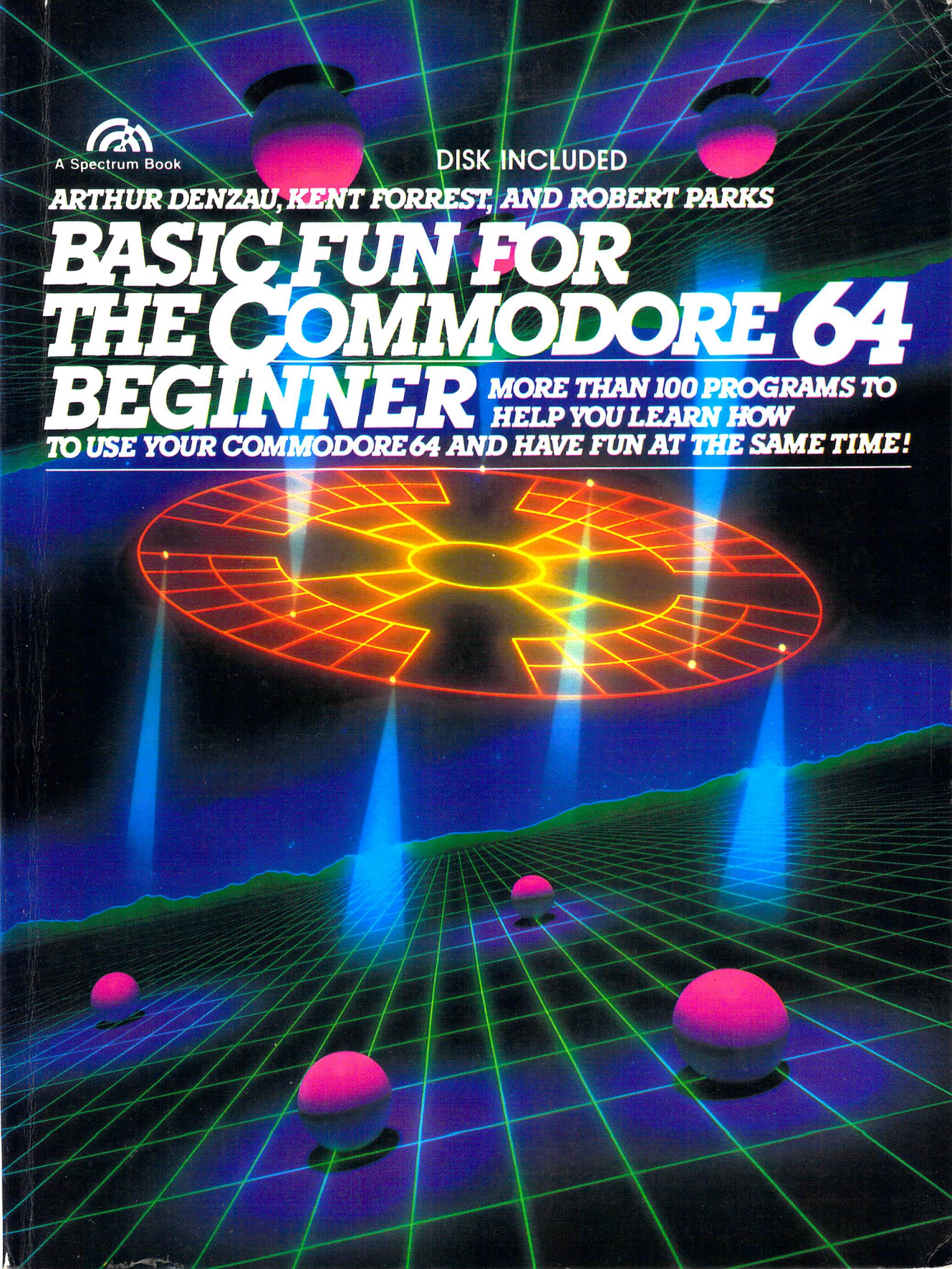

A Spectrum Book
DISK INCLUDED
ARTHUR DENZAU, KENT FORREST, AND ROBERT PARKS
BASIC FUN FOR THE COMMODORE 64 BEGINNER
MORE THAN 100 PROGRAMS TO HELP YOU LEARN HOW TO USE YOUR COMMODORE 64 AND HAVE FUN AT THE SAME TIME!

BASIC FUN
for the
COMMODORE 64
BEGINNER

Arthur Denzau, Kent Forrest, and Robert Parks are computer consultants and associate professors at Washington University in St. Louis, Missouri.

BASIC FUN for the COMMODORE 64 BEGINNER

Arthur Denzau
Kent Forrest
Robert Parks

PRENTICE-HALL, Inc., Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

DENZAU, ARTHUR T.
Basic fun for the Commodore 64 beginner.

"A Spectrum Book."
Includes index.
1. Commodore 64 (Computer—Programming. 2. Computer programs. I. Forrest, Kent L. II. Parks, Robert P. III. Title.
QA76.8.C64D46 1984 001.64'2 84-6944
ISBN 0-13-061441-6 (Pbk. w/disk)
ISBN 0-13-061490-4 (Pbk.)

This book is available at a special discount when ordered in bulk quantities. Contact Prentice-Hall, Inc., General Publishing Division, Special Sales, Englewood Cliffs, N.J. 07632.

10 9 8 7 6 5 4 3 2 1

Editorial/production supervision and interior design by Jane Zalenski
Manufacturing buyer: Joyce Levatino
Cover design by Hal Siegel
Front plate photo courtesy of Michel Tcherevkoff

ISBN 0-13-061490-4 {PBK.}

ISBN 0-13-061441-6 {PBK. W/ DISK}

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA PTY. LIMITED, *Sydney*
PRENTICE-HALL CANADA INC., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA PTE. LTD., *Singapore*
WHITEHALL BOOKS LIMITED, *Wellington, New Zealand*
EDITORA PRENTICE-HALL DO BRASIL LTDA., *Rio de Janeiro*

CONTENTS

PLEASE READ ME

This book is intended primarily for Commodore 64™ owners who want to learn more about their machine and how to program it. The book has two basic themes:

1. The easiest way to learn programming is by making changes in program models.
2. The programs you work with should be interesting, useful, and fun.

Have we got a book for you!! You can find in it the following:

Musical instruments to play

Puzzling challenges

A fast-paced video game

An artist's sketchpad

A $500 digital watch (for those with *large* wrists and a really long extension cord)

A band of 24 roaming gorillas (would you believe three monkeys?)

OK, back to more serious stuff, for example:

An inexpensive collection of useful programs

An explanation of how to use the Commodore 64 screen editor

A guide to the use of your disk drive

Essays that show you how to control the special powers of your Commodore 64

A guide for making and fixing programming errors

Hints on modifying the programs that you can use in your own programs

Features added step by step so you can see how to add them to your own programs

An index to programming routines used in the model programs

A series of tables that will aid you in understanding and writing programs

Programs that let you use your 64 as a tutor

A guide to public domain programs available from Commodore

A guide to Commodore user groups

Before going ahead to work with the programs, you should read the next chapter, "Preliminaries." We also recommend that you reread pages 114–117 of your *Commodore 64 User's Guide* (which came with your machine). These pages review all the essential commands you will need to run these programs.

BASIC FUN for the COMMODORE 64 BEGINNER

0*

PRELIMINARIES

The Screen Editor (and the Cursors)

This is not the name of a New Wave rock group, as you may have feared (or hoped). Instead, it is the name of a powerful feature built into your 64. Sooner or later, you will make a typing or programming mistake. The screen editor greatly simplifies editing such typing errors.

The screen editor provides several ways to edit any mistake, and we provide this guide to all of them. In this section, we show you how to:

Clear the screen
Delete / replace a line
Delete / insert a character
Duplicate lines
Avoid the "RUNDY." error

Clear the Screen

Locate the <CLR/HOME> and the <SHIFT> keys on your keyboard. Press the <SHIFT> key and the <CLR/HOME> key at the same time to clear all text from the screen and move the cursor (the blinking rectangle on the TV screen) to the upper left-hand screen corner.

*Computers always start counting at 0!

Delete a Line

This is the simplest form of editing. To delete a line, simply type the line number followed by a <RETURN>. This removes the line from memory.

> Remember to press the <RETURN> key after each line.

With all of the following examples, please note that <RETURN> means you should press the <RETURN> key on the Commodore 64.

```
NEW <RETURN>
10 PRINT "THIS IS MY FIRST PROGRAM" <RETURN>
LIST <RETURN>
```

Now type:

```
10 <RETURN>
LIST <RETURN>
```

The screen should no longer display line number 10.

Replace a Line

To replace a line, you type the line # followed by the new data and end with the <RETURN> key. In other words, you are simply typing the line over again. For large changes where the whole line needs to be retyped, this is the easiest thing to do. To change a single character, or just a few, there is an easier way to do this.

Delete a Character

Let's type in a simple (mistaken) program and correct the error:

```
10 PRIUNT "THIS IS LINE ONE." <RETURN>
```

Locate the <INST/DEL> and the <SHIFT> keys on your keyboard. Now locate the two keys marked <CRSR> at the bottom right of your keyboard. Note they are marked differently, one with an up and down arrow, and the other with a left and right arrow. By pressing these keys, you move the CURSOR in that direction. Press the <SHIFT> and hold it down. Then press and release the <CRSR> (up/down cursor) key. (<CRSR> is located directly below the <RETURN> key.)

By holding down the <CRSR> key along with the <SHIFT> key, you can cause the cursor to continue all the way up to the top of the screen. Using the <CRSR> key alone moves the cursor down. Try moving the cursor up and down and then place the cursor at the beginning of line 10.

Use the <CRSR> (right/left cursor) key to move the cursor to the right

within the line. Again, note that holding down the <SHIFT> key while also pressing the (right/left) <CRSR> key reverses the direction of cursor movement. Move the cursor so it is blinking on top of the N in PRIUNT. Now press the <INST/DEL> key at the top right of the keyboard. This should gobble the "U" to the left of cursor and correct the error.

The general idea is to move the cursor to the right of the character to be deleted. Each time you press the <INST/DEL> key, the character to the LEFT of the cursor will be deleted and the remaining characters to the right will close ranks (move to the left).

Insert a Character

Let's again try an example first:

```
20 PRINT "HIS IS LINE TWO."
```

We wish to change the HIS to THIS. To do so, you need to position the cursor so that it is blinking over the H. To change the H to TH, hold down the <SHIFT> key and press the <INST/DEL> key. This should INSerT a blank and move the rest of the line to the right by one character. Type T and press <RETURN> to complete the editing.

The general rule to insert a character is to move the cursor to lie on top of the first character *after* the text to be inserted. Each time you press the <SHIFT> and the <INST/DEL> keys, a "space" is inserted to the right of the cursor. Type in the missing character(s) in the "blank space(s)" followed by a <RETURN>. The "blank spaces" are not exactly spaces; rather, they are empty holes into which you can type a character.

Duplicating a Line

One of the really slick things your 64 editor can do is make duplicate lines with a minimum of effort. Move the cursor to the line you wish to duplicate and type over the existing line # with a new line #. This will create a duplicate line with the new line number when you press the <RETURN> key. The original line is not affected.

Why Your 64 Produces RUNDY.'S

You do not have to move the cursor across the entire line for line duplication to take place. This small fact might someday be the source of a RUNDY. error. For example, if the computer screen has been filled with text and the READY. message is displayed on the screen line that you are using, when you type RUN <RETURN> to execute your program, you will see RUNDY. and a SYNTAX ERROR. The RUN was typed on top of the READY. message, and the 64 does not know how to RUNDY. One solution is to clear

the screen and type RUN again. A second is to "cursor" up or down to a blank line and type RUN. Finally, if your screen is filled, you can simply type:

```
RUN:
```

The colon after the command causes the computer to ignore the rest of the text on the line.

A Quick Practice Session

Type in the following program and correct the typing errors using the screen editor.

```
NEW <RETURN>

10 PRUNT "MY NAMEE IS "          <RETURN>
20 PRINT "DE COMMODOREE 64"      <RETURN>
30 PRINT "WAT IS YOUR MAME?"     <RETURN>

LIST <RETURN>
```

Please note that if you edit a line, you *must* press <RETURN> to tell the 64 that you want that line changed. You can always move the cursor to any part of a line and hit <RETURN> to cause the line to be saved as part of your program, but without a return the 64 will not know what changes you have made.

Disk Driver

Here are some helpful hints about using the disk drive, which may not be clear from reading your disk drive manual. It is assumed you have read the previous section, "The Screen Editor," and have some understanding of the <CRSR> keys. It is extremely important that you type the commands and programs exactly as shown.

Turning On Your Computer

Some problems with the disk drive can be caused by improper electrical connections or turning on the equipment in the wrong order. Make sure the disk drive, Commodore 64, and the monitor (TV) are all properly plugged into three-prong live outlets. Do not attempt to defeat the three-prong plug because it serves an important purpose—properly grounding your computer will avoid serious and costly problems.

While the machine is OFF, be sure you have the cables properly connected between the disk drive and the computer. Next, insert the sample

diskette called VIC-1541 Test/Demo (which came with your drive) into the drive. Make sure that the end of the diskette with holes that expose the diskette's surface goes into the drive first.

Now, turn the machines ON in the following order:

1. computer,
2. disk drive, and finally,
3. printer or second disk drive, if any.

(You can turn ON your TV or monitor at any time, if you wish to see what the 64 is doing.)

You will find that trying to use two disk drives and a printer will often result in one of the disk drives "locking up." Failing to follow these instructions will result in no material harm to the disk drive, but it may not work properly.

The red and green lights on the drive may both be on while the disk drive is starting up. If things are OK, then after a moment only the green light should be on. When first turned on, your disk drive starts to run a built-in program called the Disk Operating System, or DOS. One of the simplest things that DOS does is tell the computer that a disk drive is connected to it. If you have trouble by this point, turn everything off and start over again.

The DOS Wedge

On the Test/Demo diskette is a program called C–64 WEDGE, which we will LOAD and RUN.

To do this, type:	The screen will display:
LOAD "C–64*",8 <RETURN>	SEARCHING FOR C–64* LOADING READY.
RUN <RETURN>	DOS MANAGER version/date (may vary) BY BOB FAIRBAIRN READY.

The "wedge" will be installed. The wedge abbreviates command names for ease of typing and makes life easier by automatically reporting disk errors. To discover how to use the commands that the wedge provides, we found a program on the disk called HOW TO USE. If you have the DOS WEDGE installed, then you should be able to do the following—type

@$ <RETURN>

and the catalog will be displayed. Move the cursor up to the line that has HOW TO USE, type an "up arrow" (the key just to the left of <RESTORE>), and press the <RETURN>. That should run the HOW TO USE program. The Commodore 64 *Software Bonus Pack* documentation also has a good discussion of the wedge.

If you cannot get the wedge loaded and installed, you can still see the program names on the catalog by typing:

```
LOAD "$",8 <RETURN>
```

After you see the cursor flashing, type:

```
LIST <RETURN>
```

How to NEW (FORMAT) a Disk

Like a blank audio tape, diskettes come from the manufacturer without anything written on them. But the diskette, unlike the audio tape, needs to be organized so that once the computer and disk drive store information on it, it can be retrieved again. We call this procedure to FORMAT or to NEW a disk.

This "formatting" works just like a city map—a system of streets and house numbers must exist if any of us ever expect to receive a letter. The computer refers to its own addresses as TRACKS and BLOCKS.

Display a catalog again:

```
@$ <RETURN>
```

The bottom line shows the number of BLOCKS that are still empty (available) for your use. An empty diskette has 664 empty BLOCKS, or 664 × 254 (letters or numbers) = 168,656 characters, or space for about 24,000 words, or 100 typed double-spaced pages.

The following steps supply the simplest method to NEW a disk. (If your wedge is already installed, start at step 3.)

1. Place the VIC-1541 Test/Demo diskette in your disk drive and make sure that the Commodore and the drive are turned on—Commodore first, drive second.
2. Type:

   ```
   LOAD "C-64*",8 <RETURN>
   RUN <RETURN>
   ```

   The display should be "DOS MANAGER," a version number and date, the author's name, and a copyright notice, indicating that the DOS WEDGE has been properly installed. If you have trouble by this point, turn everything off and start over again.

3. Place a brand-new diskette in the disk drive and type the following:

   @N:MY 1ST DISKETTE,A1 <RETURN>

   The READY. and cursor should reappear on the screen. The red light on the disk drive should be on, and the disk drive should be making its usual grunts and groans.

   Within a minute or so, the red light on the disk drive should be off. When it is, type:

   @$ <RETURN>

   On the first line, in reverse display, you should see:

   0 MY 1ST DISKETTE A1 2A

This is the "catalog" (listing of programs), which shows programs or "data" stored on this particular diskette. Only the title information is shown because you have not yet saved a program.

You have named your diskette MY 1ST DISKETTE and have assigned an arbitrary volume ID, A1. The name and volume ID are used in some copy programs, and it is a good idea to use a different volume ID for each diskette. You can use any two characters, including graphics, as the volume ID symbol.

The 2A located to the right of the volume ID is *one* of the methods Commodore uses to indicate the current DOS version. The 2A is generated by the computer and cannot be changed by you. The code may differ to indicate a different DOS version.

Saving Programs

Now put the VIC-1541 Test/Demo diskette in your drive. Type @$ to display the catalog. Load the program called HOW TO USE by typing:

/HOW TO USE <RETURN>

or type:

LOAD "HOW TO USE",8 <RETURN>

The / (slash) is the 64–WEDGE command to LOAD a program. To SAVE this program to your diskette, open the drive door and remove the Test/Demo diskette, placing it back into the protective sleeve. Next, insert MY 1ST DISKETTE into the drive and close the disk drive door. Then type in the following:

←HOW TO TEST <RETURN>

If you do not have the WEDGE running, you should type:

```
SAVE "HOW TO TEST",8 <RETURN>
```

With the wedge, there is no need to use ending quotation marks around the program name. In addition, you don't need to tell the computer the drive number (8) as before—it will assume you mean 8 until told otherwise. The red drive light should go on when the <RETURN> is pressed. When it goes off, the program has been SAVEd.

Type:

```
@$ <RETURN>
```

You should see:

```
0   MY 1ST DISKETTE A1   2A
13   "HOW TO TEST"        PRG
651 BLOCKS FREE
```

Now clear the 64's memory by typing:

```
NEW <RETURN>
```

Unfortunately, the term NEW means two very different things to your Commodore 64. When used in a diskette command, it means erasing and formatting a diskette. But a second use of the term is a BASIC language command meaning to clear the 64's memory of previous programs. This is what we are doing now. Have no fears—your diskette will not be harmed.

Now type in the following program:

```
10 FOR I=1 TO 10 <RETURN>
20 PRINT I <RETURN>
30 NEXT I <RETURN>
SAVE "MY FIRST PROGRAM <RETURN>
@$ <RETURN>
```

This should SAVE the program and then display the catalog. Now type:

```
RUN <RETURN>
```

You should see 1,2,3,4,5,6,7,8,9,10 on successive lines. Now LIST the program and change line 10 to the following:

```
10 FOR I=1 TO 5 <RETURN>
```

Again, a gentle reminder: Use your screen editor to do this without retyping the entire line.

To replace the version on the diskette with this version, try typing:

```
SAVE "MY FIRST PROGRAM
```

Oops! We forgot to tell you to put a @: in front of the program name. If

you do not use the @: before the program name, the 64 will display the error message FILE EXISTs. Let's do it right and type:

```
SAVE "@:MY FIRST PROGRAM
```

You should always use WEDGE commands for telling your disk drive what to do. If you do not, any errors that the disk drive experiences will be indicated by a blinking red light on the drive. If this should ever happen while you have the wedge installed, you only need to type

```
@ <RETURN>
```

to restore your drive to its normal condition. A (possibly cryptic) error message about why the drive did not like what you typed will appear on the screen. If you do not have the wedge installed, your disk drive manual shows a different, more tedious way to get the error message.

If you have a drive error and the wedge is not installed, you should then install the wedge. This can be done *without* destroying the program in memory. Remove the current diskette from your drive and replace it with the VIC-1541 Test/Demo diskette. Type:

```
LOAD "DOS*",8,1 <RETURN>
SYS 52224 <RETURN>
```

(Be sure to type the ,1 or the program you are loading from the disk will come into the 64 right on top of your old program, destroying it.) The DOS MANAGER version should be displayed and the wedge is now ready for use.

SCRATCHing (Erasing) a Program from a Disk

Suppose you wish to scratch (erase or delete) a program from your diskette (that is, you no longer want it), and the program name is MY FIRST SCRATCH. Type the following:

```
@S0:MY FIRST SCRATCH
```

What if you made a spelling mistake or there is no program named MY FIRST SCRATCH on your diskette? The 64 does not consider it an error to scratch a file that doesn't exist, and no error message is displayed. We recommend that you always display a catalog after SCRATCHing a program to make sure that it happened. A misspelling will be ignored.

Use of the Wild Card

The DOS wedge allows the use of an * as a wild card. Put your Test/Demo diskette into the drive and type:

```
@$HOW*<RETURN>
```

This displays a catalog listing with all the programs that start with HOW. The * acts just like a wild card and "matches" any program that has HOW as its first three letters. Remember how we said to load the wedge (LOAD "C-64*",8)? This used the wild card to match the first program name that began with C-64, which happens to be C-64 WEDGE. Try using the wild card to produce different catalogs and/or particular programs. While you're playing, try this:

/HOW*<RETURN>

The command is ambiguous because there are two different programs that can match the wild card—HOW TO USE and HOW PART TWO. DOS is pretty simple and just LOADs the first program it finds that matches.

The wild card is very dangerous and should usually not be used in the scratch command. One of the coauthors still cries because he entered the command @S0:*TO USE, thinking that this would scratch just the HOW TO USE program and (gasp!) lost all of the programs on the diskette. The lesson is that anything you put *after* the * is ignored!

If you have a diskette with programs named PROG 1, PROG 2, and PROG 3, it is possible to SCRATCH all three of them with the command:

@S0: PROG*

but unless you are really sure of what you're doing, it is usually safer to simply SCRATCH them one at a time.

The Disk Full Error Message

Suppose that after trying to save a program, the red light continues blinking, and typing

@ <RETURN>

produces a DISK FULL ERROR or ILLEGAL TRACK OR SECTOR. Try typing:

@$ <RETURN>

If the catalog shows some BLOCKS FREE, then you need to do a "garbage collection" on the disk. You have enough room on the disk to record data, but the room is not in one place. Garbage collection puts all the free space in one place where it can be used. Simply type:

@V <RETURN>

The cursor will return immediately, but the red light on the disk will be on and it will be making its usual noises. The V stands for Validate. You might want to use the @V command on diskettes to which you often save programs.

DOS Wedge Commands

Command	*Function*
@	Display disk status—that is, errors.
@$	Display diskette's catalog.
@$:MINE	Display catalog entry for file MINE.
@$:MINE*	Display all files beginning with MINE.
@N:DISK NAME,F1	New a disk with name DISK NAME and volume ID F1
@R:NEWFILE=OLDFILE	Rename the file named OLDFILE with the new name NEWFILE.
/FILENAME	Load the program named FILENAME.
↑FILENAME	Load and run the program named FILENAME.
←FILENAME	Save the program in memory and call it FILENAME.
@S:FILENAME	Same command as above.
@C:NEWFILE=OLDFILE	Copy the file named OLDFILE and name the copy NEWFILE.

What If . . . (It Doesn't Work)?

OK, you say you typed the program as it was listed and it still doesn't work. Now what? While it is not possible to cover every error, the following list of suggestions may get your program running. *Above all, remain calm.*

1. Refer to the ERROR MESSAGE TABLE that follows to determine the probable cause of the specific error message generated. This is your best clue as to where you should start in finding a remedy.
2. LIST the line indicated with the error, for example,

   LIST 110 <RETURN>.

3. Check for typing errors, truly the most common problem. They usually cause the error message SYNTAX ERROR IN LINE #n.

   Common Syntax errors include:

   Crunched lines (the first line wraps around, giving only the appearance of a new line)

   Type in the following WITHOUT hitting <RETURN> after line 100:

   ```
   NEW <RETURN>
   100 PRINT "O.K., THIS IS THE FIRST LINE"
   110 PRINT "AND THIS IS THE SECOND" <RETURN>
   RUN <RETURN>
   LIST <RETURN>
   LIST 100 <RETURN>
   LIST 110 <RETURN>
   ```

This set of crunched lines will cause a SYNTAX ERROR IN LINE 100 to be generated when the program is RUN. Usually, the cause of the problem is that you started to type line 110 without first typing a <RETURN> for the previous line. As in the example, if the first line is exactly 40 characters long, you may not realize you forgot the <RETURN>. To fix the crunch, duplicate line 100 as line 110. LIST the program. The fact that you now have three line 110s may be confusing. Use cursor commands to move up to the beginning of the last 110. Press the <INST/DEL> key until the 110 is at the beginning of the top line. Press <RETURN> to create a good line 110. Type LIST 100–110.

```
100 PRINT "O.K., THIS IS THE FIRST LINE"
110 PRINT "AND THIS IS THE SECOND"
110 PRINT "AND THIS IS THE SECOND"
```

Now cursor up to the end of the first line 110. Again press the <INST/DEL> key until you have erased that whole line 110 on the screen and the cursor is on line 100. This should remove the bad portion of line 100, and the crunching has been fixed.

- Missing parentheses

  ```
  10 PRINT A*(B+(C+2*D) <RETURN>
  ```

- Missing commands

  ```
  10 "ABCD" <RETURN>
  ```

- Misspelled commands, for example, PRUNT for PRINT, or a semicolon in place of a colon.

4. Note that after a program stops, even if due to an error, you can print any of the variables in immediate mode, for example, PRINT A . If the value displayed is not what you expected, it may be necessary to go through the program line by line to find your mistake. Gasp! Sigh!
5. If you are no longer patient and cannot find the problem, much less cure it, then you might
   - Contact a local microcomputer users group. See Appendix 2 at the end of this book for a list of such groups.
   - Turn off the computer and start over again later.
   - Try obtaining help from a local computer store.

Table Of Error Messages

The programs in the left column will produce the error messages in the right-hand column. we are sure you can produce the error messages without our help, but since we have had such great experience doing it ourselves, we

offer a few examples. Note that other programs can produce the same error messages shown below and not have the slightest resemblance to our programs below. Our programs are not the only way to get the error message but simply one way to get them.

If you are typing in these examples, remember to type NEW between each entry. Beginning with this section, we will no longer put a <RETURN> at the end of each line—you are expected to type it, however, to enter the line into memory. After typing in each program, type RUN to produce the error message. In these short programs, there is no need to type the REM statements.

PROGRAM TO GENERATE ERROR	*ERROR MESSAGE GENERATED*
100 PRINT HELLO 110 REM NO QUOTES FOR STRING VALUES YIELDS A 0 FOR VARIABLE NAMED HELLO	0
110 PRINT 1/A 120 REM DIVISION BY ZERO IS ILLEGAL	DIVISION BY ZERO ERROR
DEF FNA(X) = 3 * X	ILLEGAL DIRECT
REM YOU CAN'T DEFINE A FUNCTION IN IMMEDIATE MODE. NOTE, THERE IS NO LINE NUMBER.	
100 POKE 25,256 110 REM NUMBER AFTER THE COMMA MUST BE LESS THAN 256	ILLEGAL QUANTITY
100 FOR I = 1 TO 10 110 FOR J = 1 TO 10 200 NEXT I 210 NEXT J 220 REM CROSSED LOOPS ERROR	NEXT WITHOUT FOR IN LINE 210
100 READ A,B,C 1000 DATA 4,4 1010 REM MISSING DATA FOR C	OUT OF DATA IN LINE 100
100 GOSUB 1000 1000 A = A + 1 1010 PRINT A 1020 GOTO 100 1030 REM THE COUNTER IN 1000 COUNTS THE LEVELS OF GOSUBS	OUT OF MEMORY

Table of Error Messages *(continued)*

```
1040 REM MAX NUMBERS OF NESTED
     LOOPS IS 23
```

```
1000 REM ANOTHER METHOD                    OUT OF MEMORY
1010 DIM A(30000)
1020 REM POSSIBLE ONLY WITH ARRAYS.
1030 REM REDUCE SIZE OF ARRAYS.
1040 REM MAX IS 7700 for A(7700)
1050 REM OR A%(19400)
```

```
100 A = 2                                  OVERFLOW ERROR IN 110
110 A = A * A
120 PRINT A,
130 GOTO 110
140 REM CAN'T PRODUCE ANY
    HIGHER NUMBER
150 REM LIMIT IS ABOUT
    1.7 E+38
160 REM OR 1.7 TIMES
    10000. . .0000000     (38 ZEROES)
```

```
100 DIM A(10)                              REDIM'D ARRAY ERROR
110 DIM A(10)                              IN 100
120 REM CANNOT REDIMENSION
    AFTER THE FIRST DIM STATEMENT.
```

```
100 RETURN                                 RETURN WITHOUT GOSUB
                                           IN 100
```

```
100 FOR I = 1 TO 400                       STRING TOO LONG ERROR
110 A$ = A$ + CHR$(3)                      IN 110
120 PRINT I;
130 NEXT
150 REM STRING CANNOT
    EXCEED 255 CHARACTERS
    IN LENGTH
```

```
100 PRUNT 1                                SYNTAX ERROR IN LINE 100
110 REM THIS IS BUT ONE
    EXAMPLE OF PRODUCING A
    SYNTAX ERROR.
```

```
100 A$ = 23                                TYPE MISMATCH ERROR
110 A  = "ABC"                             IN 100
120 REM A STRING CAN BE
    REPRESENTED ONLY BY A STRING
    VARIABLE.
```

Table of Error Messages *(continued)*

100 PRINT FNA(3) 110 REM FUNCTION MUST BE DEFINED IN PRIOR STATEMENT.	UNDEF'D FUNCTION ERROR IN 100
100 GOSUB 1000 1005 REM WRONGLY NUMBERED IN LINE 100 1010 PRINT A	UNDEFINED STATEMENT ERROR
100 INPUT X,Y RUN 1	?? (the 64 is expecting more input)
100 INPUT X RUN 1,2	?EXTRA IGNORED (too much information)

The following errors are rare—for real error experts only.

FILE DATA

FORMULA TOO COMPLEX

CAN'T CONTINUE

FIND.STRING

If you want to see all the error messages that the 64 has, RUN the following program. Remember to type NEW before typing in the program.

```
100 BE = 40960
110 INPUT "TYPE STRING TO BE FOUND";FI$
120 PRINT CHR$(147)  :   REM CLEAR
130 PRINT "LOOKING FOR ";FI$
140 PRINT "BE PATIENT!!!  I'M SEARCHING."
200 FOR I = BE TO BE + 8192
210 FOR J = 1 TO LEN(FI$)
220 IF (PEEK(I+J-1) AND 127) <> ASC(MID$(FI$,J,1)) THEN 290
230 NEXT J
240 PRINT FI$; " STARTS AT";I
250 END
290 NEXT I
```

PRINT.MEMORY

This program is handy to examine any memory location in the BASIC ROM. By modifying the memory location in line 130, you can examine any portion of the computer's memory.

```
80   REM 40960 TO 49152 ARE THE BASIC ROM LOCATIONS.
90   PRINT "THE NO. SHOULD BE BETWEEN 40960 - 49152."
100  INPUT "TYPE INITIAL MEMORY LOCATION TO BEGIN SEARCH";BE
110 PRINT CHR$(14)
120 PRINT CHR$(PEEK(BE));
130 BE = BE + 1
140 IF BE < 42000 THEN 120
```

A Guide to this Book

This book has eight chapters of programs. In each chapter you will find programs and at least three other sections: BASIC commands used in the chapter, programming techniques used in the chapter, and challenges.

The BASIC commands are simply listed—if you want to know more about them, the *Commodore 64 User's Guide* briefly discusses them in Appendix C. The programming techniques section provides an explanation of some of the programming techniques used in the programs in that chapter. It is best read when you are typing in the program. There is also a list of all of the programming techniques at the end of the book.

In most chapters there will also be an explanation section that discusses either the special feature of the 64 or the special technique that is the topic for that chapter. These discussions are sometimes difficult, due to the difficulty of explaining the 64 to you. We hope that they will aid you in having fun with the 64. Please do not let them get you down. We want you to have fun and have provided the explanations so that you can have more fun. If they are not fun, type in the programs and run them; then come back and read about what is going on.

Before most of the program listings, we discuss some modifications of the program that might be made. Many of the programs are modifications of the previous programs. We purposely did this to help you learn. For example, in Chapter 1, "A Cheap Typewriter," you could type in the program PRINTING TYPER. But you would miss out on half the fun of building it piece by piece as we did with the five programs that preceded it.

Each program listing in this book begins with a REMark statement showing the name of the program. Even if you type in no other REMarks, *always* use a REM containing the program name as the first line of your program.

The REM statements that follow the first, in line numbers up to 30, state two things. First, if the program is a modification of some other program, this is stated along with the changes. Second, the REMs may also contain suggested changes in the program. All of the remaining REMs in the program are to help you know what the statement or section is doing. You do not have to type in the REMs (although it is good practice). If you do, please note that we have put in a lot of spaces on some lines to make the listing look

nicer. Since the 64 has only 40 columns, you will not be able to put in as many spaces and stay on one line. You can follow our example, though, in making your programs look better by putting in REMs that begin in certain columns and by using the null statements (do nothings)—a line # and a colon.

Throwing Down the Gauntlet

Every chapter ends with a set of challenges. The challenges are suggestions for major creative changes to our model programs. We have not supplied you with hints about how to resolve these challenges. Instead, we leave it to your creativity and will publish the most interesting responses in a future volume.

For those of you who accept the gauntlet (the Challenge), send a listing to:

Commodore 64 Fun Book Editor
Dept. 53248
226 Wenneker
St. Louis, Mo. 63124

If you wish a reply, please include a stamped, self-addressed envelope.

If your computer experience is limited, we again would like to suggest you reread your *Commodore 64 User's Guide* that came with your machine. Pages 114–117 review all the BASIC commands you need to start working with our programs.

1

A CHEAP TYPEWRITER

Programs:

TYPER	This simple program allows you to type text on a white screen.
TYPE WITH CURSOR	This routine adds a flashing cursor to TYPER.
SOUND	This addition adds a click to your typewriter whenever a key is pressed.
LOWER CASE	This program allows you to input and display both upper- and lower-case keyboard.
ADD COLOR	This feature allows each character to be displayed in a new color.
PRINTING TYPER	This is the final version of your typewriter.
CLICK	This is the routine used in SOUND. You can add this feature to many programs.
SCREEN PRINT	This is the routine used in PRINTING TYPER that will print your screen.

The programs in this chapter will make your 64 act like a sophisticated electronic typewriter. By the time you have completed PRINTING TYPER, your typewriter will be able to do the following:

1. Display what you type on the keyboard to the screen.
2. Display text with flashing cursor.
3. Edit with full screen capabilities.
4. Simulate the click of typewriter keys.
5. Display text and graphics in multicolors.
6. Print your screen onto your printer.

BASIC Commands Used in This Chapter

AND	IF
CHR$	NEXT
CLOSE	OPEN
CMD	PEEK
DATA	POKE
DEF	PRINT
FOR	READ
GET	REM
GOSUB	RETURN
GOTO	

Programming Techniques Used in This Chapter

1. *Flash a cursor.* Actually, what is done is to make the character appear in normal mode and then in inverse—it then "appears" to be flashing. TYPE WITH CURSOR, line 530, finds where in memory you are about to type, and then line 550 and line 590 make normal/inverse for that character in memory. It helps to know that inverse is made on the screen by having the value of the character be greater than 128. For example, 1 is the numerical value for the letter A (see Appendix E in the *Commodore 64 User's Guide*). Then, when an A is on the screen, we know that a 1 is in that memory location (the screen uses memory locations 1024–2023). When the A is shown in inverse, then 129 is in that memory location. To flash the character A, we simply alternate poking 1 and 129 to the memory location on the screen where the A should be. Try typing (make sure that there is a character at the very top left of your screen):

```
POKE 1024,1
POKE 1024,129
```

Now the problem is to figure out where the "cursor" is, that is, where the next character will be placed on the screen. Line 530 in TYPE WITH CURSOR finds the cursor location in the variable CL. Then, if we PEEK at that location (CL has the cursor location), we will know what character is there (usually a blank). We want to add 128 if the value is less than 128 and subtract 128 if the value is greater than 128, and then put (POKE) the new value back at the cursor location. The AND statement used in line 590 of TYPE WITH CURSOR does this rather than having two IF statements (one for add, one for subtract). Any number ANDed with 255 will now be less than 255 and will have 256 subtracted from it if it is greater. For example, 500 and 255 is 244 (500–256=244). The statement

```
(PEEK(CL) + 128) AND 255
```

is shorter and better than coding two IF statements if you know what the AND does. To see what AND does, try the following program, replacing the 255 with other numbers:

```
10 FOR I = 1 TO 1000 STEP 50
20 PRINT I AND 255
30 NEXT I
```

2. *Defining functions.* This makes a more readable program. See lines 50 and 70 in ADD COLOR where the functions are defined and line 630 where they are used. You should note that the second function is defined in terms of the first.
3. *Screen printing.* There is more to this than meets the eye. The characters that are displayed on the screen are stored as numbers between 0 and 255, but the numbers that are stored are not the same numbers that should be sent to the printer. Hence, one has to fix things up a bit. See SCREEN PRINT, line 30080, to see what character is stored for display. Then, depending upon upper/lower case or upper case/graphics, lines 30140 to 30200 make the translation from what you see to what you get (so to speak).

