69, 249, 253, 258, 375
Operating system 241
Oscillators 328

PEEK statement 284, 301, 367
POKE statement 284, 288, 328, 367, 376-378, 394
PRINT statement 34-39, 45-46, 154-155
PRINT # statement 56, 69, 250, 375
Pitch (musical note) 324, 347
Pixel 367, 400
Print zones 155
Printer 19-22, 69-71
Procedural errors 217-220
Pulse 322, 325, 326, 331

Quote mode 43-44

READ statement 169-171, 224
REM statement 136-137, 213, 225
RESTORE statement 171-173
RETURN key 24, 287
RETURN statement 186
RIGHT$ function 166-168
RND function 121-126, 299-300, 363
RUN STOP and RESTORE keys 26, 390
RUN command 46, 62, 223
RVS OFF key 28, 285
RVS ON key 28, 285
Random Access Memory (RAM) 240
Read Only Memory (ROM) 240
Records 246
Relational operators 87, 107, 109-111
Repetitions 148-152
Reserved words 41
Revise (disk command) 60-61, 265
Rhythm 322, 339-341

SAVE command 53, 60-61
SID chip 240, 322, 324-329, 337, 340, 344, 361, 376-368, 375
SIN function 165
ST AND 64, 263
Scientific notation 38
Scratch (disk command) 63
Screen characters 291
Screen editor 241
Screen memory map 291-300, 409
Sequential files 247
Sorting 276
Sound effects 322, 332, 361-364
Spacing 64, 155, 215
Sprite color 387
Sprite display 375
Sprite expansion 389
Sprite movement 384
Sprite pointer 376, 378, 394, 405

Sprite screen map 381
Sprite-background collision 408-409
Sprite-sprite collision 407-408
Sprites 246, 283, 366-415
Strings 39, 64, 173
Subroutines 186-207, 395-397
Subscript 173
Subscripted variables 173-183
Sustain/Release 322, 325, 333
Syntax errors 217

TAB function 68
TAN function 166
TV 6, 7, 330
Top-down-design 224, 226

User defined functions 183-185

VAL function 307, 320, 338
VIC II chip 240, 283, 289, 366, 367, 382, 387, 389
VIC II map 367, 412
Validate (disk command) 61
Variable names 64
Variables 39-43, 172, 252
Verify command 54, 55, 61
Voices 328, 337, 342, 349-358
Volume control 328, 329, 346, 361-362

Waveform 322, 324, 329, 330, 338
Waveform - Noise 324, 330, 332, 338, 363
Waveform - Pulse 323, 325, 330, 331, 338
Waveform - Sawtooth 323, 330, 331, 338
Waveform - Square 323, 338
Waveform - Triangular 323, 330, 335

Computers $15.95

Programming that's fun for the whole family.

COMMODORE 64™ BASICS

A SELF-TEACHING GUIDE

ANN HARRIS

You don't need to be a pro to start creating programs with animated characters, sophisticated graphics and music. In fact, you need no computer experience whatsoever. *Commodore 64 BASICS* makes programming so simple, even a child can do it.

This self-teaching introduction covers everything from using the keyboard and hooking up peripherals, to writing programs in BASIC. It explores the full potential of the Commodore 64's sound and graphics capabilities. The first half of the book focuses on program design, giving you the skills to write personalized programs for games, home management, education, and business. *Commodore 64 BASICS* also defines computer terms, describes the hardware you'll be using, and includes a chapter on computer math.

The second half of *Commodore 64 BASICS* surveys the more advanced BASIC commands—including arrays and subroutines. Chapters cover:

- Color and graphics—color from the keyboard, POKE, PEEK and GET statements and memory maps.
- Sound effects and music.
- Sprites—creating color and sprites, moving sprites, and collisions.
- Data files and file handling.

ANN HARRIS owns her own consulting firm, Computer Access. She designs software and teaches programming classes and workshops for adults and children.

Commodore 64™ is a registered trademark of Commodore Electronics, Ltd.

Wiley Self-Teaching Guides have taught more than three million people to use, program, and enjoy microcomputers. Look for them all at your favorite bookshop or computer store.

WILEY PRESS
a division of JOHN WILEY & SONS, Inc.
605 Third Avenue, New York, N.Y. 10158
New York • Chichester • Brisbane • Toronto • Singapore

ISBN 0 471-88008-6

COVER ILLUSTRATION: PAT PORTER