Typer

This simple program will display what you type on the keyboard onto the screen. It sets the background and border colors to white. It displays what you type in black. If you desire to change either the color of the background or the text color, then see the appropriate values listed in Table 2 ("Colors") at the end of this book. To stop the program, press the <RUN/STOP> key. To restore your 64 to its original state (blue screen, and so forth), hold down the <RUN/STOP> key while you also press the <RESTORE> key. Do it a couple of times if nothing happens at first.

```
5 REM            TYPER
30 :
100 GOSUB 1000
498 :
499 REM         MAIN LOOP
500 GET A$
600 IF A$="" THEN 500
610 PRINT A$; :            REM PUT ON PAPER
640 GOTO 500  :            REM KEEP IT UP
660 :
990 REM         INITIALIZATION ROUTINE
995 REM         CLEAR SCREEN AND SET TO WHITE
1000 PRINT CHR$(147) :    REM CLEAR SCREEN
1030 REM        THIS SETS BORDER TO WHITE
1040 POKE 53280,1
1050 REM        THIS SETS BACKGROUND TO WHITE
1060 POKE 53281,1
1070 PRINT CHR$(144) :    REM  BLACK TYPE
1099 RETURN
```

Type With Cursor

We add a flashing cursor to TYPER with lines 520 to 590. Note that the flashing cursor is done with software. You print a space and then print a reverse space (or the character and then the reverse character) at the same place, giving the illusion of flashing. Lines 520 to 590 do that, except instead of printing, we use PEEKs and POKEs to the screen memory. Note that you can only type one page of text, so when you reach the last line on the screen the next character will be put on the line above. Line 605 does this. If you want the screen to scroll, delete line 605, but then you will also lose the flashing cursor. You can't have your cake and eat it, too.

```
5 REM         TYPE WITH CURSOR
6 REM  THIS IS TYPER  WITH
7 REM  LINES 520-590  AND
8 REM  602-606 ADDED
30 :
100 GOSUB 1000
498 :
499 REM         MAIN LOOP
500 GET A$
520 REM         GET CURSOR LOCATION
530 CL = 256*PEEK(210) + PEEK(209) + PEEK(211)
540                        REM  FLASH CURSOR
550 POKE CL,(PEEK(CL) + 128) AND 255
560                        REM  DELAY LOOP
570 FOR I = 1 TO 5 : NEXT
580                        REM  FLASH CURSOR
590 POKE CL,(PEEK(CL) + 128) AND 255
```

```
600 IF A$="" THEN 500
602 :
603 REM          BACK UP IF LAST LINE
604 REM          CHR$(145) IS CURSOR UP
605 IF CL<1983 THEN A$=CHR$(145)
606 :
610 PRINT A$;:              REM PUT ON PAPER
640 GOTO 500 :              REM KEEP IT UP
650 :
660 :
990 REM          INITIALIZATION ROUTINE
995 REM          CLEAR SCREEN AND SET TO WHITE
1000 PRINT CHR$(147) :    REM CLEAR SCREEN
1030 REM         THIS SETS BORDER TO WHITE
1040 POKE 53280,1
1050 REM         THIS SETS BACKGROUND TO WHITE
1060 POKE 53281,1
1070 PRINT CHR$(144) :    REM  BLACK TYPE
1099 RETURN
```

Sound

This addition adds a key-click to your typewriter. If you want to change the sound, consult Chapter 8, "Bells and Whistles."

```
5 REM          SOUND
6 REM  THIS IS TYPE WITH CURSOR  WITH
7 REM  LINES 299-360, 620 AND
8 REM  4999-5020 ADDED
30 :
100 GOSUB 1000
298 :
299 REM          SET UP SOUND FOR CLICK
300 S=54272 :               REM  SID DATA AREA
310 POKE S+5,0 :            REM  ATTACK/DECAY
320 POKE S+6,0 :            REM  SUSTAIN/RELEASE
330 POKE S+24,15 :          REM VOLUME
340 POKE S+1, 67:           REM NOTE PART 1
350 POKE S,15 :             REM NOTE PART 2
360 POKE S+4,17 :           REM WAVEFORM
498 :
499 REM          MAIN LOOP
500 GET A$
520 REM          GET CURSOR LOCATION
530 CL = 256*PEEK(210) + PEEK(209) + PEEK(211)
540                         REM  FLASH CURSOR
550 POKE CL,(PEEK(CL) + 128) AND 255
560                         REM  DELAY LOOP
570 FOR I = 1 TO 5 : NEXT
580                         REM  FLASH CURSOR
```

```
590 POKE CL,(PEEK(CL) + 128) AND 255
600 IF A$="" THEN 500
602 :
603 REM         BACK UP IF LAST LINE
604 REM         CHR$(145) IS CURSOR UP
605 IF CL<1983 THEN A$=CHR$(145)
606 :
610 PRINT A$;:              REM PUT ON PAPER
620 GOSUB 5000:             REM  CLICK
640 GOTO 500 :              REM KEEP IT UP
650 :
660 :
990 REM         INITIALIZATION ROUTINE
995 REM         CLEAR SCREEN AND SET TO WHITE
1000 PRINT CHR$(147) :    REM CLEAR SCREEN
1030 REM        THIS SETS BORDER TO WHITE
1040 POKE 53280,1
1050 REM        THIS SETS BACKGROUND TO WHITE
1060 POKE 53281,1
1070 PRINT CHR$(144) :    REM  BLACK TYPE
1099 RETURN
4999 REM        CLICK SUBROUTINE
5000 POKE S+4,17 :        REM  ON
5010 POKE S+4,16 :        REM  OFF
5020 RETURN
```

Lower Case

This program allows you to input and display both upper and lower case, just like an old-fashioned typewriter.

```
5 REM         LOWER CASE
6 REM  THIS IS SOUND  WITH
7 REM  LINES 1010-1020 ADDED
30 :
100 GOSUB 1000
298 :
299 REM         SET UP SOUND FOR CLICK
300 S=54272 :             REM  SID DATA AREA
310 POKE S+5,0 :          REM  ATTACK/DECAY
320 POKE S+6,0 :          REM  SUSTAIN/RELEASE
330 POKE S+24,15 :        REM VOLUME
340 POKE S+1, 67:         REM NOTE PART 1
350 POKE S,15 :           REM NOTE PART 2
360 POKE S+4,17 :         REM WAVEFORM
498 :
499 REM         MAIN LOOP
500 GET A$
520 REM         GET CURSOR LOCATION
530 CL = 256*PEEK(210) + PEEK(209) + PEEK(211)
```

```
540                          REM  FLASH CURSOR
550 POKE CL,(PEEK(CL) + 128) AND 255
560                          REM  DELAY LOOP
570 FOR I = 1 TO 5 : NEXT
580                          REM  FLASH CURSOR
590 POKE CL,(PEEK(CL) + 128) AND 255
600 IF A$="" THEN 500
602 :
603 REM          BACK UP IF LAST LINE
604 REM          CHR$(145) IS CURSOR UP
605 IF CL<1983 THEN A$=CHR$(145)
606 :
610 PRINT A$;:               REM PUT ON PAPER
620 GOSUB 5000:              REM  CLICK
640 GOTO 500 :               REM KEEP IT UP
650 :
660 :
990 REM          INITIALIZATION ROUTINE
995 REM          CLEAR SCREEN AND SET TO WHITE
1000 PRINT CHR$(147) :    REM CLEAR SCREEN
1010 REM         ENABLES NORMAL KEYBOARD
1020 PRINT CHR$(14)
1030 REM         THIS SETS BORDER TO WHITE
1040 POKE 53280,1
1050 REM         THIS SETS BACKGROUND TO WHITE
1060 POKE 53281,1
1070 PRINT CHR$(144) :    REM  BLACK TYPE
1099 RETURN
4999 REM         CLICK SUBROUTINE
5000 POKE S+4,17 :           REM  ON
5010 POKE S+4,16 :           REM  OFF
5020 RETURN
```

Add Color

This feature allows each character to be displayed in a new color. Do not expect these colors to display on your printer. If it does print in color, give us a call.

```
5 REM          ADD COLOR
6 REM   THIS IS LOWER CASE WITH
7 REM   LINES 40-71 ADDED
8 REM   AS WELL AS LINES 199-230
9 REM   AND LINE 630
30 :
40 REM           STANDARD RANDOM  1 UP TO X
50 DEF FND(X) = INT(X*RND(1)+1)
51 :
60 REM           GETS NEXT RANDOM COLOR
70 DEF FNC(Y) = A(FND(7))
```

```
71 :
100 GOSUB 1000
199 REM         READ COLOR VALUES
200 FOR I = 1 TO 7
210 READ A(I)
220 NEXT
230 DATA 28,30,31,144,156,158,159
298 :
299 REM         SET UP SOUND FOR CLICK
300 S=54272 :               REM  SID DATA AREA
310 POKE S+5,0 :            REM  ATTACK/DECAY
320 POKE S+6,0 :            REM  SUSTAIN/RELEASE
330 POKE S+24,15 :          REM VOLUME
340 POKE S+1, 67:           REM NOTE PART 1
350 POKE S,15 :             REM NOTE PART 2
360 POKE S+4,17 :           REM WAVEFORM
498 :
499 REM         MAIN LOOP
500 GET A$
520 REM         GET CURSOR LOCATION
530 CL = 256*PEEK(210) + PEEK(209) + PEEK(211)
540                         REM  FLASH CURSOR
550 POKE CL,(PEEK(CL) + 128) AND 255
560                         REM  DELAY LOOP
570 FOR I = 1 TO 5 : NEXT
580                         REM  FLASH CURSOR
590 POKE CL,(PEEK(CL) + 128) AND 255
600 IF A$="" THEN 500
602 :
603 REM         BACK UP IF LAST LINE
604 REM         CHR$(145) IS CURSOR UP
605 IF CL<1983 THEN A$=CHR$(145)
606 :
610 PRINT A$;:              REM PUT ON PAPER
620 GOSUB 5000:             REM  CLICK
630 PRINT CHR$(FNC(Y));:    REM  NEXT COLOR
640 GOTO 500 :              REM KEEP IT UP
650 :
660 :
990 REM         INITIALIZATION ROUTINE
995 REM         CLEAR SCREEN AND SET TO WHITE
1000 PRINT CHR$(147) :    REM CLEAR SCREEN
1010 REM        ENABLES NORMAL KEYBOARD
1020 PRINT CHR$(14)
1030 REM        THIS SETS BORDER TO WHITE
1040 POKE 53280,1
1050 REM        THIS SETS BACKGROUND TO WHITE
1060 POKE 53281,1
1070 PRINT CHR$(144) :    REM  BLACK TYPE
1099 RETURN
4999 REM        CLICK SUBROUTINE
```

```
5000 POKE S+4,17 :          REM  ON
5010 POKE S+4,16 :          REM  OFF
5020 RETURN
```

Printing Typer

This is our final version of the typewriter, which has the capability to not only see what you get but also to print what you get (if you have a printer, of course). Use the function key F1 to print whatever is on the screen.

```
5 REM         PRINTING TYPER
6 REM    THIS IS ADD COLOR WITH
7 REM    LINE 510 AND FROM 29997 ON ADDED
30 :
40 REM          STANDARD RANDOM  1 UP TO X
50 DEF FND(X) = INT(X*RND(1)+1)
51 :
60 REM          GETS NEXT RANDOM COLOR
70 DEF FNC(Y) = A(FND(7))
71 :
100 GOSUB 1000
199 REM         READ COLOR VALUES
200 FOR I = 1 TO 7
210 READ A(I)
220 NEXT
230 DATA 28,30,31,144,156,158,159
298 :
299 REM         SET UP SOUND FOR CLICK
300 S=54272 :               REM  SID DATA AREA
310 POKE S+5,0 :            REM  ATTACK/DECAY
320 POKE S+6,0 :            REM  SUSTAIN/RELEASE
330 POKE S+24,15 :          REM VOLUME
340 POKE S+1, 67:           REM NOTE PART 1
350 POKE S,15 :             REM NOTE PART 2
360 POKE S+4,17 :           REM WAVEFORM
498 :
499 REM         MAIN LOOP
500 GET A$
510 GOSUB 30011 :           REM CHECK F1
520 REM         GET CURSOR LOCATION
530 CL = 256*PEEK(210) + PEEK(209) + PEEK(211)
540                         REM  FLASH CURSOR
550 POKE CL,(PEEK(CL) + 128) AND 255
560                         REM  DELAY LOOP
570 FOR I = 1 TO 5 : NEXT
580                         REM  FLASH CURSOR
590 POKE CL,(PEEK(CL) + 128) AND 255
600 IF A$="" THEN 500
602 :
603 REM          BACK UP IF LAST LINE
```

```
604 REM         CHR$(145) IS CURSOR UP
605 IF CL<1983 THEN A$=CHR$(145)
606 :
610 PRINT A$;:              REM PUT ON PAPER
620 GOSUB 5000:             REM  CLICK
630 PRINT CHR$(FNC(Y));:    REM  NEXT COLOR
640 GOTO 500 :              REM KEEP IT UP
650 :
660 :
990 REM         INITIALIZATION ROUTINE
995 REM         CLEAR SCREEN AND SET TO WHITE
1000 PRINT CHR$(147) :    REM CLEAR SCREEN
1010 REM        ENABLES NORMAL KEYBOARD
1020 PRINT CHR$(14)
1030 REM        THIS SETS BORDER TO WHITE
1040 POKE 53280,1
1050 REM        THIS SETS BACKGROUND TO WHITE
1060 POKE 53281,1
1070 PRINT CHR$(144) :    REM  BLACK TYPE
1099 RETURN
4999 REM        CLICK SUBROUTINE
5000 POKE S+4,17 :          REM  ON
5010 POKE S+4,16 :          REM  OFF
5020 RETURN
29997 :
29998 :
29999 REM       WAIT FOR KEYPRESS
30010 REM       STOP IF F1 IS NOT PRESSED
30011 IF A$<>CHR$(133) THEN RETURN
30020 OPEN 4,4,4 :          REM  ENABLE PRINTER
30030 CMD4 :                REM PRINT IT
30040 CG = PEEK(53272)
30050 SR = 1024 :           REM START OF SCREEN
30060 REM       FOR WHOLE 1000 CHARACTERS
30061 REM       ON THE SCREEN
30062 FOR I0 = 0 TO 999
30070 REM       GET SCREEN CONTENTS
30080 CZ = PEEK(SR + I0)
30090 IF CG=21 THEN GOSUB 30180
30100 IF CG=23 THEN GOSUB 30140
30110 REM       PRINT THE CHARACTER
30111 PRINT#4,CHR$(AZ);: LL = LL + 1
30120 IF LL=40 THEN PRINT#4,CHR$(13):LL = 0
30130 NEXT I0 : CLOSE4 : RETURN
30140 IF CZ<27 THEN AZ = CZ + 96 : RETURN
30150 IF CZ<32 THEN AZ = CZ + 64 : RETURN
30160 IF CZ<91 THEN AZ = CZ : RETURN
30170 AZ = 32 : RETURN
30180 IF CZ<32 THEN AZ = CZ + 64 : RETURN
30190 IF CZ<64 THEN AZ = CZ : RETURN
30200 A  = 32 : RETURN
```

Click

This is the sound routine in PRINTING TYPER. See Chapter 8, "Bells and Whistles," for hints in changing the sound routine.

```
5 REM CLICK
298 :
299 REM          SET UP SOUND FOR CLICK
300 S=54272 :         REM  SID DATA AREA
310 POKE S+5,0 :      REM  ATTACK/DECAY
320 POKE S+6,0 :      REM  SUSTAIN/RELEASE
330 POKE S+24,15 :    REM VOLUME
340 POKE S+1, 67:     REM NOTE PART 1
350 POKE S,15 :       REM NOTE PART 2
360 POKE S+4,17 :     REM WAVEFORM
498 :
500 GOSUB 5000
510 END
4998 :
4999 REM         CLICK SUBROUTINE
5000 POKE S+4,17 :          REM  ON
5010 POKE S+4,16 :          REM  OFF
5020 RETURN
```

Screen Print

This is a program (which can be used as a subroutine) to print your display screen. Since it is written to be used as a subroutine (lines 29999–30200), when you RUN it, you must press function key F1 to get a printout of your screen. Any other key will simply END the program.

For some non-Commodore printers, this program will not work. Those printers may need to change line number 30020 to make their printer work. Consult your interface manual or dealer for possible assistance.

```
5 REM           SCREEN PRINT
6 REM  PRINTS WHEN F1 IS PRESSED
7 REM  TO FORCE A PRINT WITHOUT A
8 REM  KEYPRESS THEN GOSUB 30020
100 GOSUB 30000
110 END
29997 :
29998 :
29999 REM       WAIT FOR KEYPRESS
30000 GET A$
30005 IF A$="" THEN 30000
30010 REM       STOP IF F1 IS NOT PRESSED
30011 IF A$<>CHR$(133) THEN RETURN
30020 OPEN 4,4,4 :          REM  ENABLE PRINTER
30030 CMD4 :                REM PRINT IT
```

```
30040 CG = PEEK(53272)
30050 SR = 1024 :          REM START OF SCREEN
30060 REM        FOR WHOLE 1000 CHARACTERS
30061 REM        ON THE SCREEN
30062 FOR I1 = 0 TO 999
30070 REM        GET SCREEN CONTENTS
30080 CZ = PEEK(SR + I1)
30090 IF CG=21 THEN GOSUB 30180
30100 IF CG=23 THEN GOSUB 30140
30110 REM        PRINT THE CHARACTER
30111 PRINT#4,CHR$(AZ);: LL = LL + 1
30120 IF LL=40 THEN PRINT#4,CHR$(13):LL = 0
30130 NEXT I1 : CLOSE4 :                RETURN
30140 IF CZ<27 THEN AZ = CZ + 96 : RETURN
30150 IF CZ<32 THEN AZ = CZ + 64 : RETURN
30160 IF CZ<91 THEN AZ = CZ :       RETURN
30170 AZ = 32 :                     RETURN
30180 IF CZ<32 THEN AZ = CZ + 64 : RETURN
30190 IF CZ<64 THEN AZ = CZ :       RETURN
30200 A  = 32 : RETURN
```

Challenges

1. Find a way to save the text to disk or tape. (Hint: if you can get it to the printer, it is not much more trouble to get it to a disk file.)
2. Better yet, can you get it back after you saved it?
3. Add a cursor routine to other programs that you or others have written.
4. Your TYPER may have done "bizarre" things when you used the full screen editor keys. Can you solve these problems? (Try "filtering" the input so those keys don't cause anything to happen.)
5. Make a bell ring near the end of each line, like a typewriter. But be careful—if you use the cursor movement keys, you might get a ring in the middle of the page.
6. Find a way to create an automatic typewriter that can write a nonsense letter.
7. Modify TYPE WITH CURSOR to scroll the text. In other words, allow the screen to scroll when there is too much for one screen, but keep the cursor blinking.

2

OUR APOLOGIES TO ETCH-A-SKETCH™

Programs:

MOVE CURSOR	This program controls the movement of the cursor on the screen.
ETCH	This is a complete Etch-a-Sketch™.
ETCH CURSOR	A flashing cursor is added to ETCH.
ETCH PENUP	This added feature allows you to choose between a pen and an eraser.
CHANGE COLOR	This routine allows you to change the color of the pen you are drawing with.
CHANGE BACKGROUND	This routine changes the color of the paper you are drawing on.
PRINTING ETCHER	This final version allows you to print your drawing if you have a printer.
READ JOYSTICK	This simple routine shows you what the joystick is saying.
KEYSCAN	This simple routine shows you what the keyboard is saying.

This collection of programs will allow you to create a simple "Etch-a-Sketch" and add a series of improvements. The final version of this program will do the following:

1. Move the cursor to wherever you want it on the screen
2. Draw your sketches on the screen

3. Flash to show where your pen is located
4. Allow you to draw with any character or symbol on the keyboard
5. Let you change the color of your pen while you are drawing
6. Let you erase parts of the drawing
7. Let you change the color of the paper you are drawing on
8. Allow you to print a copy of your drawing.

Drawing with a Commodore

The idea in this chapter is to use the joystick as the basic input device, resorting to the keyboard only for unusual inputs. In order to do this, we build a program, MOVE CURSOR, that moves the cursor around to any desired location on the screen, locating it by the row and column we designate (X,Y coordinates). This program uses a new technique: a machine language program. We POKE a set of numbers directly into memory to make a machine language program and run that program with a SYS statement. Machine language is how computer nuts refer to the only language that the little computer chip inside your 64 really understands. All other computer languages, such as BASIC, have to be translated (interpreted or compiled) into machine language. While almost anything that your 64 can do can be programmed in BASIC, sometimes it's easier to do it in machine language, for example, moving the cursor around in MOVE CURSOR; or it can be done much faster, as the JOYSTICK SPRITE programs in Chapter 7, "Video Arcade," will show you. Programs can easily be 100 times faster when programmed in machine language than when run in BASIC. It was both easier and faster to use a machine language program in this case, so we did it.

BASIC Commands Used in This Chapter

AND	NEXT
CHR$	OPEN
CLOSE	PEEK
DATA	POKE
END	PRINT
FOR	READ
GET	REM
GOSUB	RETURN
GOTO	SYS
IF	

Programming Techniques Used in This Chapter

1. *Using a machine language program from BASIC.* Relocate the cursor to any X,Y location. See MOVE CURSOR lines 1040–1090 for the setup and lines 5000–5030 for the use.
2. *Read the joystick setting.* See READ JOYSTICK. To use the readings see ETCH lines 1530–1870.
3. *Check for cursor locations going off the screen.* See ETCH lines 2040–2070.
4. *Read paddle settings.* See READ PADDLES.
5. *Get a character from the keyboard and see what it is.* See KEYSCAN. Also see ETCH line 2640. Note that other programs in this book use a GET A$ statement to get characters from the keyboard, while here we use a PEEK(197). The main difference is simply taste; either could be used.
6. *Read in DATA from a DATA line.* See MOVE CURSOR lines 1060–1090 and 9999.
7. *Make a flashing cursor.* Print the character, then a blank, then the character, then a blank, and so on. See lines 2440–2500 in ETCH CURSOR.
8. *Read joystick button.* See if the fire button on the joystick has been pressed or not. Make it operate like a push-button on/off switch. See ETCH PENUP lines 1530–1540, where we keep track of whether to print a character or not with the variable PD.
9. *Printing subroutine to dump the screen.* See PRINTING ETCHER lines 30020–30200. Note that we PEEK at the screen to see what character is there and then decide how to print, depending upon whether we are in upper/lower case or upper case/graphics.
10. *Using the function keys.* See CHANGE BACKGROUND lines 2640, 3030 and PRINTING ETCHER lines 2540, 3030–3050.

Move Cursor

This program moves a cursor (a character on the screen) to any position on the screen. You can change lines 1320 and 1330 to move the cursor to a different location. Line 1310 determines the cursor character. If you want, instead of naming the cursor by the number, you could type the character as in the following:

```
1310 CH=ASC("X")
```

where the X could be any character that you wish.

```
5 REM            MOVE CURSOR
1000 :
1010 REM         LINES 1020-1090 POKE A MACHINE
```

```
1020 REM        LANGUAGE ROUTINE THAT IS USED
1030 REM        IN LINE 5020
1040 S = 12*4096 :          REM  STARTS AT $C000
1050 PRINT CHR$(147)
1060 FOR I = 0 TO 7
1070 READ A
1080 POKE S + I,A
1090 NEXT
1300 :
1310 CH = 35:               REM THIS IS THE CURSOR
1320 Y = 11 :               REM  THE Y POSITION
1330 X = 19 :               REM  THE X POSITION
1500 :
1510 GOSUB 5000 :           REM MOVE CURSOR TO X,Y
1520 PRINT CHR$(CH)
1530 END
4998 :
4999 REM        MOVE CURSOR TO X,Y
5000 POKE S + 3,X
5010 POKE S + 1,Y
5020 SYS S  :               REM CALL MACHINE LANGUAGE
5030 RETURN
9997 :
9998 REM        DATA IS FOR READ IN LINE 1070
9999 DATA 162,20,160,15,24,76,240,255
```

Etch

This is a complete Etch-a-Sketch™. The cursor is moved on the screen by the joystick leaving its trail behind just as in the "Etch-a-Sketch™." *Be sure your joystick is connected to port #2.*

```
5 REM           ETCH
6 REM  THIS IS MOVE CURSOR  WITH
7 REM  LINES 1810-3140  ADDED
8 REM  AND LINES 1520-1530 CHANGED
9 :
10 REM  USE JOYSTICK IN PORT 2
15 :
1000 :
1010 REM        LINES 1020-1090 POKE A MACHINE
1020 REM        LANGUAGE ROUTINE THAT IS USED
1030 REM        IN LINE 5020
1040 S = 12*4096 :          REM  STARTS AT $C000
1050 PRINT CHR$(147)
1060 FOR I = 0 TO 7
1070 READ A
1080 POKE S + I,A
1090 NEXT
1300 :
1310 CH = 35:               REM THIS IS THE CURSOR
```

```
1320 Y = 11 :                REM THE Y POSITION
1330 X = 19 :                REM THE X POSITION
1500 :
1510 GOSUB 5000 :            REM MOVE CURSOR TO X,Y
1520 REM        READ JOYSTICK
1530 A = PEEK(56320) AND 31
1810 REM        THE LINES 1830-1870 CHANGE
1820 REM        X,Y IN ACCORD WITH JOYSTICK
1830 A = A AND 15
1840 IF A AND 8 THEN X=X-1
1850 IF A AND 4 THEN X=X+1
1860 IF A AND 2 THEN Y=Y-1
1870 IF A AND 1 THEN Y=Y+1
2000 :
2010 REM        LINES 2040-2070 CORRECT FOR
2020 REM        X,Y POSITIONS THAT WOULD BE
2030 REM        OFFSCREEN
2040 IF X<0 THEN X=0
2050 IF Y<0 THEN Y=0
2060 IF Y>23 THEN Y=23
2070 IF X>39 THEN X=39
2400 :
2600 :
2630 PRINT CHR$(CH);
2640 A = PEEK(197) :         REM READ KEYBOARD
3140 GOTO 1510
4998 :
4999 REM        MOVE CURSOR TO X,Y
5000 POKE S + 3,X
5010 POKE S + 1,Y
5020 SYS S  :                REM CALL MACHINE LANGUAGE
5030 RETURN
9997 :
9998 REM        DATA IS FOR READ IN LINE 1070
9999 DATA 162,20,160,15,24,76,240,255
```

Etch Cursor

A flashing cursor is added to our Etch-a-Sketch™ ETCH program. Again, be sure that the joystick is in port #2.

```
5 REM         ETCH CURSOR
6 REM   THIS IS ETCH  WITH
7 REM   LINES 2410-2500  ADDED
8 REM   AND LINES 3140 CHANGED
15 :
1000 :
1010 REM        LINES 1020-1090 POKE A MACHINE
1020 REM        LANGUAGE ROUTINE THAT IS USED
1030 REM        IN LINE 5020
```

```
1040 S = 12*4096 :          REM  STARTS AT $C000
1050 PRINT CHR$(147)
1060 FOR I = 0 TO 7
1070 READ A
1080 POKE S + I,A
1090 NEXT
1300 :
1310 CH = 35:               REM THIS IS THE CURSOR
1320 Y = 11 :               REM  THE Y POSITION
1330 X = 19 :               REM  THE X POSITION
1500 :
1510 GOSUB 5000 :           REM MOVE CURSOR TO X,Y
1520 REM        READ JOYSTICK
1530 A = PEEK(56320) AND 31
1810 REM        THE LINES 1830-1870 CHANGE
1820 REM        X,Y IN ACCORD WITH JOYSTICK
1830 A = A AND 15
1840 IF A AND 8 THEN X=X-1
1850 IF A AND 4 THEN X=X+1
1860 IF A AND 2 THEN Y=Y-1
1870 IF A AND 1 THEN Y=Y+1
2000 :
2010 REM        LINES 2040-2070 CORRECT FOR
2020 REM        X,Y POSITIONS THAT WOULD BE
2030 REM        OFFSCREEN
2040 IF X<0 THEN X=0
2050 IF Y<0 THEN Y=0
2060 IF Y>23 THEN Y=23
2070 IF X>39 THEN X=39
2400 :
2410 :
2420 REM        THE LINES 2440-2500 BLINK THE
2430 REM        CURSOR AT POSITION X,Y
2440 GOSUB 5000:            REM  PUT CURSOR AT X,Y
2450 PRINT CHR$(32);
2460 GOSUB 5000
2470 PRINT CHR$(CH);
2480 GOSUB 5000
2490 PRINT CHR$(32);
2500 GOSUB 5000
2600 :
2630 PRINT CHR$(CH);
2640 A = PEEK(197) :        REM READ KEYBOARD
3140 GOTO 1530
4998 :
4999 REM        MOVE CURSOR TO X,Y
5000 POKE S + 3,X
5010 POKE S + 1,Y
5020 SYS S  :               REM CALL MACHINE LANGUAGE
5030 RETURN
9997 :
9998 REM        DATA IS FOR READ IN LINE 1070
9999 DATA 162,20,160,15,24,76,240,255
```

Etch Penup

We add a feature that allows you to choose between a pen and an eraser. When the program begins, the cursor acts like a pen drawing on a piece of paper. Pressing the "fire" button on your joystick changes the cursor into an "eraser." Try changing lines 2450 and 2490 to obtain cursors other than a blank and crosshatch. Again, be sure that the joystick is in port #2.

```
5 REM       ETCH PENUP
6 REM     THIS IS  ETCH CURSOR   WITH
7 REM     LINES 22-90, 1540-1800,
8 REM     AND 2610-2620  ADDED
22 :
90 PD = 1 :                 REM PD IS PENDOWN FLAG
1000 :
1010 REM         LINES 1020-1090 POKE A MACHINE
1020 REM         LANGUAGE ROUTINE THAT IS USED
1030 REM         IN LINE 5020
1040 S = 12*4096 :          REM  STARTS AT $C000
1050 PRINT CHR$(147)
1060 FOR I = 0 TO 7
1070 READ A
1080 POKE S + I,A
1090 NEXT
1300 :
1310 CH = 35:               REM THIS IS THE CURSOR
1320 Y = 11 :               REM THE Y POSITION
1330 X = 19 :               REM THE X POSITION
1500 :
1510 GOSUB 5000 :           REM MOVE CURSOR TO X,Y
1520 REM        READ JOYSTICK
1530 A = PEEK(56320) AND 31
1540 IF (A AND 16)=0 THEN PD = 1 - PD
1550 REM        PD IS PENDOWN FLAG
1560 REM        PD = 1-PD TOGGLES IT
1570 REM        A TOGGLE CHANGES THE STATE
1580 REM        E.G., FROM PENUP TO PENDOWN
1800 :
1810 REM        THE LINES 1830-1870 CHANGE
1820 REM        X,Y IN ACCORD WITH JOYSTICK
1830 A = A AND 15
1840 IF A AND 8 THEN X=X-1
1850 IF A AND 4 THEN X=X+1
1860 IF A AND 2 THEN Y=Y-1
1870 IF A AND 1 THEN Y=Y+1
2000 :
2010 REM        LINES 2040-2070 CORRECT FOR
2020 REM        X,Y POSITIONS THAT WOULD BE
2030 REM        OFFSCREEN
2040 IF X<0 THEN X=0
2050 IF Y<0 THEN Y=0
```

```
2060 IF Y>23 THEN Y=23
2070 IF X>39 THEN X=39
2400 :
2410 :
2420 REM        THE LINES 2440-2500 BLINK THE
2430 REM        CURSOR AT POSITION X,Y
2440 GOSUB 5000:            REM  PUT CURSOR AT X,Y
2450 PRINT CHR$(32);
2460 GOSUB 5000
2470 PRINT CHR$(CH);
2480 GOSUB 5000
2490 PRINT CHR$(32);
2500 GOSUB 5000
2600 :
2610 REM        IF PEN IS UP (PD=0) DON'T PRINT
2620 IF PD = 0 THEN 2640
2630 PRINT CHR$(CH);
2640 A = PEEK(197) :        REM READ KEYBOARD
3140 GOTO 1530
4998 :
4999 REM        MOVE CURSOR TO X,Y
5000 POKE S + 3,X
5010 POKE S + 1,Y
5020 SYS S  :               REM CALL MACHINE LANGUAGE
5030 RETURN
9997 :
9998 REM        DATA IS FOR READ IN LINE 1070
9999 DATA 162,20,160,15,24,76,240,255
```

Change Color

This routine allows you to change the color of the drawing pen. Hit any key number, 1 through 8, to change the color to that shown on the front of those keys. Hitting the space bar erases all of your picture and starts you in the middle of the screen again. Again, be sure that the joystick is in port #2.

```
5 REM           CHANGE COLOR
6 REM  THIS IS ETCH PENUP   WITH
7 REM  LINES 2800-2860 AND
8 REM  5996-6080  ADDED
9 :
12 REM  HIT A COLOR KEY
13 REM  (NUMBER KEYS 1 TO 8)
14 REM  TO CHANGE THE DRAWING COLOR
15 :
90 PD = 1 :                 REM PD IS PENDOWN FLAG
1000 :
1010 REM        LINES 1020-1090 POKE A MACHINE
1020 REM        LANGUAGE ROUTINE THAT IS USED
1030 REM        IN LINE 5020
```

```
1040 S = 12*4096 :          REM  STARTS AT $C000
1050 PRINT CHR$(147)
1060 FOR I = 0 TO 7
1070 READ A
1080 POKE S + I,A
1090 NEXT
1300 :
1310 CH = 35:               REM  THIS IS THE CURSOR
1320 Y = 11 :               REM  THIS IS THE Y POSITION
1330 X = 19 :               REM  THIS IS THE X POSITION
1500 :
1510 GOSUB 5000 :           REM MOVE CURSOR TO X,Y
1520 REM        READ JOYSTICK
1530 A = PEEK(56320) AND 31
1540 IF (A AND 16)=0 THEN PD = 1 - PD
1550 REM        PD IS PENDOWN FLAG
1560 REM        PD = 1-PD TOGGLES IT
1570 REM        A TOGGLE CHANGES THE STATE
1580 REM        E.G., FROM PENUP TO PENDOWN
1800 :
1810 REM        THE LINES 1830-1870 CHANGE
1820 REM        X,Y IN ACCORD WITH JOYSTICK
1830 A = A AND 15
1840 IF A AND 8 THEN X=X-1
1850 IF A AND 4 THEN X=X+1
1860 IF A AND 2 THEN Y=Y-1
1870 IF A AND 1 THEN Y=Y+1
2000 :
2010 REM        LINES 2040-2070 CORRECT FOR
2020 REM        X,Y POSITIONS THAT WOULD BE
2030 REM        OFFSCREEN
2040 IF X<0 THEN X=0
2050 IF Y<0 THEN Y=0
2060 IF Y>23 THEN Y=23
2070 IF X>39 THEN X=39
2400 :
2410 :
2420 REM        THE LINES 2440-2500 BLINK THE
2430 REM        CURSOR AT POSITION X,Y
2440 GOSUB 5000:            REM       PUT CURSOR AT X,Y
2450 PRINT CHR$(32);
2460 GOSUB 5000
2470 PRINT CHR$(CH);
2480 GOSUB 5000
2490 PRINT CHR$(32);
2500 GOSUB 5000
2600 :
2610 REM        IF PEN IS UP (PD=0) DON'T PRINT
2620 IF PD = 0 THEN 2640
2630 PRINT CHR$(CH);
2640 A = PEEK(197) :        REM READ KEYBOARD
```

```
2800 :
2810 REM        IF SPACE CLEAR SCREEN, RESTART
2820 IF A=60 THEN PRINT CHR$(147): GOTO 1310
2830 GOSUB 6000:            REM  GET NEW COLOR
2840 CC = CO    :           REM  CC IS DISPLAY COLOR
2850 REM        POKE NEW COLOR INTO COLOR MAP
2860 POKE 55296 + 40*Y + X,CC
3140 GOTO 1530
4998 :
4999 REM        MOVE CURSOR TO X,Y
5000 POKE S + 3,X
5010 POKE S + 1,Y
5020 SYS S  :               REM CALL MACHINE LANGUAGE
5030 RETURN
5996 :
5997 REM        TRANSLATE NUMBER KEYS INTO
5998 REM        COLORS
5999 REM                                KEY
6000 IF A=56 THEN CO = 0: REM  1  BLACK
6010 IF A=59 THEN CO = 1: REM  2  WHITE
6020 IF A= 8 THEN CO = 2: REM  3  RED
6030 IF A=11 THEN CO = 3: REM  4  CYAN
6040 IF A=16 THEN CO = 4: REM  5  PURPLE
6050 IF A=19 THEN CO = 5: REM  6  GREEN
6060 IF A=24 THEN CO = 6: REM  7  BLUE
6070 IF A=27 THEN CO = 7: REM  8  YELLOW
6080 RETURN
9997 :
9998 REM        DATA IS FOR READ IN LINE 1070
9999 DATA 162,20,160,15,24,76,240,255
```

Change Background

This routine changes the color of the paper you are drawing on. Press the function key F1 until the cursor stops flashing. Then hit a number key from 1 through 8 to use the color displayed on the front of the key. Again, be sure that the joystick is in port #2.

```
5 REM          CHANGE BACKGR
6 REM  THIS IS CHANGE COLOR  WITH
7 REM  LINES 3000-3130 ADDED
9 :
10 REM  HIT F1 UNTIL CURSOR STOPS AND
11 REM  THEN A COLOR KEY (NUMBERS 1 TO 8)
12 REM  TO CHANGE BACKGROUND COLOR
15 :
90 PD = 1 :                 REM PD IS PENDOWN FLAG
1000 :
1010 REM        LINES 1020-1090 POKE A MACHINE
```

```
1020 REM        LANGUAGE ROUTINE THAT IS USED
1030 REM        IN LINE 5020
1040 S = 12*4096 :          REM  STARTS AT $C000
1050 PRINT CHR$(147)
1060 FOR I = 0 TO 7
1070 READ A
1080 POKE S + I,A
1090 NEXT
1300 :
1310 CH = 35:               REM THIS IS THE CURSOR
1320 Y = 11 :               REM  THE Y POSITION
1330 X = 19 :               REM  THE X POSITION
1500 :
1510 GOSUB 5000 :           REM MOVE CURSOR TO X,Y
1520 REM        READ JOYSTICK
1530 A = PEEK(56320) AND 31
1540 IF (A AND 16)=0 THEN PD = 1 - PD
1550 REM        PD IS PENDOWN FLAG
1560 REM        PD = 1-PD TOGGLES IT
1570 REM        A TOGGLE CHANGES THE STATE
1580 REM        E.G., FROM PENUP TO PENDOWN
1800 :
1810 REM        THE LINES 1830-1870 CHANGE
1820 REM        X,Y IN ACCORD WITH JOYSTICK
1830 A = A AND 15
1840 IF A AND 8 THEN X=X-1
1850 IF A AND 4 THEN X=X+1
1860 IF A AND 2 THEN Y=Y-1
1870 IF A AND 1 THEN Y=Y+1
2000 :
2010 REM        LINES 2040-2070 CORRECT FOR
2020 REM        X,Y POSITIONS THAT WOULD BE
2030 REM        OFFSCREEN
2040 IF X<0 THEN X=0
2050 IF Y<0 THEN Y=0
2060 IF Y>23 THEN Y=23
2070 IF X>39 THEN X=39
2400 :
2410 :
2420 REM        THE LINES 2440-2500 BLINK THE
2430 REM        CURSOR AT POSITION X,Y
2440 GOSUB 5000:            REM  PUT CURSOR AT X,Y
2450 PRINT CHR$(32);
2460 GOSUB 5000
2470 PRINT CHR$(CH);
2480 GOSUB 5000
2490 PRINT CHR$(32);
2500 GOSUB 5000
2600 :
2610 REM        IF PEN IS UP(PD=0)DON'T PRINT
2620 IF PD = 0 THEN 2640
```

```
2630 PRINT CHR$(CH);
2640 A = PEEK(197) :       REM READ KEYBOARD
2800 :
2810 REM        IF SPACE CLEAR SCREEN, RESTART
2820 IF A=60 THEN PRINT CHR$(147): GOTO 1310
2830 GOSUB 6000:           REM  GET NEW COLOR
2840 CC = CO :             REM  CC IS DISPLAY COLOR
2850 REM        POKE NEW COLOR INTO COLOR MAP
2860 POKE 55296 + 40*Y + X,CC
3000 :
3010 REM        IF F1 PRESSED, GET NEW
3020 REM        BACKGROUND COLOR
3030 IF A=4 THEN 3080
3070 GOTO 1530:            REM  LOOP
3080 C1 = CO :             REM  SAVE OLD COLOR
3090 A1 = PEEK(197)
3100 IF A1=A OR A1=64 THEN 3090
3110 A=A1
3120 GOSUB 6000
3130 POKE 53281,CO:        REM CHANGE BACKGROUND
3140 GOTO 1530
4998 :
4999 REM        MOVE CURSOR TO X,Y
5000 POKE S + 3,X
5010 POKE S + 1,Y
5020 SYS S  :              REM CALL MACHINE LANGUAGE
5030 RETURN
5996 :
5997 REM        TRANSLATE NUMBER KEYS INTO
5998 REM        COLORS
5999 REM                          KEY
6000 IF A=56 THEN CO = 0: REM  1 BLACK
6010 IF A=59 THEN CO = 1: REM  2 WHITE
6020 IF A= 8 THEN CO = 2: REM  3 RED
6030 IF A=11 THEN CO = 3: REM  4 CYAN
6040 IF A=16 THEN CO = 4: REM  5 PURPLE
6050 IF A=19 THEN CO = 5: REM  6 GREEN
6060 IF A=24 THEN CO = 6: REM  7 BLUE
6070 IF A=27 THEN CO = 7: REM  8 YELLOW
6080 RETURN
9997 :
9998 REM        DATA IS FOR READ IN LINE 1070
9999 DATA 162,20,160,15,24,76,240,255
```

Printing Etcher

This final version allows you to print your drawing. Hold down the function key F3 until your printer starts printing. If a printer is not attached or is not turned on, the program will give you an error message. To start over, you must run the program again. To print the picture after an error message, try

GOTO 30020. Typing GOTO 1530 may restart the program, leaving your drawing intact (with some error messages, though). Again, be sure that the joystick is in port #2.

```
5 REM       PRINTING ETCHER
6 REM   THIS IS CHANGE BACKGR  WITH
7 REM   LINES 3040-3060 AND
8 REM   29997 ON  ADDED
9 :
10 REM   HIT F1 UNTIL CURSOR STOPS AND
11 REM   THEN A COLOR KEY (NUMBERS 1 TO 8)
12 REM   TO CHANGE BACKGROUND COLOR
15 :
20 REM   HIT F3 TO PRINT SCREEN
21 :
22 :
90 PD = 1 :                  REM PD IS PENDOWN FLAG
1000 :
1010 REM         LINES 1020-1090 POKE A MACHINE
1020 REM         LANGUAGE ROUTINE THAT IS USED
1030 REM         IN LINE 5020
1040 S = 12*4096 :           REM  STARTS AT $C000
1050 PRINT CHR$(147)
1060 FOR I = 0 TO 7
1070 READ A
1080 POKE S + I,A
1090 NEXT
1300 :
1310 CH = 35:                REM  THIS IS THE CURSOR
1320 Y = 11 :                REM THE Y POSITION
1330 X = 19 :                REM THE X POSITION
1500 :
1510 GOSUB 5000 :            REM - CURSOR TO X,Y
1520 REM         READ JOYSTICK
1530 A = PEEK(56320) AND 31
1540 IF (A AND 16)=0 THEN PD = 1 - PD
1550 REM         PD IS PENDOWN FLAG
1560 REM         PD = 1-PD TOGGLES IT
1570 REM         A TOGGLE CHANGES THE STATE
1580 REM         E.G., FROM PENUP TO PENDOWN
1800 :
1810 REM         THE LINES 1830-1870 CHANGE
1820 REM         X,Y IN ACCORD WITH JOYSTICK
1830 A = A AND 15
1840 IF A AND 8 THEN X=X-1
1850 IF A AND 4 THEN X=X+1
1860 IF A AND 2 THEN Y=Y-1
1870 IF A AND 1 THEN Y=Y+1
2000 :
2010 REM         LINES 2040-2070 CORRECT FOR
```

```
2020 REM        X,Y POSITIONS THAT WOULD BE
2030 REM        OFFSCREEN
2040 IF X<0 THEN X=0
2050 IF Y<0 THEN Y=0
2060 IF Y>23 THEN Y=23
2070 IF X>39 THEN X=39
2400 :
2410 :
2420 REM        THE LINES 2440-2500 BLINK THE
2430 REM        CURSOR AT POSITION X,Y
2440 GOSUB 5000:            REM PUT CURSOR AT X,Y
2450 PRINT CHR$(32);
2460 GOSUB 5000
2470 PRINT CHR$(CH);
2480 GOSUB 5000
2490 PRINT CHR$(32);
2500 GOSUB 5000
2600 :
2610 REM        IF PEN IS UP(PD=0)DON'T PRINT
2620 IF PD = 0 THEN 2640
2630 PRINT CHR$(CH);
2640 A = PEEK(197) :        REM READ KEYBOARD
2800 :
2810 REM        IF SPACE CLEAR SCREEN, RESTART
2820 IF A=60 THEN PRINT CHR$(147) : GOTO 1310
2830 GOSUB 6000:            REM  GET NEW COLOR
2840 CC = CO :              REM  CC IS d} COLOR
2850 REM        POKE NEW COLOR INTO COLOR MAP
2860 POKE 55296 + 40*Y + X,CC
3000 :
3010 REM        IF F1 PRESSED, GET NEW
3020 REM        BACKGROUND COLOR
3030 IF A=4 THEN 3080
3040 REM        IF F2 NOT PRESSED THEN LOOP
3050 IF A<>5 THEN 1530
3060 GOSUB 30020 :          REM  DUMP SCREEN
3070 GOTO 1530:             REM  LOOP
3080 C1 = CO :              REM  SAVE OLD COLOR
3090 A1 = PEEK(197)
3100 IF A1=A OR A1=64 THEN 3090
3110 A=A1
3120 GOSUB 6000
3130 POKE 53281,CO:         REM  BACKGROUND
3140 GOTO 1530
4998 :
4999 REM        MOVE CURSOR TO X,Y
5000 POKE S + 3,X
5010 POKE S + 1,Y
5020 SYS S  :               REM CALL MACHINE LANGUAGE
5030 RETURN
5996 :
```

```
5997 REM        TRANSLATE NUMBER KEYS INTO
5998 REM        COLORS
5999 REM                              KEY
6000 IF A=56 THEN CO = 0: REM  1  BLACK
6010 IF A=59 THEN CO = 1: REM  2  WHITE
6020 IF A= 8 THEN CO = 2: REM  3  RED
6030 IF A=11 THEN CO = 3: REM  4  CYAN
6040 IF A=16 THEN CO = 4: REM  5  PURPLE
6050 IF A=19 THEN CO = 5: REM  6  GREEN
6060 IF A=24 THEN CO = 6: REM  7  BLUE
6070 IF A=27 THEN CO = 7: REM  8 YELLOW
6080 RETURN
9997 :
9998 REM        DATA IS FOR READ IN LINE 1070
9999 DATA 162,20,160,15,24,76,240,255
29997 :
29998 :
29999 REM       WAIT FOR KEYPRESS
30000 GET A$
30005 IF A$="" THEN 30000
30010 REM       STOP IF F1 IS NOT PRESSED
30011 IF A$<>CHR$(133) THEN RETURN
30020 OPEN 4,4,4 :          REM  ENABLE PRINTER
30030 CMD4 :                REM PRINT IT
30040 CG = PEEK(53272)
30050 SR = 1024 :           REM START OF SCREEN
30060 REM       FOR WHOLE 1000 CHARACTERS
30061 REM       ON THE SCREEN
30062 FOR I0 = 0 TO 999
30070 REM       GET SCREEN CONTENTS
30080 CZ = PEEK(SR + I0)
30090 IF CG=21 THEN GOSUB 30180
30100 IF CG=23 THEN GOSUB 30140
30110 REM       PRINT THE CHARACTER
30111 PRINT#4,CHR$(AZ);: LL = LL + 1
30120 IF LL=40 THEN PRINT#4,CHR$(13):LL = 0
30130 NEXT I0 : CLOSE4 : RETURN
30140 IF CZ<27 THEN AZ = CZ + 96:RETURN
30150 IF CZ<32 THEN AZ = CZ + 64:RETURN
30160 IF CZ<91 THEN AZ = CZ : RETURN
30170 AZ = 32 : RETURN
30180 IF CZ<32 THEN AZ = CZ + 64:RETURN
30190 IF CZ<64 THEN AZ = CZ : RETURN
30200 A  = 32 : RETURN
```

Read Joystick

This is a simple routine to show you what the joystick is saying. This routine is used in ETCH to control where the cursor goes. Be sure your joystick is connected to port #2.

```
5   REM       READ JOYSTICK
100 REM     56320 IS JOYSTICK 2
110 PRINT PEEK(56320) AND 31
120 GOTO 100
```

Keyscan

This is a simple routine to show you what the keyboard is saying. Run the program. Note what is displayed before you press a key. Which keys will not change the display? Change line 110 to examine other interesting locations in memory.

```
5  REM       KEYSCAN
6 REM  197 SHOWS KEY THAT IS PRESSED
7 REM  654 SHOWS IF THE SHIFT KEY IS
8 REM  PRESSED
90 :
100 PRINT PEEK(197),PEEK(654);
105 GET A$:PRINT A$:       REM NOW SHOW IT
110 GOTO 100
```

Paddle Read

This is a simple paddle read/display program. Well, it isn't as simple as we might like. The memory locations used to read the values of the paddles are also used for other purposes (such as seeing if someone's pressed a key). In Chapter 4, we discuss the idea that 60 times a second the 64 stops what it is doing and does some other things, for example, seeing if a key has been pressed on the keyboard. The halting of current work to check other things is called an interrupt. It causes a problem because if the 64 decides to use memory location 56320 to read the keyboard at (almost) the same time that we want to read the paddle, well—it gets everyone a bit confused. So we must set the interrupts off to read the paddles correctly. Note that we have to set the interrupts on after we read the paddles, or the keyboard will not work until RUN/STOP and RESTORE have both been pressed (at the same time).

Note that you could use this program and paddles to replace a joystick. A joystick gives direction, whereas paddles give X and Y values. Otherwise, they can do just about the same things if the programming is right.

```
5 REM       PADDLE READ
90 :
100 REM  POKE 56320     TO READ
110 REM    WITH           PORT
120 REM     127            1
130 REM     191            2
```

```
140 :
160 POKE 56334,0 :          REM INTERRUPT OFF
170 POKE 56320,127 :        REM PORT 1
200 P1 = PEEK(54297):       REM  PADDLE 1
210 P2 = PEEK(54298):       REM  PADDLE 2
220 REM          PADDLE FIRE BUTTONS
230 F1 = PEEK(56320) AND 4
240 F2 = PEEK(56320) AND 8
250 POKE 56334,129 :        REM INTERRUPTS ON
300 PRINT P1,P2,F1,F2
310 GET A$: IF A$="" THEN 160
```

Challenges

1. Change PRINTING ETCHER so that the character used for drawing is input from the keyboard.
2. Find a way to access the other eight colors that are available on the Commodore 64.
3. Add a sound routine to PRINTING ETCHER.
4. Add a routine to ETCH so that two people can draw at the same time. You could then make this into a "Blockade"-type game.
5. Add a random number routine to ETCH that would control the cursor movement on the screen or the character used for drawing.
6. Assume you have a finished drawing on the screen. Add a routine that would change the character used in your drawing to another character from the keyboard.
7. Make all the programs respond to the cursor keys instead of a joystick.

   (Hint: The function keys would be useful for both challenges 1 and 6.)

3

WHEEL OF FORTUNE

Programs:

DIE	This routine contains a function that simulates a die roll.
DICE	This program rolls two dice.
DICE 1	This program uses a better function to simulate a die roll.
DECIMAL DICE	This simulates the roll of 10-sided dice.
DICE ROLLS	Shades of Monte Carlo.
DICE ROLLS 1	This program graphs a hundred die rolls.
WILD SCREEN	Seeing is believing.
RANDOM COLOR	This program fills the screen with all the 64's colors.
RANDOM WINDOW	Fastest sprite mover ever.
BINGO	Calls a mean bingo game.
BINGO CARD	This program draws a bingo card on your screen.
BINGO CARD 1	This version prints bingo cards.
CARD DEALER	An honest dealer at long last.

The programs in this chapter all use the RND function to make things happen randomly. They will make your 64 do the following:

1. Make a die roll.
2. Make dice with any number of sides.
3. Make the dice roll a hundred times.
4. Make a simple graph.
5. Randomly change the background and border colors.
6. Fill the screen with random-colored squares.
7. Randomly locate a sprite.
8. Shuffle and deal a deck of cards.
9. Call the numbers for a bingo game.
10. Print a bingo card.

BASIC Commands Used in This Chapter

All of the programs use the RND(0) function to randomly do things.

DATA	OPEN
DEF	PEEK
DIM	POKE
END	PRINT
FOR	PRINT#
GET	READ
GOSUB	REM
GOTO	RND
INT	SPC
IF	STR$
LEN	THEN
NEXT	

Programming Techniques Used in This Chapter

1. *Making random numbers in a given range.* For example, 1 to 6 for dice, 1 to 75 for bingo, and so forth. The RND function produces numbers between 0 and 1. We use function definitions to get numbers in other ranges. See line 130 in DIE, line 130 in DECIMAL DICE, and line 130 in WILD SCREEN (also see line 180 for a random length delay).
2. *Formatted printing.* Making the printout pretty, that is. Spacing with the SPC function makes nicer looking printouts. See lines 270 and 300 in

DICE ROLLS, where we first make a number a string (STR$) and then see how long the string is (LEN), printing an extra space if there is only one digit. A graph can be made with SPC, as in line 250 of DICE ROLLS 1. Line 740 in BINGO CARD makes sure that single- and double-digit numbers line up. Line 710 in BINGO CARD 1 centers the title "BINGO CARD," and lines 780 to 840 make a rather nice printout.

3. *Creating random draws.* Creating random draws for a bingo card or from a deck of cards. We want to make the computer act like it is taking bingo balls out of the urn, that is, so that no number is repeated (this is known in statistics as sampling without replacement). A sophisticated technique is used to do this in both BINGO and CARD DEALER. The idea is to make an array that has the list of numbers that we want—in order. For BINGO CARD, we use 75 numbers. We make a random number and print it. But to make sure that we do not get it again, we replace that entry in the array (A(I)) with the last entry. For example, at the start we have A(I) = I. Suppose 10 is the random number on the first try. Then we print A(10), which is 10 on the first round. Now we make A(10)=A(75), which is 75. Now we have replaced 10, and it will never be printed again. On the second round, we make a number between 1 and 74. If it is 10, then we print A(10), which is 75, and make A(10)=A(74) so that 75 will never be repeated again. If the second number were 15 instead of 10, then we print A(15) and put A(74) into A(15). On the third round, we make a number between 1 and 73, and so on. See lines 250 to 280 in BINGO. This is a very sophisticated technique that can be used in other situations where you want to make sure to never repeat a number but also want to make sure that you will use every number.
4. *Counting.* In DICE ROLLS we want to count how many times two is the sum of the dice, three is the sum of the dice, and so forth. Rather than using 11 variables, we use the array A(I). Whatever the sum of the dice is, we add one to that element of the array (the element that is the sum of the dice). See lines 190 and 200 in DICE ROLLS.
5. *Watching the keyboard.* This means looking for any key to be hit and when it is, then doing something. See lines 180 and 190 in DECIMAL DICE. We might want to do something different, depending upon which key is hit. See lines 200 to 240 in BINGO.

Die

This routine contains a function that simulates a die roll. The function argument "1" in line 160 is a dummy; it is not used at all by the function in line 130. Take a minute to experiment with the function in line 130. One

possibility is to change the RND(0) to RND(1) or RND(−1). If you want to see an endless number of die rolls, add the following line:

```
170 GOTO 160
```

```
5 REM             DIE
100 REM  LINE 130   DEFINES A
110 REM  FUNCTION WHICH GIVES A RANDOM
120 REM  NUMBER BETWEEN 1 AND 6
130 DEF FND(X) = INT(6*RND(0)+1)
140 :
150 REM         PRINT A RANDOM DIE THROW
160 PRINT FND(1)
```

Dice

This program rolls two dice.

```
5 REM             DICE
6 REM  THIS IS DIE WITH LINE 160 CHANGED
100 REM  LINE 130   DEFINES A
110 REM  FUNCTION WHICH GIVES A RANDOM
120 REM  NUMBER BETWEEN 1 AND 6
130 DEF FND(X) = INT(6*RND(0)+1)
140 :
150 REM         PRINT 2 RANDOM DIE THROWS
160 PRINT FND(1),FND(1)
```

Dice 1

This program uses a better function to simulate a die roll. Note that the argument 6 in line 160 is no longer a dummy argument. Now it's used by the function. Try changing it.

```
5 REM             DICE 1
6 REM  THIS IS DICE WITH LINES
7 REM  120-130 AND 160 CHANGED
100 REM  LINE 130   DEFINES A
110 REM  FUNCTION WHICH GIVES A RANDOM
120 REM  NUMBER BETWEEN 1 AND X
130 DEF FND(X) = INT(X*RND(0)+1)
140 :
150 REM         PRINT 2 RANDOM DIE THROWS
160 PRINT FND(6),FND(6)
```

Decimal Dice

This simulates the roll of many-sided dice where you tell the program how many sides there are on each die. You get a new roll every time you press a key. Press the RUN/STOP key to stop the program.

```
5 REM       DECIMAL DICE
6 REM  THIS IS DICE 1  WITH LINES
7 REM  LINES 155 TO 190 ADDED
100 REM  LINE 130   DEFINES A
110 REM  FUNCTION WHICH GIVES A RANDOM
120 REM  NUMBER BETWEEN 1 AND 6
130 DEF FND(X) = INT(X*RND(0)+1)
140 :
150 REM         PRINT 2 RANDOM DIE THROWS
155 INPUT "HOW MANY SIDES";S
160 PRINT FND(S),FND(S)
170 PRINT "PRESS ANY KEY TO ROLL AGAIN"
180 GET A$: IF A$="" THEN 180
190 GOTO 160
```

Dice Rolls

This program rolls a pair of dice 360 times and counts the number of times it gets 2 to 12 for the sum of the dice. It takes your Commodore 64 about 15 seconds to do this. The number 2 should occur about 10 times, 3 should occur about 20 times, 4 about 30 times, 5–40 times, 6–50 times, 7–60 times, 8–50 times, 9–40 times, 10–30 times, 11–20 times, and 12–10 times. See how this compares to what you get. You might want to increase the 360 to 90,000 and go have dinner while waiting for the answer; it will take about an hour. If you do, change line 280 to:

```
280 PRINT 100*A(I)/90000; " OF ";
```

This will print the percentage of times that each sum occurred for the 90,000 rolls. The more times you roll, the closer the fractions should come to the following probabilities:

SUM	*FRACTION*
2	1/36
3	1/18
4	1/12
5	1/9
6	5/36
7	1/6
8	5/36
9	1/9
10	1/12
11	1/18
12	1/36

This sort of simulation of random events is called a "Monte Carlo" experiment—although it's not the same thing as going to Monte Carlo. Try some other Monte Carlo experiments. For example, change line 190 and 240 to:

```
190 J = FND(3)*FND(3)
240 FOR I = 1 TO 9
```

Try to guess what will happen before you RUN it.

```
5 REM             DICE ROLLS
6 REM  THIS IS DICE 1   WITH LINES
7 REM  150-160 CHANGED  AND LINES
8 REM  170-320 ADDED
100 REM  LINE 130   DEFINES A
110 REM  FUNCTION WHICH GIVES A RANDOM
120 REM  NUMBER BETWEEN 1 AND X
130 DEF FND(X) = INT(X*RND(0)+1)
140 :
150 PRINT CHR$(147) :      REM  CLEAR SCREEN
160 DIM A(12)
170 REM         GET 360 DICE ROLLS
180 FOR I= 1 TO 360
190 J =  FND(6) + FND(6)
200 A(J) = A(J) + 1
210 NEXT I
220 :
230 REM         NOW PRINT OUT RESULTS
240 FOR I = 2 TO 12
250 PRINT "THERE WERE ";
260 REM         THIS PRINTS IN A COLUMN 6 WIDE
270 PRINT SPC(6-LEN(STR$(A(I))));
280 PRINT A(I);" OF ";
290 REM         THIS PRINTS IN A COLUMN 4 WIDE
300 PRINT SPC(4-LEN(STR$(I)));
310 PRINT I
320 NEXT I
```

Dice Rolls 1

This program graphs a hundred die rolls. Note that if you exceed 100 trials, then the graph may not work. We count on there not being more than 28 occurrences of any number, or the graph gets ruined.

```
5 REM             DICE ROLLS 1
6 REM  THIS IS DICE ROLLS  WITH
7 REM  LINES 250-310 CHANGED
100 REM         LINE 130   DEFINES A
110 REM         FUNCTION WHICH GIVES A RANDOM
120 REM         NUMBER BETWEEN 1 AND X
```

```
130 DEF FND(X) = INT(X*RND(0)+1)
140 :
150 PRINT CHR$(147) :      REM  CLEAR SCREEN
160 DIM A(12)
170 REM          GET 100 DICE ROLLS
180 FOR I= 1 TO 100
190 J =  FND(6) + FND(6)
200 A(J) = A(J) + 1
210 NEXT I
220 :
230 REM          NOW PRINT OUT RESULTS
240 FOR I = 2 TO 12
260 REM          THIS PRINTS IN A COLUMN 4 WIDE
270 PRINT SPC(4-LEN(STR$(I)));
280 PRINT I;
290 REM          THIS PRINTS IN A COLUMN 4 WIDE
300 PRINT SPC(4-LEN(STR$(A(I))));
310 PRINT A(I);"I";SPC(A(I));"*"
320 NEXT I
```

Wild Screen

Seeing is believing, but we do not recommend looking at this one too long. Press the RUN/STOP and RESTORE keys together to stop and get to the usual screen colors. To make the screen blink slower, change the 50 to 100 or 500 in line 170.

```
5 REM            WILD SCREEN
90 :
100 REM          BR IS THE BORDER COLOR LOCATION
110 REM          BR+1 IS FOR BACKGROUND COLOR
120 BR = 53280
130 :
140 B=INT(16*RND(0)):      REM  RANDOM COLOR
150 POKE BR+INT(2*RND(0)),B
160 REM          THIS IS A RANDOM DELAY LOOP
170 FOR I = 1 TO 50*RND(0)
180 NEXT I
190 GOTO 140
```

Random Color

This program randomly fills the screen with all the 64's colors.

```
5 REM          RANDOM COLOR
10 REM  THIS PROGRAM MAY WORK ON YOUR
11 REM  64 WITHOUT LINES 130-150
12 REM  TRY IT
30 :
```

```
50 DEF FND(X) = INT(X*RND(TI)+1)
90 :
100 PRINT CHR$(147) :      REM  CLEAR SCREEN
110 PRINT CHR$(18); :      REM  REVERSE ON
120 REM         FILL SCREEN WITH REVERSE SPACES
130 FOR I=1 TO 999
140 PRINT CHR$(160);
150 NEXT I
160 :
170 POKE 55295+FND(1024),FND(16)-1
180 GOTO 170          :      REM  LOOP FOREVER
```

Random Window

This program is the fastest sprite mover ever. If you are unacquainted with sprites, you might want to read Chapter 7, "Video Arcade," (or at least its introduction). But watch this "window" move randomly around the screen. Hit any key to stop.

```
5 REM           RANDOM WINDOW
6 REM  THIS IS WINDOW  WITH
7 REM  LINES 280-310  CHANGED OR ADDED
30 :
50 L1 = 3*4096
60 DEF FND(X)=INT(X*RND(1))
90 :
100 PRINT CHR$(147) :       REM CLEAR SCREEN
110 REM         SET SPRITE 0 TO POINT TO
120 REM         192 IN MEMORY
130 POKE 2040,192:          REM SET SPRITE 0 PTR
140 REM         POKE 1 INTO SPRITE LOCATIONS
150 FOR S=L1 TO L1+62
160 POKE S,1
170 NEXT
180 FOR S=0 TO 2
190 POKE L1+S,255
200 POKE L1+60+S,255
210 NEXT :                  REM FILL THESE (255)
220 FOR S=3 TO 60 STEP 3
230 POKE L1+S,PEEK(L1+S) OR 128
240 NEXT :                  REM  FILL IN
250 :
260 S=53248 :               REM FIRST VIC REGISTER
270 POKE S+21,1 :           REM  DISPLAY SPRITE 0
280 POKE S+39,FND(16):      REM SET COLOR
290 POKE S,FND(236)+20:     REM  X POSITION
300 POKE S+1,FND(200)+40:   REM  Y POSITION
310 FOR I= 1 TO 1000:NEXT :REM WAIT A BIT
320 GET A$: IF A$="" THEN GOTO 280 : REM  LOOP
330 POKE S+21,0
```

Bingo

This program calls a clean mean bingo game (calling numbers only, though). Use BINGO CARD or BINGO CARD 1 to make a bingo card to play with.

```
5 REM           BINGO
29 :
30 REM  ARRAY FOR RANDOM SHUFFLE
40 DIM A(75)
50 DEF FND(X) = INT(X*RND(0)) + 1
90 :
100 PRINT CHR$(147)  :     REM CLEAR SCREEN
110 REM              SET UP ARRAY
120 FOR I = 1 TO 75
130 A(I) = I
140 NEXT I
150 :
160 PRINT "LET'S START A NEW GAME"
190 I = 75
200 PRINT "HIT A KEY FOR A NUMBER"
210 PRINT "R-RESTART, X-STOP"
220 GET A$: IF A$="" THEN 220
230 IF A$ = "R" THEN 100 :REM RESTART
240 IF A$ = "X" THEN 990 :REM EXIT
250 NU = FND(I)
260 PRINT "NUMBER";76-I;" IS";A(NU)
270 A(NU) = A(I)
280 I = I - 1
290 IF I>0 THEN 200 :      REM CONTINUE
300 PRINT "THAT'S ALL"
310 GET A$: IF A$="" THEN 310
320 GOTO 100  :            REM  RESTART
990 END
```

Bingo Card

This program prints a bingo card on your screen, generating the numbers randomly, of course.

```
5 REM       BINGO CARD
29 :
30 REM  ARRAY FOR RANDOM SHUFFLE
40 DIM A(15),CA(5,5)
50 DEF FND(X) = INT(X*RND(0)) + 1
90 :
100 PRINT CHR$(147)  :     REM CLEAR SCREEN
200 FOR J = 1 TO 5
210 GOSUB 1000 :           REM  INITIALIZE
220 FOR K = 1 TO 5 :       REM  FILL A COLUMN
230 GOSUB 2000
```

```
240 CA(J,K) = 15*(J-1) + NU
250 NEXT K
260 NEXT J
700 REM         PRINT A CARD
705 PRINT
710 FOR J = 1 TO 5
715 PRINT
720 FOR I = 1 TO 5
730 IF I=3 AND J=3 THEN PRINT"  X   ";: GOTO 760
740 IF LEN(STR$(CA(I,J)))=2 THEN PRINT SPC(1);
750 PRINT CA(I,J);SPC(2);
760 NEXT I
770 PRINT
780 NEXT J
990 END
997 :
998 REM         INITIALIZE RANDOM GENERATOR
999 REM         SET UP ARRAY
1000 FOR I = 1 TO 15
1010 A(I) = I
1020 NEXT I
1030 I = 15
1040 RETURN
1998 :
1999 REM        RANDOM NUMBER GETTER
2000 B = FND(I)
2010 NU = A(B)
2020 A(B) = A(I)
2030 I = I - 1
2040 RETURN
```

Bingo Card 1

This version prints bingo cards.

```
5 REM       BINGO CARD 1
6 REM  THIS IS BINGO CARD  WITH THE
7 REM  PRINTING OF A CARD SUBSTITUTED FOR
8 REM  SCREEN DISPLAY IN LINES 700-900
9 REM  AND LINES 60-70 ALSO ADDED
29 :
30 REM  ARRAY FOR RANDOM SHUFFLE
40 DIM A(15),CA(5,5)
50 DEF FND(X) = INT(X*RND(0)) + 1
60 REM          ENABLE PRINTER
70 OPEN 4,4
90 :
100 PRINT CHR$(147)   :     REM CLEAR SCREEN
200 FOR J = 1 TO 5
210 GOSUB 1000 :            REM  INITIALIZE
220 FOR K = 1 TO 5 :        REM  FILL A COLUMN
```

```
230 GOSUB 2000
240 CA(J,K) = 15*(J-1) + NU
250 NEXT K
260 NEXT J
700 REM         PRINT A CARD
710 PRINT#4,SPC(12);"BINGO"
720 PRINT#4,
730 PRINT#4,":----:-----:-----:-----:-----:"
740 FOR J = 1 TO 5
750 PRINT#4,":    :     :     :     :     :";
760 PRINT#4,CHR$(13);":";
770 FOR I = 1 TO 5
780 IF I=3 AND J=3 THEN PRINT#4,"  X : ";:GOTO 810
790 IF LEN(STR$(CA(I,J)))=2 THEN PRINT#4,SPC(1);
800 PRINT#4,CA(I,J);": ";
810 NEXT I
820 PRINT#4,
830 PRINT#4,":    :     :     :     :     :"
840 PRINT#4,":----:-----:-----:-----:-----:"
850 NEXT J
990 END
997 :
998 REM         INITIALIZE RANDOM GENERATOR
999 REM         SET UP ARRAY
1000 FOR I = 1 TO 15
1010 A(I) = I
1020 NEXT I
1030 I = 15
1040 RETURN
1998 :
1999 REM        RANDOM NUMBER GETTER
2000 B = FND(I)
2010 NU = A(B)
2020 A(B) = A(I)
2030 I = I - 1
2040 RETURN
```

Card Dealer

At long last, here's an honest dealer. In line 50, try changing the RND(0) to RND(1). What effect did this change have on the cards dealt? Run the program several times.

```
5 REM       CARD DEALER
6 REM  LINES 990-2040 ARE FROM
7 REM  BINGO CARD WITH LINES 1000
8 REM  AND 1030 CHANGED
29 :
30 REM  ARRAY FOR RANDOM SHUFFLE
40 DIM A(52),S$(4),C$(13)
50 DEF FND(X) = INT(X*RND(0)) + 1
```

```
90 :
100 REM         READ IN CARDS, SUITS
110 FOR C = 1 TO 13
120 READ C$(C)
130 NEXT C
140 FOR S = 1 TO 4
150 READ S$(S)
160 NEXT S
170 DATA ACE,DEUCE,THREE,FOUR,FIVE
180 DATA SIX,SEVEN,EIGHT,NINE,TEN
190 DATA JACK,QUEEN,KING
200 DATA CLUBS,DIAMONDS,HEARTS,SPADES
205 DATA CLUBS,DIAMONDS,HEARTS,SPADES
210 :
220 REM         NOW INITIALIZE THE DECK
230 GOSUB 1000
300 PRINT CHR$(147)  :     REM CLEAR SCREEN
310 REM         NOW DEAL A CARD, AND WAIT
320 GOSUB 2000
330 S = INT((NU-1)/13) + 1
340 C = NU - 13*(S-1)
350 PRINT SPC(10);C$(C);" OF ";S$(S)
360 PRINT "HIT ANY KEY TO DEAL ANOTHER"
370 GET A$
380 IF A$="" THEN 370
390 IF I > 0 THEN 320
400 PRINT "THAT'S ALL"
990 END
997 :
998 REM         INITIALIZE RANDOM GENERATOR
999 REM         SET UP ARRAY
1000 FOR I = 1 TO 52
1010 A(I) = I
1020 NEXT I
1030 I = 52
1040 RETURN
1998 :
1999 REM        RANDOM NUMBER GETTER
2000 B = FND(I)
2010 NU = A(B)
2020 A(B) = A(I)
2030 I = I - 1
2040 RETURN
```

Challenges

1. Add pictures of the dice as they are rolled.
2. Add pictures of the cards in CARD DEALER.
3. Corrupt the CARD DEALER or the DICE (by changing the odds in your own favor).

4. How about adding the following to the BINGO game?
   a. The appropriate letter to go with the number
   b. A DISPLAY CHAR routine (see Chapter 6, "Odds & Ends")
   c. A sound routine
   d. A routine to display the numbers that have been called
5. Modify CARD DEALER to become a simple solitaire card game.

4

WHAT TIME IS IT?

Programs:

Some easy timing programs:

TIMING LOOP	This program shows you how long it takes BASIC to loop a thousand times.
STOPWATCH	This is a stopwatch to a hundredth of a second.
STOPWATCH 1	This is a cleaner version of STOPWATCH.
HOW LONG	This displays the internal "jiffy" clock.
QUICK DRAW	This is a fast-draw contest with your computer.

The "Jiffy" clock:

SET CLOCK	This allows you to set the time on your jiffy clock.
SET.RUN CLOCK	This sets and displays your clock.
VIEW CLOCK 1	This clock displays the hour, minutes, and seconds.
VIEW CLOCK 2	This clock displays the hour and minutes.
WORLD CLOCK	This is our final version of the hour and minute clock. It displays the time from two different time zones.

The Time of Day (TOD) clock:

SET TOD	This program sets and starts one of your TOD clocks.
VIEW TOD 1	This displays the TOD clock with hour, minutes, and seconds.
VIEW TOD 2	This is a different program for displaying the TOD clock.
SET TOD 1	This program sets your TOD clock with an A.M. and P.M. indicator.
VIEW TOD 3	This displays the TOD clock.

The programs in this chapter will make your 64 act like a clock or stopwatch. The various clocks can do the following things:

1. Time a BASIC program
2. Allow you to time any event
3. Determine how long your computer has been on
4. Test your reaction time
5. Display the time on your screen
6. Show the time in two different time zones
7. Display a 12-hour clock with A.M. and P.M.

The Clocks in Your Commodore

Inside the 64 are two different types of clocks: a jiffy clock that the Commodore software programmers created, and two hardware time of day (TOD) clocks that the hardware designers put in your computer. Each type of clock has different capabilities and differs greatly in the way you access them. This chapter contains programs that will show you how to use both types and will illustrate their pros and cons.

The Jiffy Clock

The jiffy clock works in a rather peculiar way (to noncomputers, at least). Sixty times a second, the 64 halts its normal work (technically, it is interrupted), and does several things, including updating a counter. This counter is set to zero when you turn the 64 on, so its count is really the number of sixtieths of seconds that the computer has been turned on. The special variable TI always has this count in it. If you wish to see the current count, simply type in:

```
PRINT TI
```

This should print a number, possibly very large if you turned on your computer a while ago—in fact, the number could even be a billion, if you turned your 64 on half a year ago!

Since most people don't tell time in jiffies (that's what we call a sixtieth of a second), the Commodore software people also made it possible to get the jiffy count in ordinary hours, minutes, and seconds. They created another variable, TI$, a string variable, that is used in the same way as the TI variable:

```
PRINT TI$
```

which will print a six-digit number, such as:

```
031452
```

The way to interpret this number is as three pairs of numbers: hours, minutes, and seconds (or HHMMSS). The first pair is the hours, 03 hours; the second pair is the minutes, 14 minutes; and the right-hand pair is the seconds, 52 seconds. If you got the result above, it means that you had turned on your Commodore 64 3 hours, 14 minutes, and 52 seconds ago.

A very important feature of any clock is the ability to set it. For the jiffy clock, this is easy. Suppose the current time is 3 minutes and 5 seconds after 2 o'clock. First, translate this into the HHMMSS form: 02 hours, 03 minutes, 05 seconds. Thus, the HHMMSS form is 020305. To set your jiffy clock, simply type:

```
TI$ = "020305"
```

If you're quick, you can do this before too many seconds go by. It's better to prepare a few seconds ahead of time and hit the RETURN key just as the right time passes. Try setting the TI$ now. Now PRINT TI again.

What happened? There is only a single jiffy counter inside the 64. The TI and TI$ are simply two different ways to use the same counter. Thus, if you reset the TI$ variable, the TI variable will also change.

You cannot set the jiffy clock with the variable TI in the same way you can with TI$—it just can't be done. The only way to set the jiffy clock is with the TI$ method. Now for a serious question: If the TI$ way of accessing the jiffy clock makes better sense to us humans (you *are* human, aren't you?), then why would we ever use the TI variable?

An excellent question, even if we did ask it. Remember that the jiffy counter records in jiffies, or sixtieths of a second. But when you use TI$ to display the clock, it only displays whole seconds. Thus, if something takes less than a second to happen, the TI$ variable may not change, even though the jiffy counter has. This problem can be called one of resolution, or how finely your measuring tool can resolve small differences. The jiffy clock has a

resolution of one jiffy (1/60 of a second), but the TI$ clock has a resolution of only one second. In order to take advantage of the full resolution of the jiffy clock, we need to use the TI variable. Type in the following simple program, after typing NEW to clear the memory:

```
10 BE = TI
20 B  = 1/3
30 EN = TI
40 PRINT EN - BE
```

LIST the program and check it. If it's OK, then RUN it. The result should be the number of jiffies that it took your 64 to execute the instruction in line 20. Line 10 saved the current jiffy count. Line 20 then divided 1 by 3 and stored it as variable B. Line 30 saved the new jiffy count when it was executed. Finally, line 40 prints out the difference in the counts, which should be just the time it took your 64 to process lines 20 and 30. On our 64, the result was usually 1. Try some other statements in line 20, such as multiplication. This should give you some idea as to how fast your 64 really is. Note that most of the time you will see either a 0 or a 1 because most BASIC statements take less than a sixtieth of a second. Some of the programs in this chapter are designed to see how long it takes to do something in BASIC.

The Time of Day Clock

The time of day (TOD) clock works very differently. Remember that the software people built the jiffy clock, and they made it easily accessible. On the other hand, the hardware people put in the TOD clock and apparently didn't get the software people to do anything to make it as accessible. To use the TOD clock, you must control it through PEEKs and POKEs to memory locations. Unlike the jiffy clock, the TOD clock isn't set automatically when the 64 is turned on. It only starts when you set its time. Each of the two TOD clocks has four memory locations representing hours, minutes, seconds, and tenths of seconds. Their locations are:

	TOD 1	*TOD 2*
Hours	56331	56587
Minutes	56330	56586
Seconds	56329	56585
Tenths of seconds	56328	56584

To start either TOD clock, you must POKE the time into the memory locations, and you must POKE all four of the registers in sequence—from hours to tenths of seconds—to properly set and start the clock. For example,

```
POKE 56331,0
POKE 56330,0
POKE 56329,0
POKE 56328,0
```

This will reset the clock to zero and start it. Starting the clock at times other than 0 is somewhat harder (see the program SET TOD 1). In order to read the clock, you must PEEK the same locations in the same order and then convert the results of those PEEKs into hours, minutes, seconds, and tenths of seconds. (See the program VIEW TOD 1, in which the necessary conversion between the numbers that we humans use and the numbers that the TODs understand is made.)

How do the TOD clocks differ from the jiffy clock? First, the TOD clocks have less resolution. The jiffy clock can measure time differences in jiffies (sixtieths of seconds), whereas the TOD clock can only measure in tenths of seconds. Second, the TOD clocks are much harder to use. Third, the jiffy clock is not counting during tape and disk operations and hence may be inaccurate for timing long events, such as how long your computer has been turned on. The TOD clocks are always ticking away, so they serve better for keeping track of the time of day.

To summarize, for timing short events where precision is important or for making a simple timer, use the jiffy clock; use the TOD clocks whenever you use disk or tape, or, more generally, for timing long events.

BASIC Commands Used in This Chapter

AND	MID$
CHR$	NEXT
DEF	PEEK
FOR	POKE
GET	PRINT
GOTO	RND
INPUT	REM
INT	RIGHT$
LEFT$	STR$
LEN	VAL

Programming Techniques Used in This Chapter

1. *Converting between different measurement units.* From jiffies to seconds, see TIMING LOOP line 140. In STOPWATCH, line 200 prints seconds to two decimal places, and in STOPWATCH 1, lines 40 and 200 use a function to print seconds to two decimal places.

2. *Waiting for a keypress.* See STOPWATCH lines 140 and 180.
3. *Clearing the screen.* See STOPWATCH 1 line 100.
4. *Using string functions.* Use string functions to take apart a large string and make smaller ones from it. In HOW LONG, lines 150–190 show how to print TI$ in parts.
5. *Putting strings together (called string concatenation).* See VIEW CLOCK 1, line 1030, and SET CLOCK, line 1070.
6. *Converting between different representations of a number.* This might be confusing, but it is necessary to use the TOD clocks. In SET TOD at line 100, we take a regular number in decimal form and convert it to something called BCD (or Binary Coded Decimal, since you asked), which is what is needed to set the TOD clock. BCD is just one way to code numbers to the computer. For some designs, such as the TOD clock, it is easier to do it this way. Note that all the numbers you enter into the 64 must be changed from decimal (which is how most of us think) into some other form because the 64 does not speak decimal numbers. Except for the TOD clock, the user never needs to know that this is happening because the 64 converts back to decimal when numbers are printed out.
7. *Defining a function.* Define a function so that a formula may be used without retyping it. This makes for both fewer keystrokes and a somewhat more readable program. See:

STOPWATCH 1	line 40
SET TOD	lines 100,1060,1090,1120
VIEW TOD 2	lines 100,110,1020,1040,1060
SET TOD 1	lines 100,1090,1120,1150
VIEW TOD 3	lines 100,110,1020,1040,1060

8. *Switching between numbers and characters with VAL and STR$.* That is, a number can be a number, or it can be a string. A string can be a number, but only if it is a number, right? See WORLD CLOCK, lines 1060,1100; and SET TOD, line 1050.

Some Easy Timing Programs

Timing Loop

This program shows you how long it takes BASIC to loop a thousand times. Try adding statements to the loop between lines 110 and 120 to time them. For example, try:

```
115 PRINT I
```

Note how line 100 reads the jiffy clock. See page 113 of your *Commodore 64 User's Guide* for more information about the variables TI and TI$.

```
5 REM           TIMING LOOP
6 REM  PRINTS TIME IT TAKES TO LOOP
7 REM  1000 TIMES
100 T = TI :                REM GET THE TIME
110 FOR I = 1 TO 1000
120 NEXT I
130 REM         TI-T IS THE ELAPSED TIME
140 PRINT(TI-T)/60;"SECONDS"
150 GOTO 100
```

Stopwatch

This is a stopwatch displaying in hundredths of a second. Note the similarities between lines 150 and 200 in this program and lines 100 and 140 in TIMING LOOP. Can you explain the difference in the printout of the elapsed time between the two? The following statements will serve as a hint:

```
PRINT 1/9
PRINT INT(100*1/9)/100
```

```
5 REM           STOPWATCH
50 :
100 PRINT CHR$(147) :     REM  CLEAR SCREEN
110 PRINT "HIT ANY KEY TO START,"
120 PRINT "HIT AGAIN TO STOP"
130 REM         WAIT FOR A KEYPRESS
140 GET A$ : IF A$="" THEN 140
150 T = TI
160 PRINT "START"
170 REM         WAIT FOR A KEYPRESS
180 GET A$ : IF A$="" THEN 180
190 REM         PRINTS IN HUNDREDTHS OF SECONDS
200 PRINT (INT(100*((TI-T)/60))/100);" SECONDS"
210 GOTO 110 :              REM  LOOP FOREVER
```

Stopwatch 1

This is a cleaner version of STOPWATCH. The use of the function definition in line 40 greatly simplifies the print statement in line 200. This function in line 40 first takes the argument X and subtracts it from the current jiffy count, TI. Since jiffies are supposed to be sixtieths of a second, the elapsed jiffies (TI–X) are divided by 60 to convert to seconds. Finally, the accuracy of the elapsed time is fixed to a hundredth of a second by first multiplying by 100, taking the INTeger part, and then dividing by 100. To set the accuracy to tenths of seconds, multiply and divide by 10 instead of 100.

```
5 REM           STOPWATCH 1
6 REM  THIS IS STOPWATCH WITH LINE 40
7 REM  ADDED AND LINE 200 CHANGED
8 REM  CLEANS UP THE PRINT STATEMENT IN
9 REM  LINE 200
30 :
40 DEF FNT(X)=(INT(100*((TI-X)/60))/100)
90 :
100 PRINT CHR$(147) :     REM  CLEAR SCREEN
110 PRINT "HIT ANY KEY TO START,"
120 PRINT "HIT AGAIN TO STOP"
130 REM         WAIT FOR A KEYPRESS
140 GET A$ : IF A$="" THEN140
150 T = TI
160 PRINT "START"
170 REM         WAIT FOR A KEYPRESS
180 GET A$ : IF A$="" THEN180
190 REM         PRINTS IN HUNDREDTHS OF SECONDS
200 PRINT FNT(T);" SECONDS"
210 GOTO 110 :            REM  LOOP FOREVER
500 PRINT FNT(T);" SECONDS"
```

How Long

This displays the internal jiffy clock. Note how TI$ is used in place of TI. TI$ is a character string containing six numbers. The two far left characters represent the hour: Line 150. The middle two characters represent minutes: Line 160. The two far right characters represent seconds: Line 180. To reset the clock to zero, type:

```
TI$ = "000000"
```

```
5 REM        HOW LONG
110 PRINT CHR$(147):      REM CLEAR SCREEN
120 PRINT "OOH"
130 PRINT "YOU TURNED ME ON ";
140 T$ = TI$
150 PRINT LEFT$(T$,2);" HOURS,";
160 PRINT " ";MID$(T$,3,2);
170 PRINT " MINUTES"
180 PRINT "AND ";RIGHT$(T$,2);
190 PRINT " SECONDS AGO."
200 FOR I = 1 TO 200 :    REM  DELAY LOOP
210 NEXT I
220 GOTO 130
```

Quick Draw

This is a fast-draw contest with your computer. This program shows how long it takes you to press a key after GO is printed on the screen. Note: The program checks for cheating.

```
5 REM           QUICK DRAW
6 REM  BASED ON STOPWATCH 1
7 REM  LINES 180 IS CRUNCHED FOR
8 REM  BETTER ACCURACY
40 DEF FNT(X)=(INT(100*((TI-X)/60))/100)
90 :
100 PRINT CHR$(147) :    REM  CLEAR SCREEN
110 PRINT "WHEN READY, HIT A KEY"
120 PRINT "WAIT FOR GO TO BE TYPED,"
130 PRINT "THEN HIT A KEY"
140 REM         RANDOM DELAY LOOP
150 Z = 1000*RND(1) + 200
160 FOR I = 1 TO Z
170 NEXT
180 T = TI:GET A$:IFA$=""THEN PRINT "GO": GOTO 210
190 GOTO 2000
210 GET A$ : IF A$="" THEN 210
230 PRINT FNT(T);" SECONDS"
240 GOTO 120 :            REM  LOOP FOREVER
1990 REM        CHEATER
2000 PRINT "YOU LOSE, CHEATER"
2010 GOTO 120
```

The Jiffy Clock

Set Clock

This allows you to set the time on your jiffy clock. Line 1070 actually sets the jiffy clock.

```
5 REM          SET CLOCK
6 REM  TO SEE CLOCK  USE VIEW CLOCK 1
7 REM  OR VIEW CLOCK 2
50 :
1000 PRINT "PLEASE INPUT CURRENT TIME (HH/MM/SS)"
1010 INPUT T$
1020 REM        THE LENGTH SHOULD BE 8
1030 IF LEN(T$)<>8 THEN 1000
1040 H$ = LEFT$(T$,2) :   REM  GET HH
1050 M$ = MID$(T$,4,2) :  REM  GET MM
1060 S$ = RIGHT$(T$,2) :  REM  GET SS
1070 TI$ = H$ + M$ + S$ : REM SET CLOCK
```

SET.RUN Clock

This sets and displays your clock.

```
5 REM          SET.RUN CLOCK
6 REM  THIS IS SET CLOCK WITH LINES
7 REM  2998-3080 ADDED
```

```
8 REM  THESE NEW LINES ARE VIEW CLOCK 2
9 REM  RENUMBERED
50 :
1000 PRINT "PLEASE INPUT CURRENT TIME (HH/MM/SS)"
1010 INPUT T$
1020 REM        THE LENGTH SHOULD BE 8
1030 IF LEN(T$)<>8 THEN 1000
1040 H$ = LEFT$(T$,2) :    REM  GET HH
1050 M$ = MID$(T$,4,2) :   REM  GET MM
1060 S$ = RIGHT$(T$,2) :   REM  GET SS
1070 TI$ = H$ + M$ + S$ : REM  SET CLOCK
2998 :
2999 REM        THIS IS VIEW CLOCK 2
3000 PRINT CHR$(147):      REM  CLEAR SCREEN
3010 T$ = TI$ :            REM  GET TIME
3020                       REM  PRINT TIME
3030 PRINT LEFT$(T$,2);":";MID$(T$,3,2);
3040 PRINT " CDT";
3050 FOR I = 1 TO  50:     REM  DELAY LOOP
3060 NEXT
3070 PRINT CHR$(145):      REM  CURSOR UP
3080 GOTO 3000
```

View Clock 1

This clock displays the hour, minutes, and seconds.

```
5 REM           VIEW CLOCK 1
8 REM  TO SET CLOCK, USE SET CLOCK
1000 PRINT CHR$(147):      REM  CLEAR SCREEN
1010 T$ = TI$ :            REM  GET TIME
1020                       REM  PRINT TIME
1030 PRINT LEFT$(T$,2);":";MID$(T$,3,2);":";RIGHT$(T$,2)
1040 FOR I = 1 TO  50:     REM  DELAY LOOP
1050 NEXT
1060 PRINT CHR$(145):      REM  CURSOR UP
1070 GOTO 1000
```

View Clock 2

This clock displays the hour and minutes.

```
5 REM           VIEW CLOCK 2
6 REM  THIS IS VIEW CLOCK 1 WITH
7 REM  LINE 1030 CHANGED
8 REM  TO SET CLOCK, USE SET CLOCK
1000 PRINT CHR$(147):      REM  CLEAR SCREEN
1010 T$ = TI$ :            REM  GET TIME
1020                       REM  PRINT TIME
1030 PRINT LEFT$(T$,2);":";MID$(T$,3,2);
1040 PRINT " CDT";
```

```
1050 FOR I = 1 TO  50:     REM  DELAY LOOP
1060 NEXT
1070 PRINT CHR$(145):      REM  CURSOR UP
1080 GOTO 1000
```

World Clock

This is our final version of the hour and minute clock. It displays the time from two different time zones. The program is currently set up for the Central and Pacific time zones. Since Pacific time is two hours behind Central time, 200 is subtracted in line 1060. To convert to other time zones, you should change lines 1040, 1060, and 1120. A sample conversion:

London time = 11:00 GMT
New York time = 6:00 EST

To set up the WORLD CLOCK for this conversion, type in:

```
1040 PRINT " EST"
1060 T= INT(VAL(T$)/100) + 500
1120 PRINT " GMT"
```

The reason you should add 500 in line 1060 is that London time is five hours ahead of New York time.

```
5 REM           WORLD CLOCK
6 REM  THIS IS VIEW CLOCK 2 WITH 1050 TO 1080 CHANGED
7 REM  AND 1090 TO 1170 ADDED
8 REM  TO SET CLOCK, USE SET CLOCK
9 :
10 REM  TO SET A DIFFERENT TIME ZONE,
11 REM  CHANGE LINES 2050 AND 2100
18 REM  TO SET CLOCK, USE SET CLOCK
1000 PRINT CHR$(147):      REM  CLEAR SCREEN
1010 T$ = TI$ :            REM  GET TIME
1020 REM        PRINT TIME
1030 PRINT LEFT$(T$,2);":";MID$(T$,3,2);
1040 PRINT " CDT"
1050 REM        T$ IS A 6 CHARACTER STRING
1060 T = INT(VAL(T$)/100) - 200
1070 REM        CORRECTS FOR 24 HOUR SYSTEM
1080 IF T<0     THEN T = T+2400
1090 IF T>2359 THEN T = T-2400
1100 T$ = STR$(T)
1105 T$ = RIGHT$(T$,4)
1110 PRINT LEFT$(T$,2);":";MID$(T$,3,2);
1120 PRINT " PDT"
```

```
1130                          REM  DELAY LOOP
1140 FOR I = 1 TO 50
1150 NEXT
1160 PRINT CHR$(19) :         REM  HOME CURSOR
1170 GOTO 1000
```

The Time Of Day (TOD) Clock

Set TOD

This program sets and starts one of your TOD clocks. If you played with the jiffy clock, you should realize how inaccurate it can be when you use tape or disk commands. The TOD clock should be very accurate, because it uses your 60-cycle power line to time itself. There are two TOD clocks inside your machine. To use the other clock, change line 1000 in any of the following programs to:

```
1000 TD = 56328:     REM THE OTHER TOD
```

```
5 REM          SET TOD
6 REM  TOD IS ONE OF THE TWO TIME OF
7 REM  DAY  CLOCKS
80 :
90 REM  LINE 100 CONVERTS X FROM
95 REM  DECIMAL TO PACKED BCD
100 DEF FNB(X) = 16*INT(X/10) + (X - 10*INT(X/10))
110 :
1000 TD = 56584  :          REM  TOD REGISTERS
1010 PRINT CHR$(147):       REM  CLEAR SCREEN
1050 INPUT "INPUT TIME (HH/MM/SS)";T$
1060 IF LEN(T$)<>8 THEN 1050
1070 REM        EXTRACT HOURS
1080 H = VAL(LEFT$(T$,2))
1090 POKE TD+3,FNB(H)
1100 REM        EXTRACT MINUTES
1110 M = VAL(MID$(T$,4,2))
1120 POKE TD+2,FNB(M)
1130 REM        EXTRACT SECONDS
1140 S = VAL(RIGHT$(T$,2))
1150 POKE TD+1,FNB(S)
1160 REM        SET TENTHS OF SECONDS TO ZERO
1170 REM        YOU MUST SET THE TENTHS OF
1180 REM        SECONDS OR THE TOD CLOCK
1190 REM        WILL NOT START
1200 POKE TD,0
```

View TOD 1

This displays the TOD clock with hour, minutes, and seconds.

```
5 REM       VIEW TOD 1
11 :
12 REM    WILL NOT WORK UNTIL TOD CLOCK
13 REM    IS STARTED BY BEING SET
14 :
15 REM    SET IT WITH    SET TOD
1000 TD = 56584
1010 H = PEEK(TD+3)
1020 H$ = CHR$(48+(HAND16)) + CHR$(48+(HAND15))
1030 M = PEEK(TD+2)
1040 M$ = CHR$(48+(MAND112)/16) + CHR$(48+(MAND15))
1050 S = PEEK(TD+1)
1060 S$ = CHR$(48+(SAND112)/16) + CHR$(48+(SAND15))
1070 TS = PEEK(TD)
1190 PRINT CHR$(147) :     REM  CLEAR SCREEN
1200 PRINT H$;":";M$;":";S$;
1290 PRINT CHR$(145);:     REM  UP CURSOR
1300 GOTO 1010
```

View TOD 2

This is a different program for displaying the TOD clock. The use of functions in lines 100 and 110 have simplified the later program statements.

```
5 REM       VIEW TOD 2
6 REM   THIS IS VIEW TOD 1 WITH LINES
7 REM   110-110  ADDED  AND
8 REM   LINES 1020, 1040, AND 1060
9 REM   CHANGED
11 :
12 REM     WILL NOT WORK UNTIL TOD CLOCK
13 REM     IS STARTED BY BEING SET.
15 REM     SET IT WITH  SET TOD
100 DEF FNT(X) = 48 + (X AND 112)/16
110 DEF FNU(X) = 48 + (X AND 15)
1000 TD = 56584
1010 H = PEEK(TD+3)
1020 H$ = CHR$(FNT(H)) + CHR$(FNU(H))
1030 M = PEEK(TD+2)
1040 M$ = CHR$(FNT(M)) + CHR$(FNU(M))
1050 S = PEEK(TD+1)
1060 S$ = CHR$(FNT(S)) + CHR$(FNU(S))
1070 TS = PEEK(TD)
1190 PRINT CHR$(147) :     REM  CLEAR SCREEN
1200 PRINT H$;":";M$;":";S$;
1290 PRINT CHR$(145); :    REM  UP CURSOR
1300 GOTO 1010
```

Set TOD 1

This program sets your TOD clock with an A.M. and P.M. indicator.

```
5 REM          SET TOD 1
6 REM  THIS IS SET TOD  WITH LINES
7 REM  1020-1040 ADDED  AND
8 REM  LINE 1090 CHANGED
80 :
90 REM          LINE 100 CONVERTS X FROM
95 REM          DECIMAL TO PACKED BCD
100 DEF FNB(X) = 16*INT(X/10) + (X - 10*INT(X/10))
110 :
1000 TD = 56584  :          REM  TOD REGISTERS
1010 PRINT CHR$(147):       REM  CLEAR SCREEN
1020 INPUT "AM OR PM";A$
1030 PM = 0
1040 IF LEFT$(A$,1) = "P" THEN PM=1
1050 INPUT "INPUT TIME (HH/MM/SS)";T$
1060 IF LEN(T$)<>8 THEN 1050
1070 REM        EXTRACT HOURS
1080 H = VAL(LEFT$(T$,2))
1090 POKE TD+3,FNB(H) OR 128*PM
1100 REM        EXTRACT MINUTES
1110 M = VAL(MID$(T$,4,2))
1120 POKE TD+2,FNB(M)
1130 REM        EXTRACT SECONDS
1140 S = VAL(RIGHT$(T$,2))
1150 POKE TD+1,FNB(S)
1160 REM        SET TENTHS OF SECONDS TO ZERO
1170 REM        YOU MUST SET THE TENTHS OF
1180 REM        SECONDS OR THE TOD CLOCK
1190 REM        WILL NOT START
1200 POKE TD,0
```

View TOD 3

This displays the TOD clock. Try translating this into a 24-hour clock. Instead of printing P.M., add 12 to the hour.

```
5 REM       VIEW TOD 3
6 REM  THIS IS VIEW TOD 2 WITH LINES
7 REM  1210-1230 ADDED
11 :
12 REM     WILL NOT WORK UNTIL TOD CLOCK
13 REM     IS STARTED BY BEING SET
14 :
15 REM     SET IT WITH   SET TOD
100 DEF FNT(X) = 48 + (X AND 112)/16
110 DEF FNU(X) = 48 + (X AND 15)
1000 TD = 56584
```

```
1010 H = PEEK(TD+3)
1020 H$ = CHR$(FNT(H)) + CHR$(FNU(H))
1030 M = PEEK(TD+2)
1040 M$ = CHR$(FNT(M)) + CHR$(FNU(M))
1050 S = PEEK(TD+1)
1060 S$ = CHR$(FNT(S)) + CHR$(FNU(S))
1070 TS = PEEK(TD)
1190 PRINT CHR$(147) :     REM  CLEAR SCREEN
1200 PRINT H$;":";M$;":";S$;
1210 PRINT SPC(2);
1220 IF H>127 THEN PRINT "PM"
1230 IF H<128 THEN PRINT "AM"
1290 PRINT CHR$(145);:     REM  UP CURSOR
1300 GOTO 1010
```

Challenges

1. Make a FOR–NEXT loop in HOW LONG. Add statements that will show how much time is spent executing each statement of the program. How expensive are REM statements in terms of time? Does rewriting a program to have multiple statements on a single line really speed up a program? How much? What difference is there between using a constant (for example, 60 for 60 jiffies in a second) and using a variable set equal to that constant (JI=60):
2. Write a program to set both the jiffy timer and the TOD clock to the same time. Leave them running, and after LOADing and SAVEing some programs, calculate the difference to show how much time was spent using the disk or tape.
3. Take any of the clocks and turn it into an alarm clock. Display a reminder when the alarm goes off.
4. Create a different type of clock display (for example, a sundial, a water clock, or an hourglass).
5. Write a program that would display each time zone across the United States one at a time.

5

TICKER TAPE

Programs:

VTAB 1	This program positions the cursor at the beginning of any specified line.
CENTER TEXT	This program shows how to center text display.
DISPLAY MESSAGE	A message is scrolled across the screen in the manner of a ticker tape.
MESSAGE BOARD	A sequence of messages can be displayed on your ticker tape.
BOARD WITH CLOCK	A time display is added to the MESSAGE BOARD.
CENTERED CONE	This program displays text in an ice-cream cone.
SET CLOCK	This program sets the time on your jiffy clock, which is used in BOARD WITH CLOCK.
VTAB 2	This program positions the cursor at the beginning of any specified line; this is a better method than VTAB 1.

This collection of programs will allow you to create a tickertape-like message board. BOARD WITH CLOCK, the final version of this program, will do the following:

1. Display and center text on any line
2. Scroll messages like a ticker tape
3. Present a sequence of messages
4. Display the time along with the message.

BASIC Commands Used in This Chapter

CHR$	MID$
DIM	NEXT
END	PRINT
FOR	REM
GOSUB	RETURN
GOTO	RIGHT$
LEFT$	SPC
LEN	

Programming Techniques Used in This Chapter

1. *Tabbing vertically.* Tab vertically by using a PRINT CHR$(17) a number of times. CHR$(17) is the same as the cursor down (more commonly known as a line feed). See VTAB 1, lines 3010 to 3030, or, more elegantly, line 190 in VTAB 2.
2. *Centering text on a given line.* This technique uses LEN(A$) to find the length of A$ and then adds spaces to the beginning of the line with the SPC function. For example, if we want to center 14 characters on a 40-character line, then we should have 13 spaces (because 13 is (40−14)/2) before the beginning of the 14 characters—as any good typist knows. See line 1200 in CENTER TEXT.
3. *Windowing text on the screen.* It appears as if text is being viewed through a window. Actually, what is done is some spaces are printed (as in centering) and then the text. The trick is to make the text appear to be moving. This is done by printing a part of the text and then printing it again but starting one more character to the right. For example, we print (centered) characters 1 to 7, then 2 to 8, 3 to 9, and so on, which gives the appearance of movement. If the text is "Hello there," then what you would see is

```
H e l l o  t
e l l o  t h
```

```
l l o  t h e
l o  t h e r
o  t h e r e
 t h e r e
t h e r e  H
```

but printed on the same line in the same place. In order to complete the illusion, periods are added to each end of the character string so that the beginning and end look alike and to ensure that the message is long enough for the window; otherwise, it looks like a smaller window. In DISPLAY MESSAGE, lines 1120 to 1140 put periods on the text string to be displayed while line 1170 picks out a piece to be displayed, and line 1200 prints the piece in the center.

VTAB 1

This program positions the cursor at the beginning of any specified line. The variable C controls the line that you print on. The message in line 50 can be changed. Try making a message with color, and the reverse. You can also add screen graphics to your messages. Remember that to change the color, you must retype the whole message beginning with the quotation marks. The 64 must have a beginning quotation mark before it will understand a CNTL–color. End the program by hitting the RUN/STOP key.

```
5 REM          VTAB 1
12 :
29 REM    SEE IF YOU CAN GET LOWER CASE
30 REM    DISPLAY
50 C$ = "BE PREPARED"
55 REM          LINE 60 STARTS PRINT ON LINE 6
60 C = 5
99 :
510 PRINT CHR$(147):        REM  CLEAR SCREEN
999 :
1110 GOSUB 3000 :           REM  VERTICAL TAB
1200 PRINT C$
1510 GOTO 1510   :          REM REPEAT AGAIN
1520 END
2997 :
2998 REM        LINES 3000-3040 CAUSE PRINTING
2999 REM        TO START ON LINE C + 1
3000 PRINT CHR$(19);:       REM HOME CURSOR
3010 FOR I = 1 TO C
3020 PRINT CHR$(17);:       REM DOWN CURSOR
3030 NEXT
3040 RETURN
```

Center Text

This program shows how to center text display. Try changing line 1200 (for example, replace 40 with 80 to center printed lines on an 80-column printer). End the program by hitting RUN/STOP.

```
5 REM          CENTER TEXT
6 REM     THIS IS VTAB 1   WITH
7 REM     LINE 60 REMOVED, LINE 1190
8 REM     ADDED AND LINES 1200 AND 3010
9 REM     CHANGED
12 :
23 REM    USE THE COLOR TECHNIQUE IN
24 REM    THE COLOR TYPEWRITER PROGRAM TO
25 REM    GET A RANDOM COLOR DISPLAY
28 :
29 REM    SEE IF YOU CAN GET LOWER CASE
30 REM    DISPLAY
50 C$ = "64 COMMODORE  64"
99 :
510 PRINT CHR$(147):          REM  CLEAR SCREEN
998 :
999 :
1110 GOSUB 3000 :             REM  VERTICAL TAB
1117 :
1190 REM        LINE 1200 PRINTS C$ IN CENTER
1200 PRINT SPC((40-LEN(C$))/2);C$
1510 GOTO 1510  :             REM REPEAT AGAIN
1520 END
2997 :
2998 REM        LINES 3000-3040 CAUSE PRINTING
2999 REM        TO START ON LINE 12
3000 PRINT CHR$(19);:         REM HOME CURSOR
3010 FOR I = 1 TO 11
3020 PRINT CHR$(17);:         REM DOWN CURSOR
3030 NEXT
3040 RETURN
```

Display Message

A message is scrolled across the screen in the manner of a ticker tape. You can change line 500 to vary the length of ticker tape display. You can change lines 1130 and 1150 to vary the filler character being displayed. The speed of the ticker tape can be changed by modifying the length of the wait loop in line 1230. End the program by hitting RUN/STOP and then type PRINT B$ to see the message in full.

```
5 REM          DISPLAY MESSAGE
6 REM     THIS IS  CENTER TEXT WITH
7 REM     LINES 50 AND 1510 CHANGED AND
```

```
8 REM     LINES 198-500, 1118-1180, AND
9 REM     1210-1490  ADDED
10 REM    PLAY WITH DL IN LINE 500
12 :
23 REM    USE THE COLOR TECHNIQUE IN
24 REM    THE COLOR TYPEWRITER PROGRAM TO
25 REM    GET A RANDOM COLOR DISPLAY
28 :
29 REM    SEE IF YOU CAN GET LOWER CASE
30 REM    DISPLAY
50 B$ = "JUST CHECKING ON YOU"
199 :
200 PRINT CHR$(147) :       REM CLEAR SCREEN
490 :
498 REM         DL IS NUMBER OF LETTERS
499 REM         THAT ARE DISPLAYED AT A TIME
500 DL =15
510 PRINT CHR$(147):        REM  CLEAR SCREEN
999 :
1110 GOSUB 3000 :           REM  VERTICAL TAB
1117 :
1118 REM        LINES 1120-1150 ADD DL PERIODS
1119 REM        BEFORE B$ AND DL+1 PERIODS AFTER
1120 FOR I = 1 TO DL
1130 B$ = CHR$(46) + B$ + CHR$(46)
1140 NEXT I :
1150 B$ = B$ + CHR$(46)
1151 :
1158 REM        LINES 1160-1250 CAUSE
1159 REM        MESSAGE IN B$ TO BE DISPLAYED
1160 FOR J = 1 TO LEN(B$) - DL
1170 C$=MID$(B$,J,DL)
1180 :
1190 REM        LINE 1200 PRINTS C$ IN CENTER
1200 PRINT SPC((40-LEN(C$))/2);C$
1210 :
1230 FOR I = 1 TO 100 : NEXT :REM DELAY
1240 PRINT CHR$(145);:      REM  UP CURSOR
1250 NEXT J :
1490 :
1510 GOTO 1160   :          REM REPEAT AGAIN
1520 END
2997 :
2998 REM        LINES 3000-3040 CAUSE PRINTING
2999 REM        TO START ON LINE 12
3000 PRINT CHR$(19);:       REM HOME CURSOR
3010 FOR I = 1 TO 11
3020 PRINT CHR$(17);:       REM DOWN CURSOR
3030 NEXT
3040 RETURN
```

Message Board

A sequence of messages can be displayed on your ticker tape. You can change the messages, starting in line 141. Additional messages can be added by changing NM in line 130 and adding a line 145. For example,

```
130 NM=5: DIM ME$(NM)
145 ME$(5) = "Your new message"
```

See DISPLAY MESSAGE for other suggested changes.

```
5 REM          MESSAGE BOARD
6 REM     THIS IS DISPLAY MESSAGE WITH
7 REM     LINE 50 REMOVED, LINE 1510
8 REM     CHANGED AND LINES 100-144, 1260,
9 REM     AND 1000-1100  ADDED
10 REM    PLAY WITH DL IN LINE 500
12 :
13 REM    ADD YOUR MESSAGES IN LINES 141
14 REM    TO LINE 195
15 :
16 REM    CHANGE THE 25 MESSAGE LIMIT
17 REM    IN LINE 110
19 REM    TRY CHANGING THE BACKGROUND
20 REM    AND DISPLAY COLORS
22 :
23 REM    USE THE COLOR TECHNIQUE IN
24 REM    THE COLOR TYPEWRITER PROGRAM TO
25 REM    GET A RANDOM COLOR DISPLAY
28 :
29 REM     SEE IF YOU CAN GET LOWER CASE
30 REM     DISPLAY
99 :
119 :
120 REM          NUMBER OF MESSAGES  TO DISPLAY
130 NM = 4 : DIM ME$(NM)
138 :
139 :
140 REM          PUT YOUR MESSAGES HERE
141 ME$(1) = "THIS IS A GREAT BOOK"
142 ME$(2) = "TRY OUR GARDEN SHOP"
143 ME$(3) = "OUR BOOKSTORE HAS COMPUTER BOOKS"
144 ME$(4) = "COMMODORE 64'S NOW BEING GIVEN AWAY"
198 :
199 :
498 REM          DL IS NUMBER OF LETTERS
499 REM          THAT ARE DISPLAYED AT A TIME
500 DL =15
510 PRINT CHR$(147):        REM  CLEAR SCREEN
998 :
999 :
```

```
1000 REM        DISPLAY THE NM MESSAGES
1010 REM        LINES 1020-1260 DO THIS
1020 FOR M = 1 TO NM
1100 B$ = ME$(M) :          REM GET MESSAGE M
1110 GOSUB 3000 :           REM  VERTICAL TAB
1117 :
1118 REM        LINES 1120-1150 ADD DL PERIODS
1119 REM        BEFORE B$ AND DL+1 AFTER
1120 FOR I = 1 TO DL
1130 B$ = CHR$(46) + B$ + CHR$(46)
1140 NEXT I :
1150 B$ = B$ + CHR$(46)
1151 :
1158 REM        LINES 1160-1250 CAUSE
1159 REM        MESSAGE IN B$ TO BE DISPLAYED
1160 FOR J = 1 TO LEN(B$) - DL
1170 C$=MID$(B$,J,DL)
1180 :
1190 REM        LINE 1200 PRINTS C$ IN CENTER
1200 PRINT SPC((40-LEN(C$))/2);C$
1210 :
1230 FOR I = 1 TO 100 : NEXT : REM DELAY
1240 PRINT CHR$(145);:      REM  UP CURSOR
1250 NEXT J :
1260 NEXT M :
1490 :
1510 GOTO 1020   :          REM REPEAT AGAIN
2997 :
2998 REM        LINES 3000-3040 CAUSE PRINTING
2999 REM        TO START ON LINE 12
3000 PRINT CHR$(19);:       REM HOME CURSOR
3010 FOR I = 1 TO 11
3020 PRINT CHR$(17);:       REM DOWN CURSOR
3030 NEXT
3040 RETURN
```

Board With Clock

A time display is added to the MESSAGE BOARD. Use the program SET CLOCK to put the clock at the correct time.

```
5 REM         BOARD WITH CLOCK
6 REM     THIS IS MESSAGE BOARD
7 REM     WITH LINES 520, 1500 AND
8 REM     LINES 9998-10040 ADDED
9 REM     SET THE CLOCK WITH SET CLOCK
12 :
13 REM    ADD YOUR MESSAGES IN LINES 141
14 REM    TO LINE 195
15 :
```

```
19 REM    TRY CHANGING THE BACKGROUND
20 REM    AND DISPLAY COLORS
22 :
23 REM    USE THE NEW COLOR TECHNIQUE IN
24 REM    THE COLOR TYPEWRITER PROGRAM TO
25 REM    GET A RANDOM COLOR DISPLAY
28 :
29 REM    SEE IF YOU CAN GET LOWER CASE
30 REM    DISPLAY
99 :
100 REM         AT MOST, YOU CAN USE 25 MESSAGES
119 :
120 REM         NUMBER OF MESSAGES  TO DISPLAY
130 NM = 4: DIM  ME$(NM)
138 :
139 :
140 REM         PUT YOUR MESSAGES HERE
141 ME$(1) = "THIS IS A GREAT BOOK"
142 ME$(2) = "TRY OUR GARDEN SHOP"
143 ME$(3) = "OUR BOOKSTORE HAS COMPUTER BOOKS"
144 ME$(4) = "COMMODORE 64'S NOW BEING GIVEN AWAY"
198 :
199 :
498 REM         DL IS NUMBER OF LETTERS
499 REM         THAT ARE DISPLAYED AT A TIME
500 DL =15
510 PRINT CHR$(147):        REM  CLEAR SCREEN
520 GOSUB 10000 :           REM  DISPLAY TIME
998 :
999 :
1000 REM        DISPLAY THE NM MESSAGES
1010 REM        LINES 1020-1260 DO THIS
1020 FOR M = 1 TO NM
1100 B$ = ME$(M) :          REM GET MESSAGE M
1110 GOSUB 3000 :           REM  VERTICAL TAB
1117 :
1118 REM        LINES 1120-1150 ADD DL SPACES
1119 REM        BEFORE B$ AND DL+1 SPACES AFTER
1120 FOR I = 1 TO DL
1130 B$ = CHR$(46) + B$ + CHR$(46)
1140 NEXT I :
1150 B$ = B$ + CHR$(46)
1151 :
1158 REM        LINES 1160-1250 CAUSE
1159 REM        MESSAGE IN B$ TO BE DISPLAYED
1160 FOR J = 1 TO LEN(B$) - DL
1170 C$=MID$(B$,J,DL)
1180 :
1190 REM        LINE 1200 PRINTS C$ IN CENTER
1200 PRINT SPC((40-LEN(C$))/2);C$
1210 :
1230 FOR I = 1 TO 100 : NEXT : REM DELAY
```

```
1240 PRINT CHR$(145);:      REM  UP CURSOR
1250 NEXT J :
1260 NEXT M :
1490 :
1500 GOSUB 10000 :          REM DISPLAY TIME
1510 GOTO 1020   :          REM REPEAT AGAIN
2997 :
2998 REM        LINES 3000-3040 CAUSE PRINTING
2999 REM        TO START ON LINE 12
3000 PRINT CHR$(19);:       REM HOME CURSOR
3010 FOR I = 1 TO 11
3020 PRINT CHR$(17);:       REM DOWN CURSOR
3030 NEXT
3040 RETURN
9998 :
9999 REM            SIMPLE VIEW CLOCK ROUTINE
10000 PRINT CHR$(19):       REM  HOME CURSOR
10010 T$ = TI$ :            REM  GET TIME
10020                       REM PRINT TIME
10030 PRINT LEFT$(T$,2);":";MID$(T$,3,2)
10040 RETURN
```

Centered Cone

This program displays text in an ice-cream cone. You can change lines 1310 to 1340 to obtain a Christmas tree. Other shapes are left to your imagination.

```
5 REM          CENTERED CONE
6 REM    THIS IS CENTER TEXT WITH LINES
7 REM    1300-1340 ADDED AND 1510
8 REM    CHANGED
12 :
50 C$ = "64 COMMODORE 64"
99 :
510 PRINT CHR$(147):        REM  CLEAR SCREEN
998 :
999 :
1110 GOSUB 3000 :           REM  VERTICAL TAB
1117 :
1190 REM        LINE 1200 PRINTS C$ IN CENTER
1200 PRINT SPC((40-LEN(C$))/2);C$
1300 REM        GET LENGTH OF B$
1310 B = LEN(C$)
1320 IF B=0 THEN 1320 :     REM  LOOP FOREVER
1330 REM        DROP LAST LETTER FROM C$
1340 C$ = LEFT$(C$,B-1)
1510 GOTO 1200   :          REM REPEAT AGAIN
1520 END
2997 :
2998 REM        LINES 3000-3040 CAUSE PRINTING
2999 REM        TO START ON LINE 12
```

```
3000 PRINT CHR$(19);:       REM HOME CURSOR
3010 FOR I = 1 TO 11
3020 PRINT CHR$(17);:       REM DOWN CURSOR
3030 NEXT
3040 RETURN
```

Set Clock

This program sets the time on your jiffy clock, which is used in BOARD WITH CLOCK. Other programs using the clock are in Chapter 4, "What Time Is It?"

```
5 REM          SET CLOCK
6 REM  TO SEE CLOCK  USE VIEW CLOCK 1
7 REM  OR VIEW CLOCK 2
50 :
1000 PRINT "PLEASE INPUT CURRENT TIME (HH/MM/SS)"
1010 INPUT T$
1020 REM        THE LENGTH SHOULD BE 8
1030 IF LEN(T$)<>8 THEN 1000
1040 H$ = LEFT$(T$,2) :    REM  GET HH
1050 M$ = MID$(T$,4,2) :   REM  GET MM
1060 S$ = RIGHT$(T$,2) :   REM  GET SS
1070 TI$ = H$ + M$ + S$ : REM SET CLOCK
```

VTAB 2

This program positions the cursor to the beginning of any specified line. This is a faster method than VTAB 1. It is also easier to figure out when reading through a program. To add this to your program, use lines 100–130. To vertically tab to a line, such as line 17, use the command:

```
PRINT LEFT$(VT$,17);
```

```
5 REM          VTAB 2
70 :
80 PRINT CHR$(147) :        REM CLEAR SCREEN
90 :
97  REM        LINES 100-130 BUILDS VT$
98  REM        WHICH THEN ACTS LIKE A VTAB
99  REM        AS USED IN LINE 190
100 VT$ = CHR$(19) :        REM  HOME CURSOR
110 FOR I = 1 TO 24
120 VT$ = VT$ + CHR$(17):  REM  DOWN CURSOR
130 NEXT
140 PRINT "TO WHAT LINE";
150 INPUT Y
160 REM        Y MUST BE BETWEEN 0 AND 24
```

```
170 IF Y>24 THEN Y=24
180 IF Y<0  THEN Y=0
190 PRINT LEFT$(VT$,Y);:   REM THIS DOES IT
200 GOTO140
```

Challenges

1. Add a routine to MESSAGE BOARD or BOARD WITH CLOCK that will print the time, date, and current messages whenever some key is pressed.
2. Add color to the messages.
3. Play some music while the message scrolls.
4. Add a routine that would use the clock to vary the messages that are displayed.
5. Write a routine that will use your joystick or keyboard to control the speed of the message displayed.
6. Make a game that will use the ticker tape routine.
7. Change MESSAGE BOARD to allow the entry and deletion of messages while the program is still displaying messages.
8. Make MESSAGE BOARD read messages from a data file and display them as ticker tape.

6

ODDS AND ENDS

Programs:

New art:

WEIRD	Try it, you'll like it.
WEIRDER	This program generates a spectacular color display.
WEIRDER 1	This is another "biggie" color displayer.

Here we go . . . Loop the loop:

VISIBLE LOOPING	This is easier on the eyes than invisible looping.
WHAT'S YOUR NAME	This program prints your name in a pattern on the screen.
NAME LENGTH 1	This program prints the number of characters in your name using an IF–THEN loop.
NAME LENGTH 2	This program prints the number of characters in your name using a FOR–NEXT loop.
BIGGEST NUMBER	This program will stop at the biggest number.

STRINGS AND THINGS:

NAME LENGTH 3	This program prints the number of characters in your name using a LENgth function.
STRINGS	This routine does just about all you can do with strings of characters.

Fun and dumb things to do:

VALENTINE	Won't you be . . .
PUPPY CHOW	This is an unpaid commercial advertisement.
BUY ME	This program shows what to do with this book.
BUY ME 1	This program shows the power of a single semicolon.
PIZZA	Pi are square, but pizza are round.
LIST MAKER	This converts your $500 computer into a ten-cent paper tablet.
PEEKABOO	UN this program.
MARCHING TEXT	Your text will march across the screen.
DISPLAY CHARS	You can now see how each character displays on the screen.

Our Own Oddities™:

SYSTEM KILLER	Now you see it, now you don't.
BROKEN SIGNAL	This program fixes the wire from your 64 to your TV set.
UNCERTAIN CASE	Dr. Watson, I presume?
FLASHER	This routine will flash any text on the screen.
SLOW FLASHER	This routine slows down FLASHER.
REVERSE TEXT	This program prints anything you want in reverse.
STRANGE NO?	This program self-destructs.

Skill builders:

TIMED SCRAMBLER	This program reads a word from a data statement and times you while you unscramble it.

MATCH MAKER	This is a simple matching program.
MULTIPLE CHOICE	This is a simple multiple-choice program.
CIPHER	This program is a Junior CIA Operator's Manual.

In this chapter we give you a bunch of odds and ends, programs that are useful and programs that are simply odd (in fact, weird). Most of the odd ones use the special video capabilities of the 64, but we're not going to provide any explanation of them. Simply enjoy what is here.

New Art

Weird

Try it, you'll like it.

```
5 REM          WEIRD
6 REM    53265 NORMALLY CONTAINS 27
7 REM    TRY THIS BY TYPING:
8 REM    PRINT PEEK(53265)
9 REM    TRY USING SOMETHING OTHER
10 REM    THAN 59 IN LINE 110
90 :
100 REM          SCREEN DISPLAY CONTROL
110 POKE 53265,59
120 FOR I = 1 TO 2000 :  REM DELAY LOOP
130 NEXT I
140 POKE 53265,27 :       REM  BACK TO NORMAL
```

Weirder

This program generates a spectacular color display.

```
5 REM          WEIRDER
100 REM          LOOP THROUGH ALL POSSIBLE
110 REM          BACKGROUND COMBINATIONS
120 FOR J = 0 TO 255
130 REM          SCREEN DISPLAY CONTROL
140 POKE 53265,59
150 REM          POKE STUFF INTO TEXT SCREEN
160 FOR I = 1024 TO 2040
170 POKE I,J
180 NEXT I,J
190 POKE 53265,27 :       REM  BACK TO NORMAL
```

Weirder 1

This is another "biggie" color displayer.

```
5 REM          WEIRDER 1
6 REM  THIS IS WEIRDER  WITH LINES
7 REM  100-120 DELETED, AND LINES
8 REM  170 AND 180 CHANGED
130 REM        SCREEN DISPLAY CONTROL
140 POKE 53265,59
150 REM        POKE STUFF INTO TEXT SCREEN
160 FOR I = 1024 TO 2040
170 POKE I,I AND 255
180 NEXT I
190 POKE 53265,27 :       REM  BACK TO NORMAL
```

Here We Go . . . Loop the Loop

Visible Looping

This is easier on the eyes than invisible looping.

```
5 REM           VISIBLE LOOPING
30 :
100 PRINT CHR$(147) :     REM  CLEAR SCREEN
110 C1 = 1 :              REM  COLOR WHITE
120 MAX = 10
130 TE = 1024        :    REM  TEXT PAGE
140 CM = 55296       :    REM  COLOR MAP
150 REM         LINES 150-190    FIX FOR POKES
160 POKE CM+4,  C1
170 POKE CM+44, C1
180 POKE CM+84, C1
190 POKE CM+124,C1
200 FOR I = 1 TO MAX:     REM  LOOPS
210 POKE TE+4,I
220 FOR J = 1 TO MAX
230 POKE TE+44,J
240 FOR K = 1 TO MAX
250 POKE TE+84,K
260 FOR L = 1 TO MAX
270 POKE TE+124,L
280 NEXT L,K,J,I
```

What's Your Name

This program prints your name in a pattern on the screen.

```
5 REM           WHAT'S YOUR NAME
100 PRINT CHR$(147):  REM  CLEAR SCREEN
110 INPUT "WHAT'S YOUR NAME";NA$
120 FOR I = 1 TO 100
130 PRINT TAB(I);NA$
140 NEXT
```

Name Length 1

This program prints the number of characters in your name using an IF–THEN loop.

```
5 REM          NAME LENGTH 1
100 INPUT "WHAT IS YOUR NAME";NA$
110 LE = 0
120 IF MID$(NA$,LE+1,1)="" THEN 900
130 LE = LE + 1
140 GOTO 120
900 PRINT "YOUR NAME, ";NA$;", IS";
910 PRINT LE;"LETTERS LONG."
```

Name Length 2

This program prints the number of characters in your name using a FOR–NEXT loop.

```
5 REM          NAME LENGTH 2
6 REM  THIS IS NAME LENGTH 1 WITH
7 REM  LINES  110 AND 140 CHANGED
8 REM AND LINE 130 DELETED
90 :
100 INPUT "WHAT IS YOUR NAME";NA$
110 FOR LE = 0 TO 500
120 IF MID$(NA$,LE+1,1)="" THEN 900
140 NEXT
900 PRINT "YOUR NAME, ";NA$;", IS";
910 PRINT LE;"LETTERS LONG."
```

Biggest Number

This program will stop at the biggest number—at least, the biggest for the 64. To get there a little faster, change line 120 to:

```
120 I = I + I: GOTO 110
```

```
5 REM          BIGGEST NUMBER
100 I = 1
110 PRINT I
120 I = I + 1
130 GOTO 110
```

Strings and Things . . .

Name Length 3

This program prints the number of characters in your name using a LENgth function.

```
5 REM          NAME LENGTH 3
6 REM  THIS IS NAME LENGTH 2 WITH
7 REM  LINES  120 AND 140 DELETED
8 REM  AND LINE 110 CHANGED
90 :
100 INPUT "WHAT IS YOUR NAME";NA$
110 LE = LEN(NA$)
900 PRINT "YOUR NAME, ";NA$;", IS";
910 PRINT LE;"LETTERS LONG."
```

Strings

This program does just about all you can do with strings of characters.

```
5 REM           STRINGS
90 PRINT CHR$(147)
100 INPUT "TYPE A SHORT STRING";X$
110 FOR J = 1 TO LEN(X$)
120 A$ = MID$(X$,J)
130 R = ASC(A$)
140 PRINT A$,R
150 NEXT J
190 PRINT "HIT ANY KEY"
200 GET B$
210 IF B$="" THEN 200
300 FOR J = 1 TO LEN(X$)
310 PRINT LEFT$(X$,J),RIGHT$(X$,J)
320 NEXT J
```

Fun and Dumb Things to Do

Valentine

This program is just a lot of hearts—all over the screen.

```
5 REM           VALENTINE
6 REM  TRY 88 INSTEAD OF 83 IN LINE 160
7 REM  AND 5 INSTEAD OF 2 IN 170
90 :
100 BG=53280 :            REM  BORDER
110 POKE BG,1 :           REM  BORDER WHITE
120 POKE BG+1,1 :         REM  BACKGROUND WHITE
```

```
130 PRINT CHR$(147) :     REM CLEAR SCREEN
140 REM         POKES THE TEXT SCREEN
150 FOR I= 1024 TO 2023
160 POKE I,83 :           REM   A HEART
170 POKE I+54272,2 :      REM  DISPLAY RED
180 NEXT I
210 GOTO 210
```

Puppy Chow™

This is an unpaid (doggone it) commercial advertisement.

```
5 REM           PUPPY CHOW
90 :
100 BG=53280 :            REM  BORDER
110 POKE BG,1 :           REM  BORDER WHITE
120 POKE BG+1,1 :         REM  BACKGROUND WHITE
130 PRINT CHR$(147) :     REM CLEAR SCREEN
140 REM         POKES SCREEN
150 FOR I= 1 TO 10 :      REM FIRST 10 LINES
160 FOR J= 1 TO 15 :      REM 15 COLUMNS
170 REM         CALCULATES SCREEN LOCATION
180 K = 1024 + 40*(I - 1) + J - 1
190 POKE K,127 :          REM CHECKERBOARD
200 POKE K+54272,2:       REM COLOR TO RED
210 NEXT J,I
220 REM         ENDLESS LOOP,  STOPPED ONLY
230 REM         BY RUN/STOP
240 GOTO 240
```

Buy Me

This program shows what to do with this book.

```
5 REM         BUY ME
1010 I = 1
1020 B$ = "BUY ME"
1030 PRINT SPC(I);B$
1040 I = I + 1
1050 GOTO 1030
```

Buy Me 1

This program shows the power of a single semicolon.

```
5 REM        BUY ME 1
6 REM  THIS IS BUY ME WITH LINE
7 REM  1030 CHANGED
```

```
1010 I = 1
1020 B$ = "BUY ME"
1030 PRINT SPC(I);B$;
1040 I = I + 1
1050 GOTO 1030
```

Pizza

Pi are square, but pizza are round. This tells you how big your pizza is. Add a routine to determine the price per square inch of your pizza.

```
5 REM         PIZZA
100 PRINT CHR$(147) : REM  CLEAR SCREEN
110 REM PIZZA CALCULATOR
120 PRINT "WHAT IS THE DIAMETER"
130 PRINT "OF YOUR PIZZA?"
140 INPUT D
150 PRINT
160 PRINT "O.K. A "D" INCH PIZZA HAS"
170 PRINT *(D/2)↑2" SQ.INCHES."
180 GOTO 120
```

List Maker

This program converts your $500 computer into a ten-cent paper tablet. If you have a printer and want to have a printed list, add the following lines:

```
125 OPEN 4,4,4
170 CLOSE 4
```

and change lines 140 and 150 to:

```
140 PRINT #4,X
150 PRINT #4, "_ _ _ _ _ _ _ _ _ _ _ _"
```

Make sure that you have the comma in those lines, or you will get a SYNTAX error.

```
5 REM              LIST MAKER
6 REM  SEE SCREEN DUMP     FOR PRINTER
7 REM  ROUTINE.
100 INPUT "HOW MANY BLANKS?";N
110 REM         LIMIT # OF LINES ON PRINTER PAGE.
120 IF N>60 THEN N=60
130 FOR X= 1 TO N
140 PRINT X
150 PRINT"_____________________"
160 NEXT
```

Peekaboo

UN this program. After you RUN the program, the pun will become more apparent. Try playing with line 100. You can change both the value of the loop and the STEP. If you cannot get back to normal, just remember to press <RUN/STOP> and <RESTORE> at the same time.

```
5 REM          PEEKABOO
6 REM  TRY CHANGING  THE LOOP LIMITS
7 REM  IN LINE 100
90 :
100 FOR J =  7 TO 0 STEP -1
110 REM         SCREEN DISPLAY CONTROL
120 POKE 53270,J
130 FOR I = 1 TO 200 :    REM DELAY LOOP
140 NEXT I
150 NEXT J
```

Marching Text

Your text will march across the screen. To stop the program, you must press the <RUN/STOP> first and, while holding it down, press the <RESTORE> key.

```
5 REM           MARCHING TEXT
100 FOR J = 8 TO 15
110 REM         SCREEN DISPLAY CONTROL
120 POKE 53270,J
130 FOR I = 1 TO 200 :    REM DELAY LOOP
140 NEXT I
150 NEXT J
160 POKE 53270,8 :        REM  BACK TO NORMAL
170 GOTO 100:             REM  DO IT FOREVER
```

Display Chars

You can now see how each character displays on the screen. The first 31 characters cannot be displayed because they are control characters. You can change the 32 to a 1 in line 200, but watch quickly if you do.

```
5 REM           DISPLAY CHARS
100 PRINT CHR$(147):      REM  CLEAR SCREEN
200 FOR I = 32 TO 255
210 PRINT I,CHR$(I)
220 NEXT
```

Our Own Oddities™

System Killer

Now you see it, now you don't. Possibly you should call line 100 a system reset because that is what it is. In fact, it doesn't matter what you type after line 100.

```
5 REM           SYSTEM KILLER
6 REM  TRY CHANGING  THE LOOP LIMITS
7 REM  IN LINE 130
90 :
100 SYS 64738
110 REM         SCREEN DISPLAY CONTROL
120 POKE 53270,J
130 FOR I = 1 TO 200 :    REM DELAY LOOP
140 NEXT I
150 NEXT J
```

Broken Signal

This program fixes the wire from your 64 to your TV set.

```
5 REM           BROKEN SIGNAL
6 REM  TRY CHANGING THE LOOP LIMITS
7 REM  IN LINE 140
90 :
100 PRINT CHR$(147):      REM CLEAR SCREEN
110 PRINT CHR$( 18);:     REM REVERSE ON KEY
120 PRINT "HERE IS SOME TEXT"
130 FOR I = 1 TO 10 :     REM  REPEAT LOOP
140 FOR J = 0 TO 63
150 REM         SCREEN DISPLAY CONTROL
160 POKE 53270,J
170 NEXT J
180 NEXT I
190 POKE 53270, 8 :       REM  BACK TO NORMAL
```

Uncertain Case

Dr. Watson, I presume?

```
5 REM           UNCERTAIN CASE
50 DE = 5
100 FOR I = 1 TO 200
110 REM         CHARACTER SET CONTROL
120 POKE 53272,23
```

```
130 FOR J = 1 TO DE
140 NEXT J          :          REM  DELAY LOOP
190 POKE 53272,21:             REM  BACK TO NORMAL
200 NEXT I
```

Flasher

This routine will flash any text on the screen. You might add this routine to other programs to make them interesting.

```
5 REM         FLASHER
6 REM  THIS IS REVERSE TEXT  WITH
7 REM  LINES 200 ON ADDED
90 :
100 PRINT CHR$(147):       REM CLEAR SCREEN
110 PRINT CHR$( 18);:      REM REVERSE KEY
120 PRINT "HAPPY BIRTHDAY"
190 :
200 REM          LOOP TO FLASH
210 FOR J = 0 TO 255
220 REM          SCREEN DISPLAY CONTROL
230 POKE 53265,91 :        REM INVERSE IT
250 POKE 53265,27 :        REM BACK TO NORMAL
260 NEXT
```

Slow Flasher

This routine slows down FLASHER.

```
5 REM         SLOW FLASHER
6 REM  THIS IS FLASHER    WITH
7 REM  LINES 50,240  AND 255  ADDED
50 DE = 100
90 :
100 PRINT CHR$(147):       REM CLEAR SCREEN
110 PRINT CHR$( 18);:      REM REVERSE KEY
120 PRINT "HAPPY BIRTHDAY"
190 :
200 REM          LOOP TO FLASH
210 FOR J = 0 TO 255
220 REM          SCREEN DISPLAY CONTROL
230 POKE 53265,91 :        REM INVERSE IT
240 FOR I=1TODE:NEXT :     REM DELAY LOOP
250 POKE 53265,27 :        REM BACK TO NORMAL
255 FOR I=1TODE:NEXT :     REM DELAY LOOP
260 NEXT
```

Reverse Text

This program prints anything you want in reverse.

```
5 REM           REVERSE TEXT
100 PRINT CHR$(147):      REM CLEAR SCREEN
110 PRINT CHR$( 18);:     REM REVERSE KEY
120 PRINT "HAPPY BIRTHDAY"
```

Strange, No?

This program self-destructs. You must type this program exactly as shown. There are four spaces after REM and 1 space before NO?. Save the program before you run it. What happens if you use a different REM?

```
1000 REM    STRANGE, NO?
1010 FOR I = 1 TO 24
1020 POKE 214,I
1030 PRINT "A";
1040 NEXT
1050 GOTO 1000
```

Skill Builders

Timed Scrambler

This program reads scrambled words from DATA statements and times you while you try to unscramble them.

```
5 REM          TIMED SCRAMBLER
6 REM  LINES 990-2040 ARE FROM
7 REM  BINGO CARD, WITH CHANGES TO
8 REM  LINES 1000 AND 1030
29 :
30 REM ARRAY FOR RANDOM SHUFFLE
40 DIM A(20),A$(20)
50 DEF FND(X) = INT(X*RND(0)) + 1
90 :
100 PRINT CHR$(147)  :   REM CLEAR SCREEN
110 REM         READ IN WORDS
120 FOR I = 1 TO 20
130 READ A$(I)
140 NEXT I
142 DATA  BENGALS,STEELERS,CARDINALS
143 DATA  CHARGERS.RAMS,CHIEFS,BEARS
144 DATA  JETS,GIANTS,DOLPHINS,BANDITS
145 DATA  OUTLAWS,STARS,COSMOS,REDS
146 DATA  DODGERS,PADRES,ASTROS,COWBOYS
147 DATA  BLUEJAYS,YANKEES,TWINS,BRAVES
150 :
200 REM         NOW CHOOSE A RANDOM WORD
210 W = FND(20)
220 C$ = A$(W) :          REM  RANDOM WORD
```

```
230 REM         NOW SCRAMBLE IT
240 CL = LEN(C$)
250 GOSUB 1000
260 W$ = "" :               REM  INIT W$
270 FOR J = 1 TO CL
280 GOSUB 2000
290 W$ = W$ + MID$(C$,NU,1)
300 NEXT J
310 :
320 PRINT CHR$(147)   :    REM CLEAR SCREEN
330 PRINT CHR$(17);
340 PRINT SPC(10);W$:      REM  PRINT IT
350 TI$ = "000000" :       REM  START TIMER
360 GOSUB 10000    :       REM  PRINT TIME
370 PRINT "HIT ANY KEY TO TRY A GUESS"
380 GET A$
390 GOSUB 10000
400 IF A$="" THEN 380
410 PRINT CHR$(17);CHR$(17);CHR$(17)
420 INPUT "GUESS";G$
430 IF G$ = C$ THEN 470
440 PRINT "SORRY, NOT YET RIGHT"
450 GOTO 360
460 REM         CORRECT GUESS
470 T$ = TI$
480 PRINT "VERY GOOD, YOU GOT IT"
490 PRINT "IN ";MID$(T$,3,2);" MINUTES";
500 PRINT " AND ";RIGHT$(T$,2);
510 PRINT " SECONDS"
520 PRINT:PRINT
530 PRINT "WANT TO TRY AGAIN ?"
540 GET A$
550 IF A$="Y" THEN 210
560 IF A$="N" THEN END
570 GOTO 540
990 END
997 :
998 REM         INITIALIZE RANDOM GENERATOR
999 REM         SET UP ARRAY
1000 FOR I = 1 TO CL
1010 A(I) = I
1020 NEXT I
1030 I = CL
1040 RETURN
1998 :
1999 REM        RANDOM NUMBER GETTER
2000 B = FND(I)
2010 NU = A(B)
2020 A(B) = A(I)
2030 I = I - 1
2040 RETURN
9998 :
```

```
9999 REM        SIMPLE VIEW CLOCK ROUTINE
10000 PRINT CHR$(19):     REM  HOME CURSOR
10010 T$ = TI$       :    REM  GET TIME
10020                     REM  PRINT TIME
10030 PRINT MID$(T$,3,2);":";RIGHT$(T$,2)
10040 PRINT
10050 RETURN
```

Match Maker

This is a simple matching program.

```
5 REM          MATCH MAKER
6 :
7 REM  TRY CHANGING THE LETTER COLORS
8 REM  USE DIFFERENT BACKGROUND AND
9 REM  BORDER COLORS
10 REM  DON'T ALLOW AN INCORRECT REPLY
11 REM  TO BE TYPED
30 :
80 NU = 10 :              REM  NUMBER OF QUESTIONS
90 :
110 PRINT CHR$(147) :     REM  CLEAR SCREEN
120 T$ = "WELCOME TO MATCH MAKER"
130 L = LEN(T$)
140 GOSUB 710 :           REM  CENTER TEXT
150 GOSUB 740 :           REM  REVERSE TEXT
160 PRINT T$
170 FOR Z = 1 TO 500:NEXT
180 :
190 DIM QUEST$(NU),ANS$(NU)
200 :
210 :
220 FOR I=1 TO NU
230 REM         READ QUESTIONS INTO MEMORY
240 READ QUEST$(I), ANS$(I)
250 NEXT I
260 :
280 :
290 REM         SELECT A RANDOM NUMBER
320 X = INT(RND(0)*NU)+1
330 :
340 PRINT CHR$(147)
350 REM         PRINT ONE DATA QUESTION FROM RANDOM.
360 REM         PLACE YOUR QUESTION IN THE FOLLOWING STRING.
370 T$ = "WHAT IS THE CAPITAL OF "
380 PRINT T$;
390 L = LEN(QUEST$(X))
400 GOSUB 710 :           REM  CENTER TEXT
410 GOSUB 740 :           REM  REVERSE TEXT
420 PRINT QUEST$(X)
```

```
430 PRINT
440 :
450 REM         CHECK REPLY WITH ANSWER.
460 INPUT REPLY$
470 IF REPLY$=ANS$(X) GOTO 580
480 :
490 PRINT "SORRY, THE CORRECT ANSWER"
500 PRINT "IS ";
510 L = LEN(ANS$(X))
520 GOSUB 710 :           REM  CENTER TEXT
530 GOSUB 740 :           REM  REVERSE TEXT
540 PRINT ANS$(X)
550 PRINT
560 GOTO 670
570 :
580 PRINT: PRINT "YOU ARE CORRECT"
590 PRINT "THE ANSWER IS ";
600 L = LEN(ANS$(X))
610 GOSUB 710 :           REM  CENTER TEXT
620 GOSUB 740 :           REM  REVERSE TEXT
630 PRINT ANS$(X)
640 PRINT
650 REM         YOU MAY WANT TO PUT A SUBROUTINE
660 REM         HERE WHICH DISPLAYS GRAPHICS.
670 PRINT "PRESS RETURN TO CONTINUE."
680 INPUT P$
690 GOTO 320
700 REM         CENTER TEXT
710 PRINT TAB((40-L)/2):RETURN
720 RETURN
730 REM         REVERSE TEXT
740 PRINT CHR$(18);  :    REM  REVERSE ON
750 RETURN
1000 REM        QUESTIONS AND ANSWERS ARE
1010 REM        CONTAINED IN THE FOLLOWING
1020 REM        DATA STATEMENTS.  BE SURE
1030 REM        YOU START EACH LINE WITH THE
1040 REM        WORD DATA AND SEPERATE THE
1050 REM        INFORMATION WITH COMMAS.
1060 :
1070 REM        YOU CAN ADD ANY NUMBER OF
1080 REM        QUESTIONS BUT BE SURE
1090 REM        TO CHANGE VARIABLE NU IN
1100 REM        LINE  80
1110 REM        IN OUR EXAMPLE THAT IS THE
1120 REM        THE NUMBER 10.  DO NOT PLACE
1130 REM        A NUMBER IN IT LARGER THAN
1140 REM        THE NUMBER OF DATA STATEMENTS.
1150 :
1160 DATA ALABAMA, MONTGOMERY
1170 DATA ALASKA, JUNEAU
1180 DATA ARIZONA, PHOENIX
```

```
1190 DATA ARKANSAS, LITTLE ROCK
1200 DATA CALIFORNIA,SACRAMENTO
1210 :
1220 DATA TEXAS,AUSTIN
1230 DATA NEW YORK, ALBANY
1240 DATA MISSOURI, JEFFERSON CITY
1250 DATA MINNESOTA, ST. PAUL
1260 DATA PENNSYLVANIA, HARRISBURG
```

Multiple Choice

This is a simple multiple-choice program.

```
5 REM         MULTIPLE CHOICE
6 REM PERSONALIZE THE PROGRAM
7 REM ADD A TIMER
8 :
9 REM CONSTRUCT A COUNTER TO RECORD
10 REM THE NUMBER RIGHT AND WRONG.
11 REM INCLUDE A TOTAL PERCENTAGE.
12 :
13 REM PERSONALIZE THE PROGRAM
14 REM THE USE OF THE USER'S INPUT NAME.
15 :
16 REM WRITE A ROUTINE TO ACCEPT ONLY
17 REM THE LETTERS A - D.
18 :
19 REM WRITE A ROUTINE WHICH WOULD
20 REM "FLASH" THE CORRECT ANSWER.
21 :
22 REM WRITE A GRAPHICS DISPLAY FOR
23 REM A CORRECT ANSWER.
24 :
25 REM WRITE A ROUTINE TO DISPLAY ONLY
26 REM ONE QUESTION ON THE SCREEN
27 REM AT A TIME.
28 :
100 REM         CHANGE BACKGROUND COLOR TO WHITE.
110 POKE 53281,1 :        REM WHITE BACKGROUND
120 REM         HOW ABOUT A TIMER STARTING HERE?
130 PRINT CHR$(147):      REM CLEAR SCREEN
170 PRINT CHR$(144):      REM PRINT IN BLACK
180 REM         ENTER TITLE AS STRING VARIABLE.
190 REM         THE FIRST LETTER IN THE VARIABLE
200 REM         IS A CTRL-9 TO REVERSE THE TEXT.
210 TI$=CHR$(18)+"AMERICAN HISTORY QUIZ"
220 PRINT
230 REM         DO YOU KNOW WHY YOU CANNOT USE
240 REM         A VARIABLE "TITLE" AS A STRING?
250 REM         CHECK YOUR "USER'S GUIDE"
260 REM         FOR THE PREDEFINED VARIABLE TI.
```

```
270 :
280 REM         DETERMINE LENGTH OF TITLE
290 L=LEN(TT$)
300 RESTORE
340 GOSUB 800:PRINT CHR$(18)+TT$
350 FOR X = 1 TO 500: NEXT
360 REM         NU = # OF QUESTIONS IN QUIZ
370 NU = 2
380 PRINT
390 PRINT "THERE ARE "NU" QUESTIONS ON FILE."
400 FOR J = 1 TO NU
410 PRINT
420 READ QUEST$(J)
430 PRINT "QUESTION #";J
440 PRINT
450 :
460 L=LEN(QUEST$(J))
470 GOSUB 800
480 PRINT CHR$(18)+QUEST$(J)
490 :
500 REM         READ THE CHOICES.
510 READ A$,B$,C$,D$
520 PRINT
530 PRINT " SELECT LETTER A,B,C, OR D "
540 PRINT
550 :
560 PRINT "A. ";A$
570 PRINT "B. ";B$
580 PRINT "C. ";C$
590 PRINT "D. ";D$
600 INPUT REPLY$
610 :
620 REM         READ ANSWER FROM DATA STATEMENT.
630 READ ANS$
640 IF ANS$=REPLY$ GOTO 700
650 REM         INVERSE CORRECT ANSWER.
660 PRINT CHR$(18)+"SORRY, THE CORRECT ANSWER IS ";
670 PRINT CHR$(18)+ANS$
680 NEXT J
690 GOSUB 730
700 PRINT "CORRECT. THE ANSWER IS ";
710 PRINT ANS$
720 NEXT J
730 PRINT "DO YOU WANT TO START OVER?"
740 INPUT R$
750 REM         IF RETURN KEY IS HIT START OVER.
760 IF R$="" GOTO 110
770 IF LEFT$(R$,1)="Y" GOTO 110
780 PRINT CHR$(147) :    REM CLEAR SCREEN
790 END
800 PRINT TAB((40-L)/2)
810 RETURN
```

```
820 REM         IF YOU USE THE PRESENT FORMAT,
830 REM         YOU MUST HAVE A QUESTION LINE,
840 REM         4 LINES OF CHOICES, AND A
850 REM         CORRECT ANSWER LINE.
860 REM         USE THE COLON TO SEPARATE EACH
870 REM         GROUP OF QUESTIONS.  WHILE IT IS
880 REM         NOT NECESSARY FOR THE PROGRAM
890 REM         TO WORK, IT MAKES IT EASIER
900 REM         FOR CHANGES AND ERROR CORRECTION.
910 REM         CHANGE LINE 370 TO REFLECT THE
920 REM         TOTAL NUMBER OF QUESTIONS.
930 DATA "WHO IS BURIED IN GRANT'S TOMB?"
940 DATA "THE LONE RANGER"
950 DATA "GROUCHO MARX"
960 DATA "GENERAL GRANT"
970 DATA "RIN-TIN-TIN"
980 DATA "C"
990 :
1000 DATA "WHO IS THE PRESIDENT OF THE U.S.?"
1010 DATA "GROVER CLEVELAND"
1020 DATA "RONALD REAGAN"
1030 DATA "RIN-TIN-TIN"
1040 DATA "ABRAHAM LINCOLN"
1050 DATA "B"
1060 REM        TRY WRITING A TEST ON THE 64.
```

Cipher

This program is a Junior CIA Operator's Manual.

```
5 REM            CIPHER
90 :
100 PRINT CHR$(147) :     REM  CLEAR SCREEN
110 PRINT "TYPE CODE LETTER"
120 GET C$
130 IF C$ = "" THEN 120
140 C = ASC(C$) - 64
150 IF C<1 OR C>26 THEN 100
200 PRINT "CIPHER (C) OR DECIPHER (D)"
210 GET OP$
220 IF OP$="" THEN 210
230 IF OP$<>"C" AND OP$<>"D" THEN 210
240 IF OP$="D" THEN C = 26 - C
300 INPUT "MESSAGE";ME$
310 ML = LEN(ME$)
320 IF ML=0 THEN 300
330 EM$ = ""     :          REM  CLEAR ENCODED
340 FOR I = 1 TO ML
350 L$ = MID$(ME$,I,1) : REM GET LETTER
360 GOSUB 4000 :          REM  ENCODE IT
370 EM$ = EM$ + L$:       REM  BUILT ENCODED
```

```
380 NEXT I
400 PRINT EM$
410 PRINT "DO ANOTHER ?"
420 GET AN$
430 IF AN$="" THEN 420
440 IF AN$<>"N" THEN 300
990 END
3998                        REM  ENCODE A LETTER
3999                        REM  IF SPACE, RETURN
4000 IF L$=CHR$(32) THEN RETURN
4010 L = ASC(L$) - 64
4020 L = L + C     :        REM  ENCIPHER IT
4030 IF L>26 THEN L=L-26
4040 L$ = CHR$(L+64)
4090 RETURN
```

Challenges

1. Add a time display like the one in TIMED SCRAMBLER to the matching and multiple choice programs.
2. Add a routine to MATCH MAKER or MULTIPLE CHOICE that will total the number of right and wrong answers. Can you save the information to disk? See SAVE/LOAD SPRITE in Chapter 7.
3. Make some other "advertising" logo programs besides PUPPY CHOW™.
4. If you are interested in ciphers or codes, try reading David Kahn's *The Code Breakers.* Try your hand at making an unbreakable code.

7

THE VIDEO ARCADE

Programs:

Sprite Away:

WINDOW	The program creates a window sprite and displays it on the screen.
BIG WINDOW	This program shows your sprites "how to grow up."
TWO SPRITES	You can watch a dark cloud go by outside your sprite WINDOW.
SPRITE FLY	A sprite flies from the bottom to the top of your screen.

Hardware to build Sprites:

SPRITE EDITOR	This is an easy-to-use program to build and display any sprite you design with any chosen color.
SAVE/LOAD SPRITE	It does what it says, to tape or disk.
WHICH SPRITES	This routine tells which sprites are being displayed.
DISABLE SPRITE	This program is a real turn-off, for sprites at least.
CHANGE SPRITE	This program is a real turn-on for sprites and their friends.

How to fly a mean Sprite:

JOYSTICK SPRITE	This program allows people over 40 to fly sprites.
JOYSTICK SPRITE 2	This program trains sprite flyers from the ages of 2 through 39.
JOYSTICK SPRITE 3	This adds a device to detect collisions with text.
JOYSTICK SPRITE 4	This adds a manual override (joystick fire button) to the collision detector.
SPRITE RACER	This is your working model of an arcade game.
SPRITE RACER 1	A speed governor is added along with a much heavier penalty for crashing. In addition, a new, harder course is provided for your racing pleasure.

Sprite Away

The programs in this section will enable you to build, display, and move sprites on your TV screen. By the time you have completed this chapter, you will be able to do the following:

1. Build very simple sprites.
2. Display a sprite at any point on the screen.
3. Create simple animation routines for sprites on your screen.
4. Change the color of any sprite.
5. Double the width or height of any sprite.
6. Move several sprites on the screen at the same time.

Software to Build Sprites

The programs in this section will allow you to design, display, save, load, and manipulate any single-colored sprite. By the time you have completed this section, you should be able to do the following:

1. Design any sprite you wish.
2. Save/load a sprite with disk or tape.
3. Determine which sprites are being displayed.
4. Stop or erase a displayed sprite.
5. Display a sprite in memory as several sprites on the screen.

6. Change the display color of any sprite on the screen.
7. Double the height or width of any displayed sprite.
8. Change the location of any sprite displayed on the screen.
9. Reverse a sprite, just like the reverse key on your keyboard.

How to Fly a Mean Sprite

This section shows you how video arcade games work. By the time you finish this section, you should be able to do the following:

1. Fly a sprite around the screen, using your joystick.
2. Control the speed of your sprite both horizontally and vertically.
3. Detect collision with text.
4. Build a complete video game and then improve it.

Sprites Made a Bit Easier

This is not an easy chapter to grasp. For that reason, we placed it near the end of our book. If you often seem lost in technical mumbo jumbo, the simplest explanation is that there's no easy way around it.

What, you may ask, is a sprite? A sprite is a small picture that can be moved around on the video display. What makes sprites so important is that you can make pictures with them as well as move those pictures around on the screen with a joystick or keyboard.

Sprites come in two types: memory sprites and display sprites. Memory sprites (or at least that is what we call them) are like pictures on a table. Display sprites are like picking the pictures up and placing them on the screen so that you can see them. We can have up to 256 sprites "on the table" and have eight of them picked up and displayed on the screen at once.

Want to see a messy sprite? Type the following (*don't type the REMs*):

```
POKE 2040,3          :REM ASSIGN MEMORY SPRITE 3 TO DISPLAY 0
POKE 53248+21,1      :REM ENABLE (TURN ON) DISPLAY SPRITE 0
POKE 53248,100       :REM HORIZONTAL (X) LOCATION = 100
POKE 53249,100       :REM VERTICAL (Y) LOCATION = 100
POKE 53249,200       :REM VERTICAL (Y) LOCATION = 200
```

These POKEs can be done in any order. For example, you can set the X,Y first and then enable the sprite, and so on.

The first POKE to 2040 told the 64 to pick up picture (oops—memory sprite) number 3 and call it number 0 (assign it to display sprite 0). The next POKE to 53248+21 told the 64 to display sprite 0, and the next two POKEs told the 64 where to locate it on the screen. When you typed the last POKE,

the sprite (not a very pretty picture) should have jumped down on the screen. The sprite may "glitter" a bit at the top. This is caused by the 64 changing some memory locations in the memory sprite while you are looking at the screen.

What you see on the screen is really a bunch of dots, some of which are turned on and some are turned off. A memory sprite is simply the pattern of ons and offs that make up the picture. Sprites on the 64 are 24 dots wide and 21 dots high. If you would like to see them a bit better, type:

```
POKE 53248+23,1   : REM DOUBLE VERTICAL SIZE - SPRITE 0
POKE 53248+29,1   : REM DOUBLE HORIZONTAL SIZE - SPRITE 0
```

Now you should be able to see the individual dots a little better. You have also learned that display sprites can be doubled in size, like using a photo enlarger on your picture. Making a sprite look decent is simply a matter of turning the dots on in the right places. To see a different sprite, try:

```
POKE 2040,192     : REM MEM SPRITE 192 AS DISPLAY SPRITE 0
POKE 53248+39,0   : REM TURN SPRITE 0 ON AS BLACK
```

This should give you a white and then a black box. Now try:

```
FOR I=0 to 64: POKE 12288+I,85 : NEXT
```

and then:

```
FOR I=0 to 64: POKE 12288+I,170 : NEXT
```

or even:

```
FOR I=0 to 64: POKE 12288+I, I : NEXT
```

and, if you are tired of seeing sprites right now,

```
POKE 53248+21,0
```

What happened was that you told the 64 to assign (or connect) memory sprite 192 to display sprite 0 (POKE 2040,192). Memory sprite 192 uses 64 memory locations from 192*64=12288 to 12288+63=12351. The FOR-NEXT loops then changed those memory locations so that the pattern on the screen changed. For convenience and safety's sake, we only use memory sprites 192 to 255 (64 of them), which are memory locations 12288 to 16384. In our programs we then refer to memory sprites 0 to 63 even though these are known to the Commodore 64 as memory sprites 192 to 255.

Don't worry about having to make elaborate calculations to turn a sprite on or off; it's all automatic in our programs. The variable SB is the memory sprite (0 to 63); for example, if we (POKE 2040,192+SB) we have added our sprite number (0 to 63) to 192. Location 2040 is used to hold the location of the memory sprite that display sprite 0 will use. If SB had a value

of 6, then memory sprite (192+6) or 198 would be assigned as sprite 0. If this seems confusing, consider the following table:

Our Memory Sprite Number	*Variable SB*	*CBM 64 Memory Sprite Number*	*Where in Memory It Is*
0	0	192	12288 to 12351
1	1	193	12352 to 12415
2	2	194	12416 to 12799
.	.	.	.
.	.	.	.
.	.	.	.
63	63	255	16320 to 16383

The program SPRITE EDITOR lets you create much prettier pictures than we have shown you so far. You can make 64 different ones, all available in memory at the same time.

Display Sprites

Once one or more memory sprites have been defined in memory, we can display some of them. There are eight display sprites and locations 2040 to 2047 tell the 64 which memory sprites to use for which display sprites. This works just like putting messages into eight mailboxes (one box for each display sprite). The message you should POKE is the number of the memory sprite (192 + SB). A POKE to 2040 tells the 64 which memory sprite to use for display sprite 0; to 2041 for display sprite 1; and so on:

MEMORY LOCATION	*DISPLAY SPRITE #*
2040	0
2041	1
2042	2
2043	3
2044	4
2045	5
2046	6
2047	7

So far, it's pretty easy. We have 64 memory sprites possible, at least in our programs, and we can display any of these memory sprites as one of the eight display sprites.

Now for the good part. Each display sprite can have the following properties or qualities:

Color
On (enabled)
Off (disabled)
Location (X,Y)
Vertical size (normal or expanded)
Horizontal size (normal or expanded)
Multicolor mode (not discussed here)
Collision detection (with text or other sprites)

First, since it is the easiest, let's deal with color. The dots in a display sprite can be displayed in any one of 16 colors. In normal display sprite mode, each sprite will have dots of only one color. Each display sprite has its own memory location to contain the color value. By POKEing the color value into the right sprite memory location, you can change its color. In a previous example, we displayed the sprite in black by typing:

```
POKE 2040,192      : REM DISPLAY MEM SPRITE 0 (192)
                     AS DISPLAY SPRITE 0
POKE 53248+39,0    : REM SET DISPLAY SPRITE 0 TO BLACK
                     (0 IS THE COLOR VALUE FOR BLACK)
```

The following chart will come in handy in figuring out the right sprite color setting.

Display Sprite Number	*Memory Location*	*Sprite Colors (Numbers to POKE)*	
0	53248 + 39	0—BLACK	8—ORANGE
1	53248 + 40	1—WHITE	9—BROWN
2	53248 + 41	2—RED	10—LIGHT RED
3	53248 + 42	3—CYAN	11—DARK GRAY
4	53248 + 43	4—PURPLE	12—MEDIUM GRAY
5	53248 + 44	5—GREEN	13—LIGHT GREEN
6	53248 + 45	6—BLUE	14—LIGHT BLUE
7	53248 + 46	7—YELLOW	15—LIGHT GRAY

We listed the memory locations above with a 53248 + instead of the actual locations because of the limits of human (rather than computer) memory. The starting memory location for the Video Interface Control chip (known as the VIC II) control information is 53248. It is easier for us humans to let a computer variable remember the 53248 and just add the location number (39, 40, and so on, up to 46) to set the color of each of the eight display sprites. Note that the number for the sprite color (0–15) follows the display sprite location after a comma, for example

```
POKE 53248+39,4
```

causes display sprite 0 to turn to purple. We will come back and do some POKEing around with the color locations once we know how to locate a sprite.

Locating a sprite on the screen is just a bit more difficult (and you will see that I mean a *bit* more difficult) than telling the 64 its color. A sprite is located by its horizontal (X) and vertical (Y) positions. The top left of the video screen is defined to be X=0 and Y=0. As you move from left to right, the X value increases from 0 to a maximum of 344. As you go from the top toward the bottom, the Y value increases from 0 to 255. These X and Y values are stored in control locations in the VIC II chip, which are just memory locations. To move a sprite from 0 to 255 in the X or Y direction, you POKE the X or Y values into the following memory locations:

Display Sprite Number	*Horizontal X Location*	*Vertical Y Location*
0	53248 + 0	53248 + 1
1	53248 + 2	53248 + 3
2	53248 + 4	53248 + 5
3	53248 + 6	53248 + 7
4	53248 + 8	53248 + 9
5	53248 + 10	53248 + 11
6	53248 + 12	53248 + 13
7	53248 + 14	53248 + 15

In the rest of this section, we will assume that you set the variable S equal to 53248. We will also use the variables SA for the number of the display sprite and SB for the number of the memory sprite. To display sprite 1 at 100,100 on the screen, type the following:

```
S = 53248                  : REM VIC II LOCATIONS
SA = 1                     : REM DISPLAY SPRITE 1
SB = 0                     : REM MEMORY SPRITE 0
POKE 2040+SA,192+SB        : REM ASSIGN MEMORY SPRITE SB
                           : REM TO DISPLAY SPRITE SA
POKE S+21,2↑SA             : REM ENABLE DISPLAY SPRITE SA (1)
POKE S +2*SA,100           : REM X=100
POKE S +2*SA + 1,100       : REM Y=100
```

Now, if you do not mind stopping at 255 in the horizontal direction, then this is enough to know. But if you wish to set an X value greater than 255, the story continues. A single memory location—a byte—can hold a number no larger than 255. Two bytes can hold a number no larger than

65,535. The Commodore designers could have used two bytes to store the X position for each display sprite, but using two bytes would waste memory space on the VIC chip because 344 is quite a lot smaller than 65,535. What they did was to use a single extra bit for each display sprite. The horizontal location is the sum of the value in the memory location for the horizontal X shown above (53248 + 2*SA) plus either 0 or 256. If the extra bit is "on," then add 256. If it is off, then add 0. Thus, if the bit is turned on, you can only locate a display sprite at X locations from 256 to 344. POKEing any number larger than 88 into (53248 + 2*SA) when the bit is on causes the sprite to be off the screen on the right side, making it very hard (impossible) to see. If you want to display a sprite at 310 horizontally, then you must turn the bit on and POKE the value 54 (310–256) into (53248 + 2*SA) because, with the bit on, the machine adds the 256 to 54 and gets 310.

These "big X" bits are turned on (or off) at memory location 53264 (S + 16). Let's try it by typing:

```
FOR I= 0 TO 64:POKE 192*64 + I, 255:NEXT :REM DEFINE SPRITE
POKE 2040+SA,192                         :REM ASSIGN TO 1
POKE S + 2*SA,54                         :REM X=54
POKE S + 2*SA+1,100                      :REM Y=100
POKE S+21,2↑SA                  :REM ENABLE SA
POKE S+16,2↑SA                  :REM X=256 + 54
POKE S+16,0                              :REM X=54
```

The sprite should jump across the screen. It is moving from horizontal location 54 to 310 (very quickly). Try playing with this a bit before we go on. To provide some company to our lonely sprite, type

```
POKE 2040,192      :REM ASSIGN MEMORY 0 TO DISPLAY 0
POKE S,54          :REM X=54
POKE S,150         :REM Y=150
POKE S+21,3        :REM ENABLE BOTH DISPLAY 0 AND 1
```

and POKE the numbers 0,1,2,3 into S + 16. You should have two sprites on the screen moving from one side to the other.

These bits are called the Most Significant Bits of X, or the MSB, and memory location 53264 (S + 16) is where they are. POKEing 1 into (S + 16) turns the bit on for display sprite 0, POKEing 2 turns the bit on for display sprite 1 (*and* off for 0), while POKEing 3 turns the bits on for both display sprites 0 and 1. Note that the Commodore designers used a single memory location to store the MSBs for all eight display sprites. This is no accident because a byte has eight bits. The problem with being so efficient (using one byte instead of eight) is that it makes it difficult to turn the bits on and off independently for each of the display sprites. If you are using only display sprite 0, then POKE S+16,1 adds 256 to the horizontal position (53248) and POKE S+16,0 adds 0 to it.

If we want to move just one sprite from, say, 54 to 310, leaving the others where they are, we first need to PEEK(S+16) to find out what bits are on and off. We then OR that value with the number in the following table and POKE the result back into S+16.

DISPLAY SPRITE #	*VALUE*
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

For example, if we want display sprite 4 to move from 54 to 310, we type the following:

```
POKE S+8, 54                      :REM SPRITE 4 AT X = 54
POKE S+16,PEEK(S+16) OR 16  :REM SPRITE 4 NOW AT X = 310
```

The OR operator adds 16 to the value in S+16 unless display sprite 4's MSB is already on. You might want to play with the OR operator by printing some number ORed with another number. Of course, you don't have to understand the OR statement to move sprites around. You simply need to follow the directions we've provided. Those who wish to figure it all out may notice that the numbers in the table above are simply 2 raised to the power of the display sprite number; for example, 4 is 2 to the second power; 16 is 2 to the fourth power.

To turn the MSB of a particular sprite off, we again use the PEEK statement, but this time we AND the value of the PEEK with the numbers in the following table.

DISPLAY SPRITE #	*VALUE*
0	255 – 1
1	255 – 2
2	255 – 4
3	255 – 8
4	255 – 16
5	255 – 32
6	255 – 64
7	255 – 128

The values are listed that way to make it easy to relate the numbers to turn a sprite on and off. Using those numbers, to move sprite 4 back to 54, we would type:

```
POKE S+16, PEEK(S+16) AND (255-16)
```

Rather than remembering the numbers in the two tables above, we can let the computer do it for us. We recommend that you take care of the MSB location by first setting SA to the number of the desired display sprite and then use:

Turn off MSB of sprite SA	POKE S+16,PEEK(S+16) AND (255 − 2↑SA)
Turn on MSB of sprite SA	POKE S+16,PEEK(S+16) OR 2↑SA

In addition to the MSB of X location, there are seven other sprite control locations. For each of these, there is a single memory location that controls all eight display sprites. Their functions and locations are:

LOCATION (S = 53248)	*FUNCTION*
S+16	MSB X value
S+21	Enable (display the sprite on the screen)
S+23	Double vertical size
S+27	Background display priority
S+28	Multicolor mode
S+29	Double horizontal size
S+30	Sprite to sprite collision detect
S+31	Sprite to text collision detect

For example, suppose we wish to enable (see) display sprite 5 and leave the other sprites alone. The following statements do this:

```
SA = 5
POKE S + 21, PEEK(S+21) OR (2↑SA)
```

To disable sprite 5, we need to use the AND operator:

```
POKE S + 21, PEEK(S+21) AND (255 − (2↑SA))
```

Except for the collision detect locations, everything operates just like the MSB. In location (S + 27), the "background display priority" location, if the bit for display sprite 3 is on, then sprite 3 will go behind any text displayed on the screen instead of in front of it.

The collision detect locations are slightly different. You PEEK at them to see if a sprite is touching something else, that is, another sprite or text. The sprite-to-sprite collision detect tells only whether a particular sprite has

collided with another sprite—not which one. You usually want to do something when a sprite has hit something, using an IF–THEN statement. If sprite SA is touching some text, then:

```
PEEK(S+31) AND 2↑SA
```

will equal 2↑SA. You could use an IF–THEN statement, such as:

```
IF (PEEK(S+31) AND 2↑SA = 2↑SA) THEN GOSUB 2000
```

where line 2000 begins a routine doing what should be done when sprite SA is touching some text. For sprite to sprite collisions, substitute S+30 for S+31.

Designing a Video Game

To make clearer how to use sprites, we will describe how you go about constructing a video game. We will take as our example a familiar Atari™ game, Air Combat™. This was one of the original games made for the Atari, and it shows very simply how one uses sprite graphics to create arcade effects. For those who have not seen the game, we will describe it. The game involves shooting at targets, which are crossing the screen near the top. At the bottom is your tank or gun. The tank can shoot missiles at the targets, which look like airplanes. Thus, there are three sprites: tanks, missiles, and airplanes.

The first task in making this game is to design the three sprites. Then we need to decide how the sprites are to move. The tank is the easiest. It moves under joystick control in the horizontal direction only. The missiles are next. They move in the Y direction only, just like the sprite in SPRITE FLY. The routine there will work fine. For simplicity, allow only one missile at a time on the screen. For variants, you can either disable the fire button while a missile is on the screen or remove the old missile and start a new one whenever the fire button is hit. The X position of the missile when fired will be that of the tank. Finally, we need to move the targets. This can be done by simply modifying the routine in JOYSTICK SPRITE, which now uses joystick control, but can easily be changed to work in a FOR–NEXT loop, just like the missile movement.

The targets can be brought in at the same speed all the time, or you can try setting its speed at random. The same is true of its height (its vertical, or Y, position). This can be the same always, or in a simple variant, or can be varied randomly.

Finally, the heart of the game is detecting hits and scoring. This can be handled by using the sprite collision location. By detecting a collision and making sure that it occurred between a missile and a target (not the tank *and* the missile), you can verify a hit and add to the score accordingly. The score

can be displayed at the top or bottom, using a display routine such as is used in the clock displays in Chapter 4, "What Time Is It?".

The stopping condition for the game also needs to be decided. This can either be a time limit, a number of missiles fired, or a fixed number of targets to shoot at. Once the game is over, report the score and wait for a request to restart. Other features you can add would be a high score display and using the sprite expansion capability to make the game easier or harder. Color choices should be made to make the display as sharp as possible. Another possibility is sound effects, for example, when a target starts or when there is a hit.

Let's summarize by giving a simple sketch of making a game: First, you have to come up with a game idea or at least some initial parts of a game idea. Given the idea, decide how many sprites of what sort you want and then use the SPRITE EDITOR to design them. Once designed, you should save them with SAVE/LOAD SPRITES. Now, experiment with moving them as you wish, using variants of SPRITE FLY or JOYSTICK SPRITE 2 to obtain animation. Once the sprites are moving correctly, decide how they should interact. What should happen when they collide? Do you need sound effects? What sort of scoring?

If you proceed in a step-by-step fashion, gradually adding pieces while making sure that the whole thing still works the way you want, you can avoid the frustration that comes from trying to program an entire game from scratch. Every programmer makes believe that his or her programs have no "bugs," but it is close to impossible to program any lengthy program without some. It's best to plan ahead for this problem. Add small chunks at a time to a working version, test it, and make sure it's working properly before going further.

As you gain experience, you will also gain confidence, but be wary of excessive confidence. It is excessive confidence that tells you not to save or back up your work right before some mistake destroys it; it is foolish confidence that makes some programmers believe they can write difficult programs without bugs. Be patient, and you will accomplish far more.

BASIC Commands Used in This Chapter

ABS	DIM
AND	END
ASC	FOR
CHR$	GOSUB
CLOSE	GOTO
DATA	IF
DEF	INPUT

INPUT#	POKE
INT	PRINT
LEFT$	PRINT#
LEN	READ
LET	REM
MID$	RETURN
NEXT	SPC
ON	STEP
OPEN	SYS
OR	TAB
PEEK	TI

Programming Techniques Used in This Chapter

1. *Convert 8-string characters to a byte.* In SPRITE EDITOR, at lines 620–690, eight characters from A$(I) are translated into a single byte. Each character in A$(I) is tested: if it is a space, then the relevant bit is turned off; if the character is not a space, then the bit is turned on. This is the basic tool used to convert the 21 lines of strings in A$(I) into 64 bytes POKEd into a memory sprite.
2. *Detecting collisions with text and other sprites.* The text collision location (S + 31) is initialized by PEEKing at it in line 105 of JOYSTICK SPRITE 3. Line 105 reads:

   ```
   105 A = PEEK(S+31)
   ```

   The variable A is ignored and is only used to make the PEEK a valid statement. Line 810 then PEEKs the location for real, and line 840 does something only when a new collision with text has occurred. This scheme could be used as a means of inputting data to the 64, so that one could use the joystick as the sole input device.

   Collision with another sprite is checked in the SPRITE RACER program at lines 820–830. The vertical line across the race course at the top of the screen is a sprite, so finishing the race is determined by seeing if the sprite car has hit the sprite finish line!
3. *Disabling (turning off) a sprite.* The program DISABLE SPRITE shows you how to do this to any sprite at line 5030.
4. *Display a sprite location.* In WHICH SPRITES, the sprite enable location (S + 21) is PEEKed in line 1000, and the lines from 1040 to 1140 take it apart, bit by bit, showing which sprites are enabled. This technique could be used with any sprite location. Try it.

5. *Enabling a single sprite.* To enable (display) display sprite SA without affecting the other display sprites, use line 3070 of SPRITE EDITOR.
6. *Expanding a sprite horizontally and vertically.* Lines 4160 and 4180 of CHANGE SPRITE show you how to double the height or width of any display sprite.
7. *Loading a file from or saving it to tape or disk.* This is a most important idea: Data can be stored on some external device and not evaporate when you turn the power off. If we were not able to do this, the computer would be of quite little value. Both these tasks are done in SAVE/LOAD SPRITE, which may be one of the most instructive programs in this book. The lines that do this are:

Lines	*Function*	*Tape/Disk*
4300–4360	LOAD	TAPE
4220–4280	LOAD	DISK
3300–3360	SAVE	TAPE
3220–3270	SAVE	DISK

These routines can easily be modified for use elsewhere. The key statements are the PRINT# and INPUT# ones in lines 3250, 3330, 4240, and 4320. A comma must follow the file number, and then you add a variable list. For example, if you wished to SAVE the race course in SPRITE RACER, you would need to output the string array R$(I). The following would accomplish this:

```
FOR I = 1 TO 23
PRINT#8,R$(I)
NEXT I
```

The output must end with a CLOSE 8 statement or whatever file number (the 8) that you used (by OPENing). The most difficult statement in file access is the OPEN. This is used in SAVE/LOAD SPRITE in lines 3220 and 4220 for disk, and 3300 and 4300. Since tape access is so simple, we will only discuss disk. Line 3220 reads:

```
3220 OPEN 8,8,8,"0:"+SN$+",S,W"
```

The first 8 is an arbitrary number (up to 255) by which later PRINT# and INPUT# commands will refer to the file. The second 8 simply refers to your disk drive. The third 8 is called a channel number and must be between 2 and 14. The SN$ variable contains the file name under which you save the data. The "S,W" refer first to the type of file to be written, sequential, and then to the fact that we are writing it, not reading, as in line 4220. Try your hand at saving other types of data. There are two other ways to access files: (1) instead of reading with

INPUT#, you can use a GET# statement; (2) to put data into a sequential file, one can use a CMD command. For example, suppose you wish to put a program listing into a sequential file, possibly to read it with some type of text editor. If the program is already in memory, simply type:

```
OPEN 131,8,8,"0:MY PROGRAM,S,W"
CMD 131
LIST
```

To look at this file, simply read it by using a program containing INPUT# or GET# commands.

8. *Menu input.* A menu of alternatives is one of the best ways to get user input to a program. In CHANGE SPRITE, lines 110–210 print a menu on the screen, line 220 inputs the desired choice, and lines 300–340 do the desired task.

Sprite Away

Window

This program creates a sprite that looks like a window and displays it on the screen. To change the color of the sprite, change the number 15 in line 280 to any number between 0 and 15. To see the sprite immediately change color after running the program, type (when you see READY.):

```
POKE 53248+39, 13
```

This will change the color of sprite 0 to a light green (color 13—see Table 2 of the Appendix for the color table). The horizontal and vertical location(s) of sprite 0 can be changed in lines 290 and 300. Better yet, you can move the window around after the program runs. POKE numbers between 0 and 255 in locations 53248 (horizontal) and 53249 (vertical). Since we set S = 53248, try (just after running the program):

```
POKE S,200
POKE S+1,200
```

Watch out, though, you can move the window off the screen completely. Oh, by the way, the sprite will stay there until you turn it off by:

```
POKE 53248+21,0
```

```
5 REM              WINDOW
6 REM  THIS CREATES A WINDOW SPRITE
7 REM  AS MEMORY SPRITE 1
8 REM  AND DISPLAYS AS SPRITE Ø
5Ø L1 = 3*4Ø96
```

```
90 :
100 PRINT CHR$(147) :      REM CLEAR SCREEN
110 REM          SET SPRITE 0 TO POINT TO
120 REM          L1 IN MEMORY
130 POKE 2040,192:         REM SET SPRITE 0 PTR
140 REM          POKE 1 INTO SPRITE LOCATIONS
150 FOR I=L1 TO L1+62
160 POKE I,1
170 NEXT
180 FOR I=0 TO 2
190 POKE L1+I,255
200 POKE L1+60+I,255
210 NEXT :                 REM FILL THESE (255)
220 FOR I=3 TO 60 STEP 3
230 POKE L1+I,PEEK(L1+I) OR 128
240 NEXT :                 REM  FILL IN
250 :
260 S=53248 :              REM FIRST VIC REGISTER
270 POKE S+21,1 :          REM  DISPLAY SPRITE 0
280 POKE S+39,15 :         REM SET COLOR (15)
290 POKE S,168 :           REM SET X POSITION
300 POKE S+1,150 :         REM SET Y POSITION
```

Big Window

This program shows your sprites "how to grow up." Letting S=53248, POKEing S+23 with a 1 doubles the width of the sprite, and POKEing S+29 with a 1 doubles the height. POKEing 0s makes it normal (this is for sprite 0 only, remember). The program just keeps on doing it until you hit RUN/STOP. Remember that to get rid of the sprite, you will have to POKE S+21,0.

```
5 REM        BIG  WINDOW
6 REM  THIS IS WINDOW WITH
7 REM  LINES 310 TO 390 ADDED
50 L1 = 3*4096
90 :
100 PRINT CHR$(147) :      REM CLEAR SCREEN
110 REM          SET SPRITE 0 TO POINT TO
120 REM          L1 IN MEMORY
130 POKE 2040,192:         REM SET SPRITE 0 PTR
140 REM          POKE 1 INTO SPRITE LOCATIONS
150 FOR I=L1 TO L1+62
160 POKE I,1
170 NEXT
180 FOR I=0 TO 2
190 POKE L1+I,255
200 POKE L1+60+I,255
210 NEXT :                 REM FILL THESE (255)
220 FOR I=3 TO 60 STEP 3
```

```
230 POKE L1+I,PEEK(L1+I) OR 128
240 NEXT :                    REM  FILL IN
250 :
260 S=53248 :                 REM FIRST VIC REGISTER
270 POKE S+21,1 :             REM  DISPLAY SPRITE 0
280 POKE S+39,15 :            REM SET COLOR (15)
290 POKE S,168 :              REM SET X POSITION
300 POKE S+1,150 :            REM SET Y POSITION
310 REM          NOW CHANGE THE SIZE
320 FOR I=0 TO 1
330 FOR J=0 TO 1
340 FOR K=0 TO 200:NEXT:      REM WAIT A BIT
350 POKE S+29,I :             REM WIDTH
360 POKE S+23,J :             REM HEIGHT
370 FOR K=0 TO 200:NEXT:      REM WAIT A BIT
380 NEXT J: NEXT I
390 GOTO 320
```

Two Sprites

You can watch a dark cloud (well, black box) go by outside your sprite WINDOW. Note lines 400 to 420 move the cloud; obviously, they can be changed. Sprite 0 is always in front of sprite 1. This makes the window appear to be in front of the cloud. You may want to leave one or both of the sprites on the screen to try SPRITE FLY, the next program.

```
5 REM          TWO SPRITES
6 REM  THIS IS WINDOW  WITH LINES FROM
7 REM  300 ON ADDED AND 290 CHANGED
8 :
10 REM THIS CREATES TWO MEMORY SPRITES,
11 REM 0 AND 1, AND DISPLAYS THEM
12 REM AS DISPLAY SPRITES 0 AND 1
30 :
50 L1 = 3*4096
90 :
100 PRINT CHR$(147) :      REM CLEAR SCREEN
110 REM          SET SPRITE 0 TO POINT TO
120 REM          L1 IN MEMORY
130 POKE 2040,192:         REM SET SPRITE 0 PTR
140 REM          POKE 1 INTO SPRITE LOCATIONS
150 FOR I=L1 TO L1+62
160 POKE I,1
170 NEXT
180 FOR I=0 TO 2
190 POKE L1+I,255
200 POKE L1+60+I,255
210 NEXT :                    REM FILL THESE (255)
220 FOR I=3 TO 60 STEP 3
230 POKE L1+I,PEEK(L1+I) OR 128
```

```
240 NEXT :                  REM  FILL IN
250 :
260 S=53248 :               REM FIRST VIC REGISTER
270 POKE S+21,1 :           REM  DISPLAY SPRITE 0
280 POKE S+39,15 :          REM SET COLOR (15)
290 POKE S,35  :            REM SET X POSITION
300 POKE S+1,150 :          REM SET Y POSITION
310 POKE S+29,1
320 REM         LINES 330-350 MAKE A BOX SPRITE
330 FOR I=L1+64 TO L1+64+62
340 POKEI,255
350 NEXT
355 POKE 2041,193:          REM SET SPRITE 1 PTR
360 POKE S+40,0 :           REM SET COLOR  (0)
370 POKE S+21,3 :           REM DISPLAY 2 SPRITES
380 POKE S+3,140:           REM SET Y LOCATION
385 REM         LINES 400-420 MOVE SPRITE 1
390 REM         LINES 400-420 MOVE SPRITE 1
400 FOR I=180 TO 24 STEP-1
410 POKE S+2,I
420 NEXT
```

Sprite Fly

A sprite flies from the bottom to the top of your screen. Its speed can be changed by changing line 3000. To fly the sprite from right to left, remove the "+1" in line 3030. To make it fly diagonally add:

```
3031 POKE S + 2*SA,I
```

Remember, you must have a sprite already displayed on the screen before you can fly it. We used the dark cloud from TWO SPRITES (POKE S+21,2 gets rid of the window and keeps the cloud). Try complicated pattern flights like "Big Trak."™ A very interesting thing happens when you add the following lines:

```
3031 POKE S + 2*SA     ,I
3032 POKE S + 2*SA + 1,255-I
3033 POKE S + 2*SA     ,255-I
```

```
5 REM       SPRITE FLY
11 REM  REQUIRES A SPRITE TO BE
12 REM  DISPLAYED ON THE SCREEN
15 :
20 REM  TRY CHANGING YV IN LINE 3000
90 :
100 REM         SPRITES ARE NUMBERED 0 TO 7
110 INPUT "WHICH SPRITE";SA
```

```
120 S=53248 :                REM  VIC  REGISTERS
130 POKE S+2*SA,100
3000 YV = - 1 :              REM  Y VELOCITY
3010 FOR I = 255 TO 0 STEP YV
3020 REM       SETS Y POSITION FOR SPRITE SA
3030 POKE S + 2*SA + 1,I
3040 NEXT I
```

Hardware to Build Sprites

Sprite Editor

This is an easy-to-use program to build and display any sprite you design with any chosen color. In lines 200 through 220, there must be 24 characters or spaces between the quotation marks, and so we check for that in lines 300 to 390. Use the screen editing features of the Commodore 64 in creating lines 201 to 220. To design a sprite, type the following:

```
LOAD "SPRITE EDITOR",8
LIST 200-220
```

Use the screen editing features to change the airplane now shown into your own design. Remember to hit the RETURN key on each line. We LIST 200–220 and make all the changes on the screen without hitting return, and then cursor to the beginning of line 200 and return 20 times to make sure the lines are stored correctly. This procedure seems the simplest way to do it to us. Running the program will display your sprite. Use the program SAVE/LOAD SPRITE to save a copy of your sprite to tape or disk.

```
5 REM        SPRITE EDITOR
13 :
14 REM    THERE ARE 8 DISPLAY SPRITES
15 REM    WHICH CAN BE DISPLAYED,
16 REM    NUMBERED 0 TO 7 -- SA
19 :
20 REM   THIS PROGRAM CAN SAVE UP TO
21 REM   64 DIFFERENT MEMORY SPRITES,
22 REM   NUMBERED 0 TO 63 -- SB
30 :
90 :
100 DIM A$(20)
110 PRINT CHR$(147) :      REM  CLEAR SCREEN
120 INPUT "SAVE AS WHICH MEMORY SPRITE (0 TO 63)";SB
140 INPUT "DISPLAY AS WHICH COLOR (0 TO 15)";CO
150 INPUT "DISPLAY AS WHICH SPRITE (0 TO 7)";SA
180 :
```

```
190 :
200 A$( 0)="           XXX           "
201 A$( 1)="           XXX           "
202 A$( 2)="           XXX           "
203 A$( 3)="           XXX           "
204 A$( 4)="           XXX           "
205 A$( 5)="         XXXXX           "
206 A$( 6)="        XXXXXXX          "
207 A$( 7)="       XXXXXXXXX         "
208 A$( 8)="      XXXXXXXXXXX        "
209 A$( 9)="     XXXX XXX XXXX       "
210 A$(10)="    XXXX  XXX  XXXX      "
211 A$(11)="   XXXX   XXX   XXXX     "
212 A$(12)="          XXX            "
213 A$(13)="          XXX            "
214 A$(14)="          XXX            "
215 A$(15)="         XXXXX           "
216 A$(16)="        XXXXXXX          "
217 A$(17)="       XXXXXXXXX         "
218 A$(18)="                         "
219 A$(19)="                         "
220 A$(20)="                         "
300 FOR I=0 TO 20
310 IF LEN(A$(I))=24 THEN 370
320 PRINT "A(";I;") IS NOT 24 LONG"
330 PRINT A$(I)
350 ER=1
370 NEXT
390 IF ER=1 THEN END
400 GOSUB 3000:            REM TURN ON SPRITE
500 :
510 :
520 REM  LOCATION OF SPRITE IN MEMORY
530 REM  --STARTS AT 3*4096
540 SP=3*4096 + 64*SB
550 REM  LINES 570 TO 740 TRANSFORM THE
560 REM  A$ ARRAY INTO A MEMORY SPRITE
570 FOR I = 0 TO 20 :      REM  X LOCATION
580 FOR J = 0 TO 2 :       REM  BYTE ACROSS Y
590 BI = 0 :               REM BUILD DATA BYTE
600 :
610 REM      BIT BY BIT, GET DATA
620 FOR K = 7 TO 0 STEP -1
630 REM      GET NEXT BIT FOR BI
640 B$ = MID$(A$(I),(2-J)*8+K+1,1)
650 REM   SET HIGH BIT
660 B = ASC(B$) OR 128
670 REM   IF NOT A SPACE THEN SET BIT
680 IF B<>160 THEN BI=BI+2^(7-K)
690 NEXT K
700 :
710 REM   POKE INTO SPRITE
```

```
720 POKE SP + 3*I + (2-J),BI
730 NEXT J
740 NEXT I
999 END
2997 :
2998 REM  DISPLAY MEMORY SPRITE SB AS
2999 REM  DISPLAY SPRITE SA
3000 POKE 2040+SA,192+SB
3010 REM  LOCATION OF THE BEGINNING
3020 REM  OF THE SPRITE REGISTERS
3030 S=53248
3040 POKE S+39+SA,CO
3050 REM  THIS ENABLES SPRITE SA,
3060 REM  LEAVES OTHERS
3070 POKE S+21,PEEK(S+21)OR(2↑SA)
3080 REM  THIS IS X POSITION
3090 POKE S+2*SA,50+30*SA
3100 REM  THIS IS Y POSITION
3110 POKE S+2*SA+1,100
3200 RETURN
```

Save/Load Sprite

It does what it says, to tape or disk. You are given a chance to look at the sprite before doing it, though.

```
5 REM           SAVE/LOAD SPRITE
100 PRINT CHR$(147):S = 53248
110 C$ = "SAVE/LOAD SPRITE"
120 PRINT SPC((40-LEN(C$))/2);C$
130 PRINT:PRINT:PRINT
140                          REM  D=2 TAPE
141                          REM  D=1 DISK
150 INPUT "TAPE OR DISK";I$:D=0
160 IF LEFT$(I$,1) = "D" THEN D=1
170 IF LEFT$(I$,1) = "T" THEN D=2
180 IF D<>1 AND D<>2 THEN 100
190 PRINT "WHAT IS THE SPRITE'S NAME"
200 INPUT "E.G., FILE NAME";SN$
210 INPUT "LOAD OR SAVE";OP$
220                          REM  S=2  LOAD
230                          REM  S=1  SAVE
240 IF LEFT$(OP$,1) = "S" THEN SL=1
250 IF LEFT$(OP$,1) = "L" THEN SL=2
260 IF SL<>1 AND SL<>2 THEN 210
270 ON SL GOSUB 3000,4000
300 GOSUB 1100
310 GOSUB 1200
320 GOSUB 1300
330 GOSUB 1400
340 GOSUB 1500
```

```
990 END
999 REM  INPUT MEMORY SPRITE SB
1000 INPUT "USE WHICH SPRITE (0 TO 63)";SB
1090 RETURN
1099 REM  INPUT COLOR OF DISPLAY SPRITE SA
1100 INPUT "DISPLAY WITH WHICH COLOR (0 TO 15)";C1
1190 RETURN
1199 REM  INPUT DISPLAY SPRITE SA
1200 INPUT "DISPLAY AS WHICH SPRITE (0 TO 7)";SA
1290 RETURN
1299 REM INPUT X,Y LOCATIONS
1300 PRINT"   X,Y LOCATIONS FOR SPRITE";SA;
1320 INPUT X,Y
1390 RETURN
1399 REM DISPLAY SPRITE SA AT X,Y
1400 POKE S+2*SA,XAND255
1410 POKE S+2*SA+1,Y
1420 POKE S+16,(PEEK(S+16) AND 2↑SA) + INT(X/256)*2↑SA
1430 RETURN
1499 REM DISPLAY IT
1500 POKE 2040+SA,192+SB
1510 POKE S+21,PEEK(S+21) OR 2↑SA
1520 POKE S+39+SA,C1
1530 RETURN
2990 :
2998 REM  SAVE SPRITE TO TAPE/DISK
2999 REM DISPLAY AND THEN ASK
3000 GOSUB 1000
3010 GOSUB 1100
3020 GOSUB 1200
3030 GOSUB 1300
3040 GOSUB 1400
3050 GOSUB 1500
3070 INPUT "OK TO SAVE";A$
3080 IF LEFT$(A$,1)="Y" THEN 3200
3090 POKE S+21,0
3100 GOTO 3000
3200 L1 = 3*4096 + 64*SB
3210 IF D=0 THEN 3100:    REM TAPE OR DISK
3220 OPEN 8,8,8,"0:"+SN$+",S,W"
3230 FOR I = 0 TO 62
3240 A = PEEK(L1+I)
3250 PRINT#8,A
3260 NEXT I
3270 CLOSE8
3280 GOTO 3360
3290 REM    THIS IS THE TAPE VERSION
3300 OPEN 1,1,1,SN$
3310 FOR I = 0 TO 62
3320 A = PEEK(L1+I)
3330 PRINT#1,A
3340 NEXT I
```

```
3350 CLOSE1
3360 INPUT "DO ANOTHER?";A$
3370 IF LEFT$(A$,1)="Y" THEN RUN
3900 END
3990 :
3999 REM  LOAD SPRITE FROM TAPE/DISK
4000 GOSUB 1000
4010 GOSUB 1100
4020 GOSUB 1200
4030 GOSUB 1300
4040 GOSUB 1400
4050 GOSUB 1500
4060 PRINT "OK TO REPLACE THIS SPRITE"
4070 INPUT "WITH ONE ON DISK?";A$
4080 IF LEFT$(A$,1)="Y" THEN 4200
4090 POKE S+21,0
4100 GOTO 4000
4200 L1 = 3*4096 + 64*SB
4210 IF D=0 THEN 4100:     REM TAPE OR DISK
4220 OPEN 8,8,8,"0:"+SN$+",S,R"
4230 FOR I = 0 TO 62
4240 INPUT#8,A
4250 POKE L1+I,A
4260 NEXT I
4270 CLOSE8
4280 RETURN
4290 REM     THIS IS THE TAPE VERSION
4300 OPEN 1,1,1,SN$
4310 FOR I = 0 TO 62
4320 INPUT#1,A
4330 POKE L1+I,A
4340 NEXT I
4350 CLOSE1
4360 RETURN
```

Which Sprites

This routine tells which sprites are being displayed. It will also tell you which memory sprite is being used by each display sprite.

```
5 REM          WHICH SPRITES
6 REM  TELLS WHICH SPRITES ARE ENABLED
7 REM  AND TO WHICH MEMORY SPRITE THAT
8 REM  EACH DISPLAY SPRITE IS SET
9 :
100 S = 53248     :          REM  VIC REGISTERS
1000 X = PEEK(S+21)
1010 S1$ = "SPRITE "
1020 S2$ = " IS ENABLED"
1030 S3$ = " USES MEMORY SPRITE "
1040 SP = 0
```

```
1050 IF (X AND 1) = 1 THEN PRINT S1$;SP;S2$
1060 SP = SP + 1
1070 X = INT(X/2)
1080 IF X>0 THEN 1050
1090 L1 = 2040 :          REM  SPRITE POINTERS
1100 FOR SP = 0 TO 7
1110 SB = PEEK(L1+SP) - 192
1120 IF SB<0 OR SB>63 THEN 1140
1130 PRINT S1$;SP;S3$;SB
1140 NEXT SP
```

Disable Sprite

This program is a real turn-off, for sprites at least. Each display sprite can be turned off individually and no longer be displayed.

```
5 REM   DISABLE SPRITE
110 PRINT CHR$(147) :     REM  CLEAR SCREEN
150 INPUT "DISABLE WHICH SPRITE (0 TO 7)";SA
5000 :
5010 REM  BEGINNING OF SPRITE REGISTERS
5011 S=53248
5020 REM  THIS DISABLES SPRITE SA
5030 POKE S+21,PEEK(S+21)AND(255-2↑SA)
```

Change Sprite

This is a real turn-on for sprites and their friends. The menu is displayed by lines 110 to 210. To input a command, simply press a key. No return is needed (the GET statement in line 220 accomplishes this).

COMMANDS

"C" for Change Sprite	Enables you to change the assignment of memory sprites to display sprites. It also resets the color of the display sprite.
"E" for Expand Sprite	Allows you to expand or contract a display sprite. Separate controls for horizontal (X) and vertical (Y) are provided.
"L" for Location	Allows you to relocate a sprite anywhere on the screen. The horizontal range is between 0 and 344. The vertical range is 0 to 255.
"R" for Reverse Sprite	Reverses a memory sprite. Colored areas become empty, and empty areas become colored.
"X" for Exit	That's all, folks . . .

```
5 REM   CHANGE SPRITE
90 :
95 S = 53248
100 REM  LINES 110-210 PRINT A MENU
110 PRINT CHR$(147) :      REM  CLEAR SCREEN
120 PRINT:PRINT
130 PRINT TAB(5);"C FOR CHANGE SPRITE ":PRINT
140 PRINT TAB(5);"E FOR EXPAND SPRITE ":PRINT
150 PRINT TAB(5);"L FOR LOCATION OF SPRITE":PRINT
160 PRINT TAB(5);"R FOR REVERSE SPRITE":PRINT
200 PRINT TAB(5);"X FOR EXIT ":PRINT
210 PRINT "COMMAND ?"
215 :
216 REM  LINES            HANDLE COMMANDS
220 GET A$ : IF A$ = "" THEN 220
300 IF A$ = "X" THEN 990 :REM END PROGRAM
310 IF A$ = "C" THEN GOSUB 2000 : GOTO 110
320 IF A$ = "R" THEN GOSUB 3000 : GOTO 110
330 IF A$ = "E" THEN GOSUB 4000 : GOTO 110
340 IF A$ = "L" THEN GOSUB 5000 : GOTO 110
399 GOTO 110
990 END
999 REM  INPUT MEMORY SPRITE SB
1000 INPUT "USE WHICH SPRITE (0 TO 63)";SB
1090 RETURN
1099 REM  INPUT COLOR OF DISPLAY SPRITE SA
1100 INPUT "DISPLAY WITH WHICH COLOR (0 TO 15)";C1
1190 RETURN
1199 REM  INPUT DISPLAY SPRITE SA
1200 INPUT "DISPLAY AS WHICH SPRITE (0 TO 7)";SA
1290 RETURN
1299 REM INPUT X,Y LOCATIONS
1300 PRINT"   X,Y LOCATIONS FOR SPRITE";SA;
1320 INPUT X,Y
1390 RETURN
1392 REM DISPLAY SPRITE SA AT X,Y
1396 REM  THIS ENABLES SPRITE SA,
1398 REM  LEAVES OTHERS
1400 POKE S+21,PEEK(S+21)OR(2↑SA)
1430 REM  LEAST SIGNIFICANT PART OF X
1435 X1 = X AND 255
1440 REM  MOST SIGNIFICANT PART OF X
1445 X2 = INT(X/256)
1450 POKE S+2*SA,X1 :      REM  X POSITION
1455 POKE S+2*SA+1,Y :     REM  Y POSITION
1460 REM  FOR X > 255 ONLY
1465 POKE S+16,(PEEK(S+16) AND 2↑SA) + X2*2↑SA
1470 RETURN
1990 REM  CHANGE SPRITE ASSIGNMENTS, COLOR
2000 GOSUB 1000 :          REM GET MEMORY SPRITE
2020 GOSUB 1100 :          REM GET DISPLAY COLOR
2030 GOSUB 1200 :          REM GET DISPLAY SPRITE
```

```
2100 REM  DISPLAY MEMORY SPRITE SB AS
2101 REM  DISPLAY SPRITE SA
2102 POKE 2040+SA,192+SB
2110 X = 24 + 25*SA :      REM INITIAL X
2120 Y = 100 :             REM INITIAL Y
2130 GOSUB 1400 :          REM DISPLAY SPRITE
2140 POKE S+39+SA,CO :     REM SET COLOR
2199 RETURN
2999 REM  REVERSE THE MEMORY SPRITE
3000 GOSUB 1000 :          REM GET MEMORY SPRITE
3100 REM                   REVERSE SPRITE SB
3110 REM  GET MEMORY LOCATION OF SPRITE
3111 SP = 3*4096 + 64*SB
3120 FOR I = 0 TO 63 :     REM  DO EACH BITE
3130 REM  COMPLEMENT THE BYTE
3131 POKE SP+I,255-PEEK(SP+I)
3140 NEXT
3199 RETURN
3998 REM  EXPAND/CONTRACT DISPLAY
3999 REM  SPRITE SA
4000 GOSUB 1200 :          REM DETERMINE SA
4010 PRINT " E - EXPAND  OR  C - CONTRACT ?";
4020 GET A$ : IF A$ = "" THEN 4020
4030 PRINT
4040 IF A$="E" OR A$="C" THEN 4090
4050 REM  IF NOT E OR C THEN GO
4060 REM  BACK TO MAIN MENU
4070 RETURN
4080 :
4090 PRINT "X OR Y";
4100 GET B$ : IF B$ = "" THEN 4100
4110 REM  IF NEITHER X OR Y, RETURN
4120 REM  TO MAIN MENU
4130 IF B$<>"X" AND B$<>"Y" THEN RETURN
4140 IF A$="C" THEN 4200: REM CONTRACT
4150 REM  EXPAND X
4160 IF B$="X" THEN POKE S+29,PEEK(S+29) OR 2↑SA
4170 REM  EXPAND Y
4180 IF B$="Y" THEN POKE S+23,PEEK(S+23) OR 2↑SA
4190 RETURN
4195 REM  CONTRACT SPRITE SA
4198 REM  CONTRACT X
4200 IF B$="X" THEN POKE S+29,PEEK(S+29) AND (255 - 2↑SA)
4230 REM  CONTRACT Y
4240 IF B$="Y" THEN POKE S+23,PEEK(S+23) AND (255 - 2↑SA)
4250 RETURN
4990 :
4995 REM  RELOCATE DISPLAY SPRITE  SA
4999 :
5000 GOSUB 1200 :          REM GET MEMORY SPRITE
5020 GOSUB 1300 :          REM GET X,Y LOCATION
5030 GOSUB 1400 :          REM DISPLAY SPRITE
5040 RETURN
```

How to Fly a Mean Sprite

Joystick Sprite

This program allows people over forty to fly sprites with a joystick. It is slow because the commands are written in BASIC. There is a velocity setting that will make the movement fast but jerky. Be careful not to fly off the screen, or the program may end in an error. The joystick must be in port #2.

```
5 REM         JOYSTICK SPRITE
100 S = 53248         :      REM VIC REGISTERS
490 :
500 REM  GET NEEDED SPRITE DATA
510 GOSUB 1000
520 GOSUB 1100
530 GOSUB 1200
540 REM        SET DISPLAY SPRITE SA TO
550 REM               USE MEMORY SPRITE SB
560 POKE 2040+SA,192+SB
570 REM               SET SPRITE COLOR
580 POKE S+39+SA,CO
620 POKE S+21,2↑SA :       REM ENABLE SPRITE
630 GOSUB 1500
670 GOSUB 1300
860 REM  READ JOYSTICK 2 INPUT
861 A = PEEK(56320) AND 15
870 IF A AND 8 THEN X=X-VX
880 IF A AND 4 THEN X=X+VX
890 IF A AND 2 THEN Y=Y-VY
900 IF A AND 1 THEN Y=Y+VY
910 GOSUB 1400
920 GOTO 861
990 :
999 REM  INPUT MEMORY SPRITE SB
1000 INPUT "USE WHICH SPRITE (0 TO 63)";SB
1090 RETURN
1099 REM  INPUT COLOR OF DISPLAY SPRITE SA
1100 INPUT "DISPLAY WITH WHICH COLOR (0 TO 15)";C1
1190 RETURN
1199 REM  INPUT DISPLAY SPRITE SA
1200 INPUT "DISPLAY AS WHICH SPRITE (0 TO 7)";SA
1290 RETURN
1299 REM INPUT X,Y LOCATIONS
1300 PRINT"   X,Y LOCATIONS FOR SPRITE";SA;
1320 INPUT X,Y
1390 RETURN
1392 REM DISPLAY SPRITE SA AT X,Y
1396 REM  THIS ENABLES SPRITE SA,
1398 REM  LEAVES OTHERS
1400 POKE S+21,PEEK(S+21)OR(2↑SA)
1430 REM  LEAST SIGNIFICANT PART OF X
1435 X1 = X AND 255
```

```
1440 REM  MOST SIGNIFICANT PART OF X
1445 X2 = INT(X/256)
1450 POKE S+2*SA,X1 :      REM  X POSITION
1455 POKE S+2*SA+1,Y :     REM  Y POSITION
1460 REM  FOR X > 255 ONLY
1465 POKE S+16,(PEEK(S+16) AND 2↑SA) + X2*2↑SA
1470 RETURN
1490 REM  INPUT VELOCITIES
1500 INPUT "X VELOCITY";VX
1520 INPUT "Y VELOCITY";VY
1530 VX = ABS(VX)
1540 VY = ABS(VY)
1590 RETURN
```

Joystick Sprite 2

This program is for sprite flyers from the ages of 2 through 39. The BASIC commands have been replaced by a machine language routine. The DATA statements in lines 2010 to 2300 contain the machine language. Lines 190 to 260 put the data into memory locations where your computer can use it. The machine language routines are used in lines 730 and 800. Note that the entire BASIC program that will actually be used after the first 20 seconds is the two lines 800 and 850. The use of machine language is what provides the fantastic speedup of JOYSTICK SPRITE. Again, be sure that the joystick is in port #2.

```
5 REM         JOYSTICK SPRITE2
6 REM  THIS IS JOYSTICK SPRITE  WITH
7 REM  LINES 150-400, 590-610, 640-660,
8 REM  AND 680-850 ADDED,  AND
9 REM  LINES 860-920, AND 1392-1470
10 REM DELETED
100 S = 53248        :      REM VIC REGISTERS
150 REM  LINES 190 TO 260
160 REM  READ IN THE MACHINE LANGUAGE
170 REM  ROUTINE THAT FLIES THE SPRITE
180 REM  UNDER JOYSTICK CONTROL
190 FOR I = 49152 TO 49157
200 READ A
210 POKE I,A
220 NEXT I
230 FOR I = 49210 TO 49417
240 READ A
250 POKE I,A
260 NEXT I
300 :
310 REM  SET UP LOCATIONS FOR POKEING
320 REM  DATA TO THE MACHINE LANGUAGE
330 REM  ROUTINE THAT FLIES THE SPRITE
```

```
340 BEGIN = 12*4096
350 S1 = BE + 6
360 X0 = BE + 7
370 Y0 = BE + 9
380 XV = BE + 10
390 YV = BE + 11
400 DISP = BE + 3
490 :
500 REM  GET NEEDED SPRITE DATA
510 GOSUB 1000
520 GOSUB 1100
530 GOSUB 1200
540 REM        SET DISPLAY SPRITE SA TO
550 REM              USE MEMORY SPRITE SB
560 POKE 2040+SA,192+SB
570 REM              SET SPRITE COLOR
580 POKE S+39+SA,C1
590 REM  TELL MACHINE LANGUAGE PROGRAM
600 REM  WHICH SPRITE TO USE
610 POKE S1,SA
620 POKE S+21,2↑SA :       REM ENABLE SPRITE
630 GOSUB 1500
640 REM              SET VELOCITIES
650 POKE XV,VX
660 POKE YV,VY
670 GOSUB 1300
680 REM              SET INITIAL POSITION
690 POKE Y0,Y
700 POKE X0,X AND 255
710 POKE X0+1,INT(X/256)
720 REM              DISPLAY THE SPRITE
730 SYS DISP
780 :
790 REM  LINES 800-850 FLY THE SPRITE
791 REM  UNDER JOYSTICK CONTROL
800 SYS BE+102
850 GOTO 800
990 :
999 REM  INPUT MEMORY SPRITE SB
1000 INPUT "USE WHICH SPRITE (0 TO 63)";SB
1090 RETURN
1099 REM  INPUT COLOR OF DISPLAY SPRITE SA
1100 INPUT "DISPLAY WITH WHICH COLOR (0 TO 15)";C1
1190 RETURN
1199 REM  INPUT DISPLAY SPRITE SA
1200 INPUT "DISPLAY AS WHICH SPRITE (0 TO 7)";SA
1290 RETURN
1299 REM INPUT X,Y LOCATIONS
1300 PRINT"   X,Y LOCATIONS FOR SPRITE";SA;
1320 INPUT X,Y
1390 RETURN
1490 REM  INPUT VELOCITIES
```

```
1500 INPUT "X VELOCITY";VX
1520 INPUT "Y VELOCITY";VY
1530 VX = ABS(VX)
1540 VY = ABS(VY)
1590 RETURN
2000 :
2010  DATA  76,39,192,76,221,192
2020  DATA  169,1,174,6,192
2030  DATA  240,7,10,141,12,192
2040  DATA  202,208,249,96
2050  DATA  169
2060  DATA  0,141,7,192,141,8, 192,76
2070  DATA  168,192,169,1,141,8,192,169
2080  DATA  73,141,7,192,76,168,192,169
2090  DATA  0,141,8,192,173,0,220,73
2100  DATA  127,41,15,240,110,201,8,144
2110  DATA  24,72,173,7,192,24,109,10
2120  DATA  192,141,7,192,144,40,173
2130  DATA  8,192,208,207,169,1,141,8
2140  DATA  192,104,41,7,201,4,144,24
2150  DATA  72,173,7,192,56,237,10,192
2160  DATA  141,7,192,176,10,173,8,192
2170  DATA  240,166,169,0,141,8,192,104
2180  DATA  41,3,201,2,144,21,72,173
2190  DATA  9,192,24,109,11,192,141,9
2200  DATA  192,201,246,144
2210  DATA  30,169,246,141,9,192,104,41
2220  DATA  1,201,1,144,19,72,173,9
2230  DATA  192,56,237,11,192,141,9,192
2240  DATA  176,5,169,0,141,9,192,104
2250  DATA  173,6,192,10,170,173,7,192
2260  DATA  157,0,208,232,173,9,192,157
2270  DATA  0,208,32,58,192,73,255,45
2280  DATA  16,208,141,13,192,172,8,192
2290  DATA  240,7,173,12,192,24,109,13
2300  DATA  192,141,16,208,96
```

Joystick Sprite 3

This adds a device to detect collisions with text. Any PEEK(53279) that returns a nonzero value means a collision between a sprite and text that is on the screen. The following chart will tell you which sprite has collided:

PEEK(53279)	*DISPLAY SPRITE*
0	No collision
1	sprite 0
2	sprite 1
4	sprite 2

PEEK(53279)	*DISPLAY SPRITE*
8	sprite 3
16	sprite 4
32	sprite 5
64	sprite 6
128	sprite 7

```
5 REM         JOYSTICK SPRITE3
6 REM  THIS IS JOYSTICK SPRITE2  WITH
7 REM  LINES 105-110, 792, AND
8 REM  810-840 ADDED
100 S = 53248   :          REM VIC REGISTERS
105 A = PEEK(S+31) :       REM INIT COLLISION
110 CR = 0  :              REM INIT CRASHES
150 REM  LINES 190 TO 260
160 REM  READ IN THE MACHINE LANGUAGE
170 REM  ROUTINE THAT FLIES THE SPRITE
180 REM  UNDER JOYSTICK CONTROL
190 FOR I = 49152 TO 49157
200 READ A
210 POKE I,A
220 NEXT I
230 FOR I = 49210 TO 49417
240 READ A
250 POKE I,A
260 NEXT I
300 :
310 REM  SET UP LOCATIONS FOR POKEING
320 REM  DATA TO THE MACHINE LANGUAGE
330 REM  ROUTINE THAT FLIES THE SPRITE
340 BEGIN = 12*4096
350 S1 = BE + 6
360 X0 = BE + 7
370 Y0 = BE + 9
380 XV = BE + 10
390 YV = BE + 11
400 DISP = BE + 3
490 :
500 REM  GET NEEDED SPRITE DATA
510 GOSUB 1000
520 GOSUB 1100
530 GOSUB 1200
540 REM       SET DISPLAY SPRITE SA TO
550 REM             USE MEMORY SPRITE SB
560 POKE 2040+SA,192+SB
570 REM             SET SPRITE COLOR
580 POKE S+39+SA,C1
590 REM  TELL MACHINE LANGUAGE PROGRAM
600 REM  WHICH SPRITE TO USE
```

```
610 POKE S1,SA
620 POKE S+21,2↑SA :       REM ENABLE SPRITE
630 GOSUB 1500
640 REM                SET VELOCITIES
650 POKE XV,VX
660 POKE YV,VY
670 GOSUB 1300
680 REM                SET INITIAL POSITION
690 POKE Y0,Y
700 POKE X0,X AND 255
710 POKE X0+1,INT(X/256)
720 REM                DISPLAY THE SPRITE
730 SYS DISP
780 :
790 REM  LINES 800-850 FLY THE SPRITE
791 REM  UNDER JOYSTICK CONTROL
792 REM  CHECKING FOR COLLISIONS
800 SYS BE+102
810 A = PEEK(53279)
840 IF A<>0 THEN CR=CR+1:PRINT"CRASH ";CR
850 GOTO 800
990 :
999 REM  INPUT MEMORY SPRITE SB
1000 INPUT "USE WHICH SPRITE (0 TO 63)";SB
1090 RETURN
1099 REM  INPUT COLOR OF DISPLAY SPRITE SA
1100 INPUT "DISPLAY WITH WHICH COLOR (0 TO 15)";C1
1190 RETURN
1199 REM  INPUT DISPLAY SPRITE SA
1200 INPUT "DISPLAY AS WHICH SPRITE (0 TO 7)";SA
1290 RETURN
1299 REM INPUT X,Y LOCATIONS
1300 PRINT"   X,Y LOCATIONS FOR SPRITE";SA;
1320 INPUT X,Y
1390 RETURN
1490 REM  INPUT VELOCITIES
1500 INPUT "X VELOCITY";VX
1520 INPUT "Y VELOCITY";VY
1530 VX = ABS(VX)
1540 VY = ABS(VY)
1590 RETURN
2000 :
2010  DATA  76,39,192,76,221,192
2020  DATA  169,1,174,6,192
2030  DATA  240,7,10,141,12,192
2040  DATA  202,208,249,96
2050  DATA  169
2060  DATA  0,141,7,192,141,8, 192,76
2070  DATA  168,192,169,1,141,8,192,169
2080  DATA  73,141,7,192,76,168,192,169
2090  DATA  0,141,8,192,173,0,220,73
2100  DATA  127,41,15,240,110,201,8,144
```

```
2110  DATA  24,72,173,7,192,24,109,10
2120  DATA  192,141,7,192,144,40,173
2130  DATA  8,192,208,207,169,1,141,8
2140  DATA  192,104,41,7,201,4,144,24
2150  DATA  72,173,7,192,56,237,10,192
2160  DATA  141,7,192,176,10,173,8,192
2170  DATA  240,166,169,0,141,8,192,104
2180  DATA  41,3,201,2,144,21,72,173
2190  DATA  9,192,24,109,11,192,141,9
2200  DATA  192,201,246,144
2210  DATA  30,169,246,141,9,192,104,41
2220  DATA  1,201,1,144,19,72,173,9
2230  DATA  192,56,237,11,192,141,9,192
2240  DATA  176,5,169,0,141,9,192,104
2250  DATA  173,6,192,10,170,173,7,192
2260  DATA  157,0,208,232,173,9,192,157
2270  DATA  0,208,32,58,192,73,255,45
2280  DATA  16,208,141,13,192,172,8,192
2290  DATA  240,7,173,12,192,24,109,13
2300  DATA  192,141,16,208,96
```

Joystick Sprite 4

We add two lines to JOYSTICK SPRITE 3—820 and 830. Now if the sprite hits the text, no message is printed unless you have the paddle button pressed.

```
5 REM         JOYSTICK SPRITE4
6 REM  THIS IS JOYSTICK SPRITE3  WITH
7 REM  LINES  820-830  ADDED
100 S = 53248   :          REM VIC REGISTERS
105 A = PEEK(S+31) :       REM INIT COLLISION
110 CR = 0  :              REM INIT CRASHES
150 REM  LINES 190 TO 260
160 REM  READ IN THE MACHINE LANGUAGE
170 REM  ROUTINE THAT FLIES THE SPRITE
180 REM  UNDER JOYSTICK CONTROL
190 FOR I = 49152 TO 49157
200 READ A
210 POKE I,A
220 NEXT I
230 FOR I = 49210 TO 49417
240 READ A
250 POKE I,A
260 NEXT I
300 :
310 REM  SET UP LOCATIONS FOR POKEING
320 REM  DATA TO THE MACHINE LANGUAGE
330 REM  ROUTINE THAT FLIES THE SPRITE
340 BEGIN = 12*4096
350 S1 = BE + 6
```

```
360 X0 = BE + 7
370 Y0 = BE + 9
380 XV = BE + 10
390 YV = BE + 11
400 DISP = BE + 3
490 :
500 REM  GET NEEDED SPRITE DATA
510 GOSUB 1000
520 GOSUB 1100
530 GOSUB 1200
540 REM        SET DISPLAY SPRITE SA TO
550 REM              USE MEMORY SPRITE SB
560 POKE 2040+SA,192+SB
570 REM              SET SPRITE COLOR
580 POKE S+39+SA,C1
590 REM  TELL MACHINE LANGUAGE PROGRAM
600 REM  WHICH SPRITE TO USE
610 POKE S1,SA
620 POKE S+21,2↑SA :       REM ENABLE SPRITE
630 GOSUB 1500
640 REM              SET VELOCITIES
650 POKE XV,VX
660 POKE YV,VY
670 GOSUB 1300
680 REM              SET INITIAL POSITION
690 POKE Y0,Y
700 POKE X0,X AND 255
710 POKE X0+1,INT(X/256)
720 REM              DISPLAY THE SPRITE
730 SYS DISP
780 :
790 REM  LINES 800-820 FLY THE SPRITE
791 REM  UNDER JOYSTICK CONTROL
792 REM  CHECKING FOR COLLISIONS
800 SYS BE+102
810 A = PEEK(S+31)
820 BU = PEEK(56320) AND 16
830 IF BU =16 THEN 800
840 IF A<>0 THEN CR=CR+1:PRINT"CRASH ";CR
850 GOTO 800
990 :
999 REM  INPUT MEMORY SPRITE SB
1000 INPUT "USE WHICH SPRITE (0 TO 63)";SB
1090 RETURN
1099 REM  INPUT COLOR OF DISPLAY SPRITE SA
1100 INPUT "DISPLAY WITH WHICH COLOR (0 TO 15)";C1
1190 RETURN
1199 REM  INPUT DISPLAY SPRITE SA
1200 INPUT "DISPLAY AS WHICH SPRITE (0 TO 7)";SA
1290 RETURN
1299 REM INPUT X,Y LOCATIONS
```

```
1300 PRINT"   X,Y LOCATIONS FOR SPRITE";SA;
1320 INPUT X,Y
1390 RETURN
1490 REM  INPUT VELOCITIES
1500 INPUT "X VELOCITY";VX
1520 INPUT "Y VELOCITY";VY
1530 VX = ABS(VX)
1540 VY = ABS(VY)
1590 RETURN
2000 :
2010  DATA  76,39,192,76,221,192
2020  DATA  169,1,174,6,192
2030  DATA  240,7,10,141,12,192
2040  DATA  202,208,249,96
2050  DATA  169
2060  DATA  0,141,7,192,141,8, 192,76
2070  DATA  168,192,169,1,141,8,192,169
2080  DATA  73,141,7,192,76,168,192,169
2090  DATA  0,141,8,192,173,0,220,73
2100  DATA  127,41,15,240,110,201,8,144
2110  DATA  24,72,173,7,192,24,109,10
2120  DATA  192,141,7,192,144,40,173
2130  DATA  8,192,208,207,169,1,141,8
2140  DATA  192,104,41,7,201,4,144,24
2150  DATA  72,173,7,192,56,237,10,192
2160  DATA  141,7,192,176,10,173,8,192
2170  DATA  240,166,169,0,141,8,192,104
2180  DATA  41,3,201,2,144,21,72,173
2190  DATA  9,192,24,109,11,192,141,9
2200  DATA  192,201,246,144
2210  DATA  30,169,246,141,9,192,104,41
2220  DATA  1,201,1,144,19,72,173,9
2230  DATA  192,56,237,11,192,141,9,192
2240  DATA  176,5,169,0,141,9,192,104
2250  DATA  173,6,192,10,170,173,7,192
2260  DATA  157,0,208,232,173,9,192,157
2270  DATA  0,208,32,58,192,73,255,45
2280  DATA  16,208,141,13,192,172,8,192
2290  DATA  240,7,173,12,192,24,109,13
2300  DATA  192,141,16,208,96
```

Sprite Racer

To use this program, you first need to design your race car with SPRITE EDITOR. You will need to remember the correct memory sprite number to use your race car in this program. The race begins with your racer to the left of the start–finish line. You should proceed counterclockwise around the race track, thus starting by moving to the left. Use the joystick to move your SPRITE RACER. To go faster, press the fire button on your joystick. Each

time you press it, your speed increases. By letting go of your joystick, the racer will stop. To reset speeds to their starting levels, press the space bar. When you finish the race, your elapsed time will be shown on the screen.

To change the design of the race track, change lines 1711–1733. You can use any letter, number, or graphics symbol to define the edges of your race track. Any of these symbols can be printed in any of the colors available on the 64. These symbols can also be placed as obstacles on your race track.

```
5 REM         SPRITE RACER
6 REM   THIS IS A MAJOR REVISION OF
7 REM   JOYSTICK SPRITE3
8 REM   LINES 100-520 AND THE DATA
9 REM   STATEMENTS (LINES 3000 ON)
10 REM ARE IDENTICAL
11 REM THE REMAINING LINES ARE NEW OR
12 REM CHANGED
50 DIM R$(24)
60 DEF FNT(X) = (INT(100*(TI-X)/60)/100)
90 :
100 S = 53248   :          REM VIC REGISTERS
110 A = PEEK(S+31) :       REM INIT COLLISION
120 CR = 0  :              REM INIT CRASHES
130 POKE 53281,1   :       REM WHITE SCREEN
150 REM         LINES 190 TO 260
160 REM         READ IN THE MACHINE LANGUAGE
170 REM         ROUTINE THAT FLIES THE SPRITE
180 REM         UNDER JOYSTICK CONTROL
190 FOR I = 49152 TO 49157
200 READ A
210 POKE I,A
220 NEXT I
230 FOR I = 49210 TO 49417
240 READ A
250 POKE I,A
260 NEXT I
300 :
310 REM         SET UP LOCATIONS FOR POKEING
320 REM         DATA TO THE MACHINE LANGUAGE
330 REM         ROUTINE THAT FLIES THE SPRITE
340 BEGIN = 12*4096
350 S1 = BE + 6
360 X0 = BE + 7
370 Y0 = BE + 9
380 XV = BE + 10
390 YV = BE + 11
400 DISP = BE + 3
490 :
500 REM         GET NEEDED SPRITE DATA
510 GOSUB 1000
520 GOSUB 1100
```

```
530 SA = 0
540 REM         SET DISPLAY SPRITE SA TO
550 REM         USE MEMORY SPRITE SB
560 POKE 2040+SA,192+SB
570 REM         SET SPRITE COLOR
580 POKE S+39+SA,C1
590 REM         TELL MACHINE LANGUAGE PROGRAM
600 REM         WHICH SPRITE TO USE
610 POKE S1,SA
615 POKE S+29,0     :       REM CONTRACT X
620 POKE S+23,0     :       REM CONTRACT Y
640 REM         SET VELOCITIES
650 GOSUB 2100      :       REM INIT SPEED
660 GOSUB 2400      :       REM INITIALIZE
740 GOSUB 1700      :       REM DRAW COURSE
750 T0 = TI         :       REM INIT TIME
780 :
790 REM         LINES 800-850 FLY THE SPRITE
791 REM         UNDER JOYSTICK CONTROL
792 REM         CHECKING FOR COLLISIONS
800 SYS BE+102
820 A1 = PEEK(S+30):        REM SPRITE-SPRITE
830 IF A1=3 THEN 900:       REM END OF RACE
840 IF PEEK(S+31)<>0 THEN  GOSUB 1600
850 A2 = PEEK(56320):       REM READ JOYSTICK
860 IF (A2 AND 16)=0 THEN GOSUB 2000
870 GET A$          :       REM CHECK BRAKE
880 IF A$ = " " THEN GOSUB 2100
890 GOTO 800
900 T = FNT(T0)     :       REM GET TIME
910 PRINT CHR$(147)
920 PRINT "ELAPSED TIME ";T;" SECONDS"
930 PRINT:PRINT "GO AGAIN ? ";
940 GET A$
950 IF A$="" THEN 940
960 IF A$="Y" THEN 610
980 END
990 :
995 REM         INPUT MEMORY SPRITE SB
1000 INPUT "USE WHICH SPRITE (0 TO 63)";SB
1090 RETURN
1099 REM        INPUT COLOR OF DISPLAY SPRITE SA
1100 INPUT "DISPLAY WITH WHICH COLOR (0 TO 15)";C1
1190 RETURN
1590 REM        FLASH SPRITE SA
1600 LC = S + 39 + SA
1610 CA = PEEK(LC)  :       REM  SPRITE COLOR
1620 CZ = (CA + 1) AND 16
1630 FOR I = 1 TO 20
1640 POKE LC,CZ
1650 POKE LC,CA
1660 NEXT
```

```
1670 FOR I = 1 TO 100
1680 NEXT
1690 RETURN
1695 REM        SETUP UP RACE COURSE
1700 PRINT CHR$(147)
1705 REM        THESE ARE 39 WIDE
1711 R$( 1) = ":---------------------------------------:"
1712 R$( 2) = ":                                       :"
1713 R$( 3) = ":                                       :"
1714 R$( 4) = ":     .--------------------------.      :"
1715 R$( 5) = ":     :                          :      :"
1716 R$( 6) = ":     :                          :      :"
1717 R$( 7) = ":     :                          :      :"
1718 R$( 8) = ":     :                          :      :"
1719 R$( 9) = ":     :                          :      :"
1720 R$(10) = ":     :                          :      :"
1721 R$(11) = ":     :                          :      :"
1722 R$(12) = ":     :                          :      :"
1723 R$(13) = ":     :                          :      :"
1724 R$(14) = ":     :                          :      :"
1725 R$(15) = ":     :                          :      :"
1726 R$(16) = ":     :                          :      :"
1727 R$(17) = ":     :                          :      :"
1728 R$(18) = ":     :                          :      :"
1729 R$(19) = ":     :                          :      :"
1730 R$(20) = ":     '--------------------------'      :"
1731 R$(21) = ":                                       :"
1732 R$(22) = ":                                       :"
1733 R$(23) = ":---------------------------------------:"
1740 FOR I = 1 TO 23
1750 IF LEN(R$(I))=39 THEN 1780
1755 PRINT"LENGTH ERROR IN LINE ";1710+I
1760 PRINT "LENGTH SHOULD BE 39, BUT IS NOW ";LEN(R$(I))
1770 STOP
1780 PRINT R$(I)
1790 NEXT
1795 :
1796 REM        THIS DRAWS THE FINISH LINE
1800 S2 = 832
1810 POKE S+16,0
1820 FOR I = 0 TO 63
1830 POKE S2 + I,0
1840 NEXT
1850 FOR I = 0 TO 60 STEP 3
1860 POKE S2 + I,15
1870 NEXT
1880 POKE 2041,13
1890 POKE S+2,180
1900 POKE S+3,63
1910 POKE S+30,0
1920 POKE S+21,3    :      REM ENABLE SPRITE
```

```
1990 :
2000 VX = VX + 1
2010 VY = VY + 1
2020 POKE XV,VX
2030 POKE YV,VY
2090 RETURN
2099 REM        RESET SPEED
2100 VX = 1
2110 VY = 1
2120 GOTO 2020
2390 REM        INITIALIZATION ROUTINE
2400 X = 150
2410 Y = 63
2420 REM        SET INITIAL POSITION
2430 POKE Y0,Y
2440 POKE X0,X AND 255
2450 POKE X0+1,INT(X/256)
2460 REM        DISPLAY THE SPRITE
2470 SYS DISP
2480 RETURN
3000 :
3010  DATA  76,39,192,76,221,192
3020  DATA  169,1,174,6,192
3030  DATA  240,4,10,202,208,252
3040  DATA  141,12,192,96
3050  DATA  169
3060  DATA  0,141,7,192,141,8,  192,76
3070  DATA  168,192,169,1,141,8,192,169
3080  DATA  73,141,7,192,76,168,192,169
3090  DATA  0,141,8,192,173,0,220,73
3100  DATA  127,41,15,240,110,201,8,144
3110  DATA  24,72,173,7,192,24,109,10
3120  DATA  192,141,7,192,144,40,173
3130  DATA  8,192,208,207,169,1,141,8
3140  DATA  192,104,41,7,201,4,144,24
3150  DATA  72,173,7,192,56,237,10,192
3160  DATA  141,7,192,176,10,173,8,192
3170  DATA  240,166,169,0,141,8,192,104
3180  DATA  41,3,201,2,144,21,72,173
3190  DATA  9,192,24,109,11,192,141,9
3200  DATA  192,201,246,144
3210  DATA  30,169,246,141,9,192,104,41
3220  DATA  1,201,1,144,19,72,173,9
3230  DATA  192,56,237,11,192,141,9,192
3240  DATA  176,5,169,0,141,9,192,104
3250  DATA  173,6,192,10,170,173,7,192
3260  DATA  157,0,208,232,173,9,192,157
3270  DATA  0,208,32,58,192,73,255,45
3280  DATA  16,208,141,13,192,172,8,192
3290  DATA  240,7,173,12,192,24,13,13
3300  DATA  192,141,16,208,96
```

Sprite Racer 1

This program adds a speed governor, a much heavier penalty for crashing, and a new race course for your racing pleasure. The speed governor is in line 860 and limits speed to 5 no matter how many times you hit the fire button. In line 840, a collision detection now causes a restart, which should slow you down a bit. The new course is more difficult and shows how to create problems for the racer.

```
5 REM         SPRITE RACER 1
6 REM  THIS IS SPRITE RACER WITH
7 REM  LINES 840 AND 860 CHANGED
8 REM  AS WELL AS A NEW COURSE IN
9 REM  LINES 1711-1733
50 DIM R$(24)
60 DEF FNT(X) = (INT(100*(TI-X)/60)/100)
90 :
100 S = 53248   :          REM VIC REGISTERS
110 A = PEEK(S+31) :       REM INIT COLLISION
120 CR = 0  :              REM INIT CRASHES
130 POKE 53281,1   :       REM WHITE SCREEN
150 REM          LINES 190 TO 260
160 REM          READ IN THE MACHINE LANGUAGE
170 REM          ROUTINE THAT FLIES THE SPRITE
180 REM          UNDER JOYSTICK CONTROL
190 FOR I = 49152 TO 49157
200 READ A
210 POKE I,A
220 NEXT I
230 FOR I = 49210 TO 49417
240 READ A
250 POKE I,A
260 NEXT I
300 :
310 REM          SET UP LOCATIONS FOR POKEING
320 REM          DATA TO THE MACHINE LANGUAGE
330 REM          ROUTINE THAT FLIES THE SPRITE
340 BEGIN = 12*4096
350 S1 = BE + 6
360 X0 = BE + 7
370 Y0 = BE + 9
380 XV = BE + 10
390 YV = BE + 11
400 DISP = BE + 3
490 :
500 REM          GET NEEDED SPRITE DATA
510 GOSUB 1000
520 GOSUB 1100
530 SA = 0
540 REM          SET DISPLAY SPRITE SA TO
550 REM          USE MEMORY SPRITE SB
```

```
560 POKE 2040+SA,192+SB
570 REM         SET SPRITE COLOR
580 POKE S+39+SA,C1
590 REM         TELL MACHINE LANGUAGE PROGRAM
600 REM         WHICH SPRITE TO USE
610 POKE S1,SA
615 POKE S+29,0     :       REM CONTRACT X
620 POKE S+23,0     :       REM CONTRACT Y
640 REM         SET VELOCITIES
650 GOSUB 2100      :       REM INIT SPEED
660 GOSUB 2400      :       REM INITIALIZE
740 GOSUB 1700      :       REM DRAW COURSE
750 T0 = TI         :       REM INIT TIME
780 :
790 REM         LINES 800-850 FLY THE SPRITE
791 REM         UNDER JOYSTICK CONTROL
792 REM         CHECKING FOR COLLISIONS
800 SYS BE+102
820 A1 = PEEK(S+30):        REM SPRITE-SPRITE
830 IF A1=3 THEN 900:       REM END OF RACE
840 IF PEEK(S+31)<>0 THEN  GOSUB 2100: GOSUB 2400:GOTO 800
850 A2 = PEEK(56320):       REM READ JOYSTICK
860 IF (A2 AND 16)=0 AND VX<5 THEN GOSUB 2000
870 GET A$          :       REM CHECK BRAKE
880 IF A$ = " " THEN GOSUB 2100
890 GOTO 800
900 T = FNT(T0)     :       REM GET TIME
910 PRINT CHR$(147)
920 PRINT "ELAPSED TIME ";T;" SECONDS"
930 PRINT:PRINT "GO AGAIN ? ";
940 GET A$
950 IF A$="" THEN 940
960 IF A$="Y" THEN 610
980 END
990 :
995 REM         INPUT MEMORY SPRITE SB
1000 INPUT "USE WHICH SPRITE (0 TO 63)";SB
1090 RETURN
1099 REM        INPUT COLOR OF DISPLAY SPRITE SA
1100 INPUT "DISPLAY WITH WHICH COLOR (0 TO 15)";C1
1190 RETURN
1590 REM        FLASH SPRITE SA
1600 LC = S + 39 + SA
1610 CA = PEEK(LC)  :       REM  SPRITE COLOR
1620 CZ = (CA + 1) AND 16
1630 FOR I = 1 TO 20
1640 POKE LC,CZ
1650 POKE LC,CA
1660 NEXT
1670 FOR I = 1 TO 100
1680 NEXT
1690 RETURN
```

```
1695 REM        SETUP UP RACE COURSE
1700 PRINT CHR$(147)
1705 REM        THESE ARE 39 WIDE
1711 R$( 1) = ":---------------------------------------:"
1712 R$( 2) = ":                                       :"
1713 R$( 3) = ":                                       :"
1714 R$( 4) = ":      .------------------------.       :"
1715 R$( 5) = ":     :                          :      :"
1716 R$( 6) = ":      '-------------------.      :      :"
1717 R$( 7) = ":                         :      :      :"
1718 R$( 8) = ":                        :       :      :"
1719 R$( 9) = ":--------------------.    :       :      :"
1720 R$(10) = ":                    :    :       :      :"
1721 R$(11) = ":                     :   :       :      :"
1722 R$(12) = ":---------------------'   :       :      :"
1723 R$(13) = ":                        :       :      :"
1724 R$(14) = ":                       :        :      :"
1725 R$(15) = ":      .-----------------:       :      :"
1726 R$(16) = ":     :                          :      :"
1727 R$(17) = ":     :                          :      :"
1728 R$(18) = ":     :                          :      :"
1729 R$(19) = ":      :                        :       :"
1730 R$(20) = ":       '------------------------'      :"
1731 R$(21) = ":                                       :"
1732 R$(22) = ":                                       :"
1733 R$(23) = ":---------------------------------------:"
1740 FOR I = 1 TO 23
1750 IF LEN(R$(I))=39 THEN 1780
1755 PRINT"LENGTH ERROR IN LINE ";1710+I
1760 PRINT "LENGTH SHOULD BE 39, BUT IS NOW ";LEN(R$(I))
1770 STOP
1780 PRINT R$(I)
1790 NEXT
1795 :
1796 REM        THIS DRAWS THE FINISH LINE
1800 S2 = 832
1810 POKE S+16,0
1820 FOR I = 0 TO 63
1830 POKE S2 + I,0
1840 NEXT
1850 FOR I = 0 TO 60 STEP 3
1860 POKE S2 + I,15
1870 NEXT
1880 POKE 2041,13
1890 POKE S+2,180
1900 POKE S+3,63
1910 POKE S+30,0
1920 POKE S+21,3    :      REM ENABLE SPRITE
1990 :
2000 VX = VX + 1
```

```
2010 VY = VY + 1
2020 POKE XV,VX
2030 POKE YV,VY
2090 RETURN
2099 REM        RESET SPEED
2100 VX = 1
2110 VY = 1
2120 GOTO 2020
2390 REM        INITIALIZATION ROUTINE
2400 X = 150
2410 Y = 63
2420 REM        SET INITIAL POSITION
2430 POKE Y0,Y
2440 POKE X0,X AND 255
2450 POKE X0+1,INT(X/256)
2460 REM        DISPLAY THE SPRITE
2470 SYS DISP
2480 RETURN
3000 :
3010  DATA  76,39,192,76,221,192
3020  DATA  169,1,174,6,192
3030  DATA  240,4,10,202,208,252
3040  DATA  141,12,192,96
3050  DATA  169
3060  DATA  0,141,7,192,141,8, 192,76
3070  DATA  168,192,169,1,141,8,192,169
3080  DATA  73,141,7,192,76,168,192,169
3090  DATA  0,141,8,192,173,0,220,73
3100  DATA  127,41,15,240,110,201,8,144
3110  DATA  24,72,173,7,192,24,109,10
3120  DATA  192,141,7,192,144,40,173
3130  DATA  8,192,208,207,169,1,141,8
3140  DATA  192,104,41,7,201,4,144,24
3150  DATA  72,173,7,192,56,237,10,192
3160  DATA  141,7,192,176,10,173,8,192
3170  DATA  240,166,169,0,141,8,192,104
3180  DATA  41,3,201,2,144,21,72,173
3190  DATA  9,192,24,109,11,192,141,9
3200  DATA  192,201,246,144
3210  DATA  30,169,246,141,9,192,104,41
3220  DATA  1,201,1,144,19,72,173,9
3230  DATA  192,56,237,11,192,141,9,192
3240  DATA  176,5,169,0,141,9,192,104
3250  DATA  173,6,192,10,170,173,7,192
3260  DATA  157,0,208,232,173,9,192,157
3270  DATA  0,208,32,58,192,73,255,45
3280  DATA  16,208,141,13,192,172,8,192
3290  DATA  240,7,173,12,192,24,13,13
3300  DATA  192,141,16,208,96
```

Challenges

1. Add additional sprite routines to the CHANGE SPRITE program.
2. Add visual displays or sound effects for collisions in SPRITE RACER.
3. Combine SPRITE EDITOR, SAVE/LOAD SPRITE, CHANGE SPRITE, and DISABLE SPRITE into one program to do it all.
4. Add a random obstacle generator for your SPRITE RACER.
5. Put four or more sprites on the screen at one time and control them from the keyboard.
6. Try putting together SPRITE RACER with a maze-generating program, such as LABYRINTH on the disk/cassette Bonus Pack. (The Bonus Pack is a valuable collection of programs sold by Commodore at a very reasonable price. Check with your local retailer.)
7. Add a routine based on SAVE/LOAD SPRITE to SPRITE RACER to read the race course from tape or disk. A simple race editor based on SPRITE EDITOR that saves courses to tape or disk would be neat.
8. Add sound to your SPRITE RACER or JOYSTICK SPRITE.
9. Modify SPRITE RACER 1 so that a fast crash returns you to the starting point, but a softer crash does not.
10. *SUPER CHALLENGE.* Make an animated movie with your sprites. Add music or sound effects to the animation.

8

BELLS AND WHISTLES (AND A BIT MORE)

Programs:

BUZZER	Someone's at the door.
BELL	As in ding-dong.
WILD SOUND	Notes, notes everywhere.
BEEP	It does what it says.
COMPUTER TALK	This is the way computers talk in the movies.
AMBULANCE	It gets closer and then goes away.
SOUND EFFECTS	Random sound effects.
SCALE	Every note on the way up.
SCALE 2	The Do–Re–Mi kind.
MUSIC BOX	Many bells—random three-part harmony.
MARCHING FEET	Tramp . . . tramp . . . tramp.
MIDNIGHT	The clock strikes midnight at the graveyard.
BUSY SIGNAL	This is just like calling "Information."
COP AT CORNER	The other kind of siren.
WOLF WHISTLE	The only whistle in this chapter.
NOISY LOOP	Shows what happens when you POKE a variety of waveforms.

CHORD ORGAN	Three-tone chords—eight different chords played from the keyboard.
VOICE MAKER	Makes every sound that the 64 can make.
NOTE FREQUENCIES	The hertz you get for the POKE you make and vice versa.

The programs in this chapter will make your 64 into a music synthesizer. By the time you have completed VOICE MAKER, you and your music synthesizer will be able to do the following:

1. Make a variety of simple sounds, such as beeps, buzzers, and bells.
2. Create random sequences of sounds.
3. Create more complicated sound effects, such as sirens, ray guns, and UFOs.
4. Create random chords using a bell as the instrument played.
5. Understand how to create the sounds of a variety of musical instruments.
6. See how to create a playable musical instrument: a six-octave chord organ.
7. Control every feature of the music synthesizer in the 64.

Making Sound

Let's make a simple sound. In immediate mode, type the following:

```
SI = 54272
POKE SI,97
POKE SI+1,8
POKE SI+5,0
POKE SI+6,240
POKE SI+24,15
POKE SI+4,17
```

You should now be hearing a low-pitched tone. If not, check the volume control on your TV set and then check your typing to be sure you got it right. To stop the sound, simply type:

```
POKE SI+4,0
```

What did these POKE commands do? We will try to explain these and the many other features of the musical ability of the 64. All of the 64's sound ability comes from the Sound Interface Device (SID) chip in the 64. The POKE commands above told the SID to make a sound. Briefly, the first two

POKEs told what frequency; the next two told about the ADSR (discussion about ADSR follows later in this section); the next the volume; and the last the type of sound. The final POKE turned the sound off. We begin the detailed discussion with frequency.

Frequency

An essential command to the SID chip is the particular frequency, or pitch, that you wish to hear. In all of our programs, we let the variable SI be the starting memory location of the SID chip, which is 54272. The POKEs that set the frequency are those to SI and SI+1. Try changing these numbers. For example, try:

```
POKE SI+4,17
POKE SI,200
POKE SI+1,200
```

Remember that POKE SI+4,0 will turn it off.

The first POKE turned the sound on (more about this later). The second POKE raised the pitch slightly. The third POKE put it way up there. Both SI and SI+1 control the frequency, but the effect of a single unit change in SI+1 is equal to a change of 256 in SI. Thus, changes in SI act as fine-tuning, while coarser changes are made by SI+1. From Appendix M of the *Commodore 64 User's Guide,* we can get some idea of the meanings of different POKE values:

SI Low Freq	*SI+1 High Freq*	*Note Octave*	*256*(High Freq) + (Low Freq)*	*Actual Frequency in Hertz*
97	8	C–3	2145	131
225	8	C#–3	2273	139
104	9	D–3	2408	147
195	16	C–4	4291	263
243	200	G–7	51443	3136

The actual frequency will be explained below under "More Detail." For non-music buffs, C–3 means a third octave C and C#–3 means a third octave C sharp. The values in the columns SI and SI+1 are the values that need to be POKEd to SI and SI+1. The column titled *256*(High Freq) + (Low Freq)* gives the result of multiplying 256 times the second column plus the first column. In our programs, we use this variable (value) to keep track of the frequency, and then split it into two parts, *High Freq* and *Low Freq,* to POKE to the SID chip. To hear the effect of different frequencies, you might want to type in the program SCALE and run it. In SCALE we use the variable N to

keep the value of the frequency and lines 210 and 220 to separate the variable N into the two parts to POKE to SI and SI+1.

Volume

A POKE to SI+24 is your volume control. Start the tone again and change the volume by poking different values to SI+24 (we first lower the frequency to keep from going crazy with the high pitch):

```
POKE SI+1,8
POKE SI+4,17
POKE SI+24,5
POKE SI+24,0
POKE SI+24,15
```

The allowable settings for this control are from 0 to 15 only. You will hear a distinct click every time that you change the setting of this location while the SID chip is producing sound.

ADSR Envelope

We mentioned that the POKEs to SI+5 and SI+6 controlled the ADSR, which stands for Attack, Decay, Sustain, and Release. When a sound, any sound, is made, it begins at zero volume and then increases to a maximum level. The time it takes to do this is called attack time. The loudness then tends to drop off to another level. The time it takes to drop off is decay time. The level it drops to is called the sustain level. Some sounds can then maintain the sustain level for a long time and others a short time. In any event, you directly control this, and there is no name for the length of time that the sustain level is maintained. Sounds finally go silent, and the time it takes to go from the sustain level of volume to zero volume is called the release time.

Music theorists like to call the attack time, decay time, sustain level, and release time an ADSR envelope (for reasons most of us might question, since you certainly cannot put letters into an ADSR envelope). They also claim that one major way that sounds differ, from piano to violin, trumpet to drum, can be characterized by their ADSR envelope. Different sounds, those from different musical instruments, have a characteristic envelope. For example, a bell has a very short attack, very short decay, a high sustain volume, and a very long release. A trumpet, on the other hand, has a longer attack, a longer decay, and a very short release time—when you stop blowing, it stops making any sound.

We can set up the SID chip to sound somewhat like each of these instruments and many other sounds besides. The POKE to SI+5 controls the attack/decay, while the POKE to SI+6 controls the sustain volume/

release. This saves memory (only one location for two things) but complicates the telling. Each of A, D, S, and R can take on a value from 0 to 15. To tell SID the proper A/D, you actually need to POKE to memory location SI+5 with 16 times the attack time plus the delay time, as we do in the following:

```
A = 6
D = 0
POKE SI+5,16*A + D
POKE SI+4,17
POKE SI+4,16
```

The same needs to be done for sustain/release, as in:

```
S = 0
R = 0
POKE SI+6,16*S + R
POKE SI+4,17
```

This should sound similar to a flute (but do not be too demanding about hearing a flute). Some other suggested values are:

Instrument	A	D	S	R
Organ	0	0	12	0
Accordion	6	6	0	0
Bell	0	0	15	12

Remember that TVs differ greatly in their sound, so these instruments are only suggestions. You may want to experiment with other values. One easy way is to get your screen to look like:

```
A= 6: D= 6: POKE SI+5, 16*A + D
READY.
S= 0: R= 0; POKE SI+6, 16*S + D
READY.
POKE SI+4,17
READY.
POKE SI+4,16
READY.
```

Then use the cursor editing keys to make changes and hit return to reenter the line (the READY. is the 64 response, not your typing).

The program VOICE MAKER is the other way to experiment with making sounds. It makes keyboard control of all the features of the SID chip easy. So if you like experimenting with sounds, then you may want to start typing it in now. It's long, and most likely you will want to type it in in two or more sessions. Come back and read more when you get tired of typing.

The following table shows what the attack/decay/release times and sustain values are (in percentage of the volume in SI+24) for what you POKE.

Value of A,D,S,R	*Attack Time*	*Decay Time*	*Release Time*	*Percent Sustain Level*
	(all times in seconds)			
0	.002	.006	.006	0%
1	.008	.024	.024	7%
2	.016	.048	.048	13%
3	.024	.072	.072	20%
4	.038	.114	.114	27%
5	.056	.168	.168	33%
6	.068	.204	.204	40%
7	.080	.240	.240	47%
8	.100	.300	.300	53%
9	.250	.750	.750	60%
10	.500	1.500	1.500	67%
11	.800	2.400	2.400	73%
12	1.000	3.000	3.000	80%
13	3.000	9.000	9.000	87%
14	5.000	15.000	15.000	93%
15	8.000	24.000	24.000	100%

Waveform

Sound in the 64 started and stopped by POKEing information into memory location SI+4. An odd number starts the attack/decay sequence and an even number starts the release sequence. This location also controls which of the four possible waveforms will be used to generate the sound: triangle (17,16), sawtooth (33,32), pulse (65,64), and noise (129,128). The first number will start the attack/decay sequence of the sound, and the second (even) number will start the release sequence. For most sounds, you want the attack/decay to have the same waveform as the release, but some rather weird sounds can be made by having one waveform for the attack/decay and another one for the release (something that is not allowed in VOICE MAKER). For example, try the following:

```
A=10: D= 6: POKE SI+5, 16*A + D
S= 6: R≠12: POKE SI+6, 16*S + D
POKE SI+4,17
POKE SI+4,32
```

Be sure to wait for the attack/decay cycle to complete before hitting return on the last line to start the release cycle, or it will not work.

The triangle waveform sounds soft and full, whereas the sawtooth is more tinny, and the pulse can be changed all the way from tinny to smooth. The way to change the sound of the pulse waveform is by POKEing different numbers to locations SI+2 and SI+3. Just as with frequency, where there was a fine and coarse adjustment, SI+3 is the coarse adjustment to the pulse waveform and can range from 0 to 16, whereas SI+2 is the fine adjustment and can range from 0 to 255. Again, experiment with different values, because a single sound is probably worth the proverbial thousand words. Try the following:

```
POKE SI+5,0
POKE SI+6,0
POKE SI+4,65
POKE SI+3,0
POKE SI+3,1
POKE SI+3,2
```

and so on up to 15 to hear the difference (POKE SI+4,64 to stop the sound).

More Detail

Frequency

Frequency is usually measured in repetitions per second, or hertz, after the German physicist of the 19th century. Unfortunately, the 64's SID does not use hertz for the frequency. Suppose that we want to make a sound at 440 hertz (which is a 4th octave A and is used for concert tuning). Then we must do some POKEing to SI and SI+1. Here is how it is done. From 440 hertz we multiply by 16.402273 to get 7217.000120, or 7217, rounded off. This is the frequency number that the 64 understands (but we don't have a name for it—well, we do, but it's not nice). We then take 7217 and split it into HIGH FREQUENCY for SI+1 and LOW FREQUENCY for SI. Given a frequency number for the 64 (not hertz), we divide by 256 and drop the decimals to get the number for SI+1, or HIGH FREQUENCY. In our example, that is INT(7217/256), or 28. For the LOW FREQUENCY, we take what is left over, that is, 7217 − (28 × 256), or 49.

If you have a frequency number for the 64 and wish to know hertz, then multiply the 64's number by .0609672. The program NOTE FREQUENCIES shows all the translations between the 64's frequencies, hertz, and POKE values. Thus, the highest frequency you can create is 3995 Hz, which equals (255 × 256 + 255) × .0609672.

Scales

Most of us are used to hearing a standard scale composed of eight notes—do, re, mi, and so on. On a musical instrument, we learn to play such scales. On a computer, we must create the appropriate frequencies. If you look on a piano, you know that there are 12 notes per octave. Now an octave on a computer is easy: Just double the frequency to go up an octave and halve it to go down an octave. To get the notes in the middle is more tedious, but the same principle applies: multiply. We always go from one note to another by multiplying frequencies by some number. For our piano-type scales, we multiply by some power of the 12th root of 2. To go an octave, we multiply by the 12th power of the 12th root of 2, which is 2. As an example, start at 440 hertz, the fourth octave A. To make a B, which is two notes higher, we must have 440 × ((2↑(1/12)))↑2 = 493.883302 hertz. Now, to get to C, which is one note above B, we can get the hertz by 493.883302 times (2↑(1/12)),or, since C is three notes above A, we can multiply 440 hertz by (2↑(1/12))↑3—it comes out the same. That is the beauty of the "12th root of 2" system. See the programs SCALE and SCALE 2 for more information.

The Sound of Three Voices

The SID chip has three voices, and each voice is controlled by 7 bytes of information. Variables SI and SI+1 control the frequency, or pitch, of voice 1; SI+7 and SI+8 control the pitch of voice 2; and SI+14 and SI+15 control the pitch of voice 3. Recall that SI+2 and SI+3 control the pulse waveform for voice 1, so that SI+9 and SI+10 control the pulse waveform for voice 2, and SI+16 and SI+17 control the pulse waveform for voice 3. The waveforms and start/stop for voice 1 are POKEd in SI+4, so for voice 2, they are POKEd in SI+11, and for voice 3, in SI+18. Attack/decay was set for voice 1 in SI+5, so SI+12 is used for voice 2, and SI+19 for voice 3. The final setting is sustain/release: for voice 1 in SI+6, for voice 2 in SI+13, and for voice 3 in SI+20. The following table may help.

SI + Memory Locations for SID Control

Control Function	*Voices* 1	2	3
Low frequency	0	7	14
High frequency	1	8	15
Pulse—fine control	2	9	16
Pulse—coarse control	3	10	17
Start/stop and wave type ring modulation synchronization	4	11	18
Attack/decay	5	12	19
Sustain/release	6	13	20

Non-Voice Specific Locations

The four SID control locations SI+21 to SI+24 are not voice specific. Volume is controlled in SI+24 and varies from 0 to 15. Note that POKEing at SI+24 produces an annoying click, so we try to set the volume and leave it alone.

The last major control over sound that SID offers is filtering. Just as the name implies, filtering a sound removes part of the sound. There are three types of filters: (1) low-pass, which allows low frequency sound to pass through (filtering out higher frequencies); (2) high-pass, which allows high frequencies to pass through; and (3) band-pass, which allows pitches in the middle to pass through. The frequency for the filter is specified at SI+21 and SI+22, in the same way that frequency was set with POKEs to SI and SI+1. The sharpness or resonance of the filter ranges from 0 to 15 and, after being multiplied by 16, is POKEd at SI+23. Also, one tells the SID which voices to filter at SI+23. To turn on the filter for voice 1, you must add 1 to the filter resonance desired, for voice 2 add 2, and for voice 3 add 4, and then POKE the result to SI+23. For example, if we wanted to filter all three voices and have a filter resonance of 11, then we would type:

```
POKE SI+23, (16*11) + 1 + 2 + 3
```

To select the type of filter, we have to add to the volume. Add 64 for a high-pass, 32 for a band-pass, and 16 for a low-pass. Hence, for maximum volume and a band-pass filter, we would type:

```
POKE SI+24, 15 + 32.
```

The program voice maker will allow you to set all of these controls easily and hear the results. Happy sounds!

Special note on the SID chip: In Chapter 7, "Video Arcade," we often PEEKed at the VIC control locations to see what the values were. For the SID, this does not work. PEEKing at the SID locations always finds a zero.

BASIC Commands Used in This Chapter

ABS	FOR.NEXT
ASC	GET
CHR$	GOSUB
DATA	GOTO
DIM	IF.THEN
DEF FN	INT
END	LEN

MID$	REM
ON I GOSUB	RETURN
POKE	RND
PRINT	SPC
READ	STR$

Programming Techniques Used in This Chapter

1. *Wait loops—there are two types:*

   • Wait a specified amount of time with a FOR–NEXT loop.

   See lines 210,220 in BUZZER; 400 in BELL.

   • Wait for some key to be pressed and then do something.

   Most of the programs keep running until some (any) key is pressed and then they end. See lines 280,290 in WILD SOUND; 280,290 in COMPUTER TALK.

2. *Keyboard parsing.* CHORD ORGAN and VOICE MAKER do different things depending upon which key is pressed. The technique is called "keyboard parsing." A very simple form can be found in NOTE FREQUENCIES in lines 240–290, where something different happens if H,P, or S is pressed.

   In VOICE MAKER a string of valid keypresses is created in Q$. It is just a string of the keyboard. Then the actual key pressed is compared, character by character, to this "parsing string." When a match is found, then you will know the number of the character in the string (for example, the fifth character) and do a ON GOSUB to act on it. See lines 1030 to 1090 to set up the keyboard string, line 4010, to parse it.

   In CHORD ORGAN, lines 200 to 240, more complex form of parsing is used. The ASCII (number) value of the key pressed is determined, and from this number we decide to do one of two things: (1) use the number to make a frequency, or (2) make up a new chord. Instead of using the ASCII value directly (which can be rather mixed up), we define an array A%, which has 256 entries—mainly 0s, because we do not use most ASCII values. Now the ASCII value for the function key F1 is 133, and we let A%(133) be 1. In this way we can sequence the keys however we want. We make the "1" key a 12 (note the importance of the number 12 for scales), the "2" key a 13, the "Q" key a 22, and so on. If the key pressed is assigned a number from 1 to 8, then it is a function key, and we have some GOSUBs to take care of those commands. If the number assigned is greater than 8, then we make up a new note and play a chord based on it.

3. *Bounds.* To make a program user-friendly, or, sometimes, to keep it from quitting due to an error, you must make sure that a variable does not get too large or small. For example, you cannot POKE a value greater than 255 into a memory location. See lines 250,260 in WILD SOUND, 310,320 in AMBULANCE, and 320 in SOUND EFFECTS.
4. *Randomness.* In WILD SOUND, COMPUTER TALK, SOUND EFFECTS, and NOISY LOOP, a random number frequency is used to generate the sound. The RND function in BASIC gives a "random" number from 0 to 1. Lines 350,360 in MUSIC BOX generate random numbers to be added to frequency in the range −10 to 10 (basically, 20*RND−10 does that). Also see line 240 in WILD SOUND and line 40 in SOUND EFFECTS.
5. *Define function.* This adds readability to your program and makes for fewer keystrokes to enter a program. For example, we might want to find the remainder after dividing by 256 for a number of different variables. For the variable X, we would have X − INT(X/256) × 256. Instead of coding this for each different variable, we can define a function FNI(Z). Then, whenever we want the result, we can simply use FNI(X), FNI(Y), and so on. See lines 40, 190 in SOUND EFFECTS, where the effect is to make the program a bit more readable.
6. *Centering text.* See line 180 in MIDNIGHT or lines 5000–5080 in NOTE FREQUENCIES.
7. *Speeding up a program.* Do this by placing subroutines at the beginning of the program. This makes a more confusing program (the top is not the beginning) but can speed it up. See lines 100–190 in VOICE MAKER, which make a subroutine called from line 4220.

Buzzer

Someone's at the door.

```
5 REM          BUZZER
100 WV = 33           :     REM  SAWTOOTH
110 SI = 54272        :     REM  SID REGISTER
120 GOSUB 1000        :     REM  RESET SID
130 POKE SI+24,15     :     REM  VOLUME
140 POKE SI+1,5       :     REM  HIGH FREQ.
150 POKE SI+5,0       :     REM  ATTACK/DECAY
160 POKE SI+6,240     :     REM  SUSTAIN/REL
170 POKE SI+4,WV      :     REM  WAVEFORM
180 REM          WAIT
190 FOR I = 1 TO 350
200 NEXT
600 GOSUB 1000        :     REM  RESET SID
```

```
610 END
980 :
990 REM         RESETS SID REGISTERS
1000 FOR I = 0 TO 24
1010 POKE SI+I,0
1020 NEXT
1090 RETURN
```

Bell

As in ding–dong.

```
5 REM          BELL
6 REM  THIS IS BUZZER  WITH
7 REM  LINES 100 AND 140-160
8 REM  CHANGED  AND LINES 210-240 ADDED
30 :
100 WV = 17          :     REM  SAWTOOTH
110 SI = 54272       :     REM  SID REGISTER
120 GOSUB 1000       :     REM  RESET SID
130 POKE SI+24,15    :     REM  VOLUME
140 POKE SI+1,100    :     REM  HIGH FREQ.
150 POKE SI+5,15     :     REM  ATTACK/DECAY
160 POKE SI+6,252    :     REM  SUSTAIN/REL
170 POKE SI+4,WV     :     REM  WAVEFORM
180 REM         WAIT
190 FOR I = 1 TO 350
200 NEXT
210 POKE SI+4,WV-1 :       REM  TURN OFF
220 REM         WAIT
230 FOR I = 1 TO 3000
240 NEXT
600 GOSUB 1000       :     REM  RESET SID
610 END
980 :
990 REM         RESETS SID REGISTERS
1000 FOR I = 0 TO 24
1010 POKE SI+I,0
1020 NEXT
1090 RETURN
```

Wild Sound

Notes, notes everywhere.

```
5 REM           WILD SOUND
6 REM  THIS IS BEEP WITH LINES 230-290 ADDED
10 :
20 REM HIT ANY KEY TO STOP
30 :
```

```
100 WV = 17          :     REM  SAWTOOTH
110 SI = 54272       :     REM  SID REGISTER
120 GOSUB 1000       :     REM  RESET SID
130 POKE SI+24,15    :     REM  VOLUME
140 POKE SI+1,25     :     REM  HIGH FREQ.
150 POKE SI+5,16     :     REM  ATTACK/DECAY
160 POKE SI+6,240    :     REM  SUSTAIN/REL
170 POKE SI+4,WV     :     REM  WAVEFORM
180 REM         WAIT
190 FOR I = 1 TO 75
200 NEXT
230 SP = 20
240 FR = FR + SP*RND(0) - SP/2
250 IF FR>255 THEN FR=256-ABS(FR-255)
260 IF FR<0 THEN FR=ABS(FR)
270 POKE SI+1,FR
280 GET A$:IF A$<>"" THEN 600
290 GOTO 240
600 GOSUB 1000       :     REM  RESET SID
610 END
980 :
990 REM         RESETS SID REGISTERS
1000 FOR I = 0 TO 24
1010 POKE SI+I,0
1020 NEXT
1090 RETURN
```

Beep

It does what it says.

```
5 REM          BEEP
6 REM  THIS IS BUZZER  WITH
7 REM  LINES 100, 140-150 AND 190 CHANGED
30 :
100 WV = 17          :     REM  SAWTOOTH
110 SI = 54272       :     REM  SID REGISTER
120 GOSUB 1000       :     REM  RESET SID
130 POKE SI+24,15    :     REM  VOLUME
140 POKE SI+1,25     :     REM  HIGH FREQ.
150 POKE SI+5,16     :     REM  ATTACK/DECAY
160 POKE SI+6,240    :     REM  SUSTAIN/REL
170 POKE SI+4,WV     :     REM  WAVEFORM
180 REM         WAIT
190 FOR I = 1 TO 75
200 NEXT
600 GOSUB 1000       :     REM  RESET SID
610 END
980 :
990 REM         RESETS SID REGISTERS
1000 FOR I = 0 TO 24
```

```
1010 POKE SI+I,0
1020 NEXT
1090 RETURN
```

Computer Talk

This is the way computers talk in movies.

```
5 REM          COMPUTER TALK
6 REM  THIS IS BEEP  WITH LINES 230-610 ADDED
30 :
100 WV = 17          :     REM  SAWTOOTH
110 SI = 54272       :     REM  SID REGISTER
120 GOSUB 1000       :     REM  RESET SID
130 POKE SI+24,15    :     REM  VOLUME
140 POKE SI+1,25     :     REM  HIGH FREQ.
150 POKE SI+5,16     :     REM  ATTACK/DECAY
160 POKE SI+6,240    :     REM  SUSTAIN/REL
170 POKE SI+4,WV     :     REM  WAVEFORM
180 REM         WAIT
190 FOR I = 1 TO 75
200 NEXT
230 POKE SI+1,256*RND(0)
280 GET A$:IF A$<>"" THEN 600
290 GOTO 230
600 GOSUB 1000       :     REM  RESET SID
610 END
980 :
990 REM         RESETS SID REGISTERS
1000 FOR I = 0 TO 24
1010 POKE SI+I,0
1020 NEXT
1090 RETURN
```

Ambulance

It gets closer and then goes away. Try some other values for RA and SP in lines 190 and 200.

```
5 REM          AMBULANCE
20  REM  HIT A KEY TO STOP
30 :
100 SI = 54272       :     REM  SID REGISTER
110 GOSUB 1000       :     REM  RESET SID
150 :
160 POKE SI+6,240    :     REM  SUSTAIN/REL
170 POKE SI+4,33     :     REM  SAWTOOTH
180 L1 = 100         :     REM  LOW FREQ.
190 RA = 1    :            REM  SIREN CHANGE RATE
200 SP = 5    :            REM  VOLUME CHANGE RATE
```

```
210 V1=2
220 POKE SI+24,V1    :     REM  VOLUME
230 FOR I=255 TO L1 STEP -RA
240 POKE SI+1,I      :     REM  FREQUENCY
250 NEXT
260 FOR I= L1 TO 255 STEP RA
270 POKE SI+1,I      :     REM  FREQUENCY
280 NEXT
290 GET A$:IF A$<>"" THEN 600
300 REM         REVERSE VOLUME CHANGE IF
310 REM         TOO BIG OR TOO SMALL
320 IF V1+SP>15 OR V1+SP<0 THEN SP=-SP
330 V1=V1+SP
340 GOTO 220         :     REM  DO IT AGAIN
600 GOSUB 1000       :     REM  RESET SID
610 END
980 :
990 REM         RESETS SID REGISTERS
1000 FOR I = 0 TO 24
1010 POKE SI+I,0
1020 NEXT
1090 RETURN
```

Sound Effects

As is, this program will produce random sound effects, starting with a low volume that gradually increases. If you want some specific effects, you can set the variable RA in line 190 and SP in line 200 to the values below. For example, for a RAY GUN, change lines 190 and 200 to read:

```
190 RA= 30
200 SP=-1
```

SOUND EFFECTS

Name	*RA*	*SP*
UFO approaching	50	0.2
Ray gun	30	−1
Siren	2 or 6	0
Siren leaving	6	−1

```
5 REM          SOUND EFFECTS
6 REM  THIS IS AMBULANCE WITH LINE
7 REM  40 ADDED  AND LINES 190-200 AND
8 REM  320 CHANGED
20 REM  HIT A KEY TO STOP
30 :
40 DEF FND(X) = 2 * (X*RND(0)+1) - X
100 SI = 54272       :     REM  SID REGISTER
```

```
110 GOSUB 1000      :     REM  RESET SID
150 :
160 POKE SI+6,240   :     REM  SUSTAIN/REL
170 POKE SI+4,33    :     REM  SAWTOOTH
180 L1 = 100        :     REM  LOW FREQ.
190 RA=FND(50):           REM  SIREN CHANGE RATE
200 SP = 6*RND(0) - 3
210 V1=2
220 POKE SI+24,V1   :     REM  VOLUME
230 FOR I=255 TO L1 STEP -RA
240 POKE SI+1,I     :     REM  FREQUENCY
250 NEXT
260 FOR I= L1 TO 255 STEP RA
270 POKE SI+1,I     :     REM  FREQUENCY
280 NEXT
290 GET A$:IF A$<>"" THEN 600
300 REM         REVERSE VOLUME CHANGE IF
310 REM         TOO BIG OR TOO SMALL
320 IF V1+SP>15 OR V1+SP<0 THEN 190
330 V1=V1+SP
340 GOTO 220        :     REM  DO IT AGAIN
600 GOSUB 1000      :     REM  RESET SID
610 END
980 :
990 REM         RESETS SID REGISTERS
1000 FOR I = 0 TO 24
1010 POKE SI+I,0
1020 NEXT
1090 RETURN
```

Scale

This program plays half notes all the way up.

```
5 REM          SCALE
6 REM  THIS IS BELL    WITH
7 REM  LINES 180 TO 250 MODIFIED
30 :
100 WV = 17         :     REM  SAWTOOTH
110 SI = 54272      :     REM  SID REGISTER
120 GOSUB 1000      :     REM  RESET SID
130 POKE SI+24,15   :     REM  VOLUME
150 POKE SI+5,15    :     REM  ATTACK/DECAY
160 POKE SI+6,252   :     REM  SUSTAIN/REL
170 POKE SI+4,WV    :     REM  WAVEFORM
180 REM         WAIT
190 FOR I = 0 TO 74
200 N=2703*(2↑((I-20)/12))
205 PRINT INT(N),I
210 POKE SI,N-INT(N/256)*256
220 POKE SI+1,INT(N/256)
```

```
230 FOR J = 1 TO 100
240 NEXT
250 NEXT
600 GOSUB 1000       :     REM  RESET SID
610 END
980 :
990 REM         RESETS SID REGISTERS
1000 FOR I = 0 TO 24
1010 POKE SI+I,0
1020 NEXT
1090 RETURN
```

Scale 2

This program plays eight-note do–re–mi scales. You can modify the values for the array A in lines 2000–2020 to make scales other than major scales. For example, in line 2010, if A(2)=3, then a minor scale will be played. Note that if you change BE in line 180, any higher value will make the program end with an error in line 220 when N is greater than 256 × 256.

```
5 REM          SCALE2
6 REM  THIS IS SCALE   WITH
7 REM  LINES 175 TO 260 MODIFIED
8 REM  AND LINES 2000-2030 ADDED
30 :
100 WV = 17          :     REM  SAWTOOTH
110 SI = 54272       :     REM  SID REGISTER
120 GOSUB 1000       :     REM  RESET SID
130 POKE SI+24,15    :     REM  VOLUME
150 POKE SI+5,15     :     REM  ATTACK/DECAY
160 POKE SI+6,252    :     REM  SUSTAIN/REL
170 POKE SI+4,WV     :     REM  WAVEFORM
175 GOSUB 2000
180 BE=255
183 FOR K=2 TO 7
190 FOR I = 0 TO 7
200 N=BE*(2↑K*(2↑((A(I)    )/12)))
205 PRINT INT(N),A(I)
210 POKE SI,N-INT(N/256)*256
220 POKE SI+1,INT(N/256)
230 FOR J = 1 TO 100
240 NEXT J
250 NEXT I
260 NEXT K
600 GOSUB 1000       :     REM  RESET SID
610 END
980 :
990 REM         RESETS SID REGISTERS
1000 FOR I = 0 TO 24
1010 POKE SI+I,0
```

```
1020 NEXT
1090 RETURN
2000 DIM A(8):A(0)=0
2010 A(1)=2:A(2)=4:A(3)=5:A(4)=7
2020 A(5)=9:A(6)=11:A(7)=12:A(8)=14
2030 RETURN
```

Music Box

This program provides many bells in random three-part harmony. For very different music boxes, change the value in line 520 from 500 to 100, 50, or 200 and change the value in line 800 from 40 to 1, 10, or 100.

```
5 REM          MUSIC BOX
20 REM  HIT ANY KEY TO STOP
40 :
100 SI = 54272       :     REM  SID REGISTER
110 GOSUB 1000       :     REM  RESET SID
140 POKE SI+24,15    :     REM  VOLUME
150 POKE SI+5,0      :     REM  ATTACK/DECAY
160 POKE SI+12,0
170 POKE SI+19,0
180 POKE SI+6,252    :     REM  SUSTAIN/REL
190 POKE SI+13,252
200 POKE SI+20,252
230 SP = 20
350 FR = FR + SP*RND(0) - SP/2
360 IF FR>220 THEN FR=221-ABS(FR-220)
370 IF FR<30 THEN FR=60-FR
380 POKE SI+1,FR
390 POKE SI+4,17      :     REM START
400 GOSUB 800
410 POKE SI+4,16      :     REM RELEASE START
420 GOSUB 800
430 POKE SI+8,FR*(2↑(4/12))
440 POKE SI+11,17     :     REM VOICE 2
450 GOSUB 800
460 POKE SI+11,16
470 GOSUB 800
480 POKE SI+15,FR*(2↑(7/12))
490 POKE SI+18,17     :     REM VOICE 3
500 GOSUB 800
510 POKE SI+18,16
520 FOR I=1 TO 500:NEXT
530 GET A$:IF A$<>"" THEN 600
540 GOTO 350
600 GOSUB 1000       :     REM  RESET SID
610 END
690 :
790 REM         DELAY SUBROUTINE
```

```
800 FOR I = 1 TO 40
810 NEXT
820 RETURN
990 REM         RESETS SID REGISTERS
1000 FOR I = 0 TO 24
1010 POKE SI+I,0
1020 NEXT
1090 RETURN
```

Marching Feet

Tramp . . . tramp . . . tramp.

```
5 REM          MARCHING FEET
6 REM  THIS IS BEEP     WITH
7 REM  LINES 230-260 ADDED
8 REM  AND LINE 100 CHANGED
30 :
100 WV = 129          :     REM  SAWTOOTH
110 SI = 54272        :     REM  SID REGISTER
120 GOSUB 1000        :     REM  RESET SID
130 POKE SI+24,15     :     REM  VOLUME
140 POKE SI+1,25      :     REM  HIGH FREQ.
150 POKE SI+5,16      :     REM  ATTACK/DECAY
160 POKE SI+6,240     :     REM  SUSTAIN/REL
170 POKE SI+4,WV      :     REM  WAVEFORM
180 REM         WAIT
190 FOR I = 1 TO 75
200 NEXT
230 POKE SI+4,WV-1    :     REM TURN OFF
240 FOR I=1 TO 350:NEXT: REM WAIT A BIT
250 GET A$:IF A$<>"" THEN 600
260 GOTO 170
600 GOSUB 1000        :     REM  RESET SID
610 END
980 :
990 REM         RESETS SID REGISTERS
1000 FOR I = 0 TO 24
1010 POKE SI+I,0
1020 NEXT
1090 RETURN
```

Midnight

The clock strikes midnight at the graveyard.

```
5 REM          MIDNIGHT
6 REM  THIS IS BELL  WITH
7 REM  LINES 140 AND 170-240
8 REM  CHANGED  AND LINE 250  ADDED
```

```
30 :
100 WV = 17          :     REM  SAWTOOTH
110 SI = 54272       :     REM  SID REGISTER
120 GOSUB 1000       :     REM  RESET SID
130 POKE SI+24,15    :     REM  VOLUME
140 POKE SI+1,8      :     REM  HIGH FREQ.
150 POKE SI+5,15     :     REM  ATTACK/DECAY
160 POKE SI+6,252    :     REM  SUSTAIN/REL
170 FOR J=1 TO 12
180 PRINT CHR$(147):PRINT SPC((40-LEN(STR$(J)))/2);J
190 POKE SI+4,WV     :    REM WAVE FORM
200 FOR I=1 TO 100   :    REM WAIT A BIT
210 NEXT
220 POKE SI+4,WV-1   :    REM TURN OFF
230 FOR I=1 TO 2000
240 NEXT
250 NEXT
600 GOSUB 1000       :     REM  RESET SID
610 END
980 :
990 REM         RESETS SID REGISTERS
1000 FOR I = 0 TO 24
1010 POKE SI+I,0
1020 NEXT
1090 RETURN
```

Busy Signal

This is just like calling "Information."

```
5 REM          BUSY SIGNAL
30 :
100 SI = 54272          : REM  SID REGISTER
110 GOSUB 1000          : REM  RESET SID
120 POKE SI+24,15       : REM  VOLUME
130 POKE SI+1,38        : REM  FREQUENCY
140 POKE SI+3,3         : REM  PULSE RATE
150 POKE SI+22,90       : REM  FILTER
160 POKE SI+23,1        : REM  FILTER
170 POKE SI+24,63       : REM  FILTER
180 POKE SI+5,16        : REM ATTACK/DECAY
190 POKE SI+6,240       : REM SUSTAIN/REL
200 POKE SI+4,65        : REM WAVE FORM
210 :
220 FOR I=1 TO 350:NEXT: REM WAIT A BIT
230 POKE SI+4,16        : REM TURN OFF
240 FOR I=1 TO 350:NEXT: REM WAIT A BIT
250 :
260 GET A$:IF A$<>"" THEN 600
270 GOTO 200
600 GOSUB 1000
```

```
610 END
990 REM         RESETS SID REGISTERS
1000 FOR I = 0 TO 24
1010 POKE SI+I,0
1020 NEXT
1090 RETURN
```

Cop at Corner

This is the other kind of siren.

```
5 REM      COP AT CORNER
20 REM  HIT A KEY TO STOP
30 :
100 SI = 54272        :     REM  SID REGISTER
110 GOSUB 1000        :     REM  RESET SID
120 POKE SI+6,240     :     REM  SUSTAIN/REL
130 POKE SI+4,33      :     REM  SAWTOOTH
180 POKE SI+24,12     :     REM VOLUME
190 POKE SI+1,45
200 FOR I = 1 TO 100:NEXT
210 POKE SI+1,20
220 FOR I = 1 TO 100:NEXT
230 GET A$:IF A$<>"" THEN 600
240 GOTO 180          :     REM  DO IT AGAIN
600 GOSUB 1000
610 END
990 REM         RESETS SID REGISTERS
1000 FOR I = 0 TO 24
1010 POKE SI+I,0
1020 NEXT
1090 RETURN
```

Wolf Whistle

This is the only whistle in this chapter.

```
5  REM      WOLF WHISTLE
30 :
100 SI = 54272        :     REM  SID REGISTER
110 GOSUB 1000        :     REM  RESET SID
120 POKE SI+6,241     :     REM  SUSTAIN/REL
130 POKE SI+4,17      :     REM  SAWTOOTH
140 RA = 1.5          :     REM  SIREN CHANGE RATE
150 POKE SI+24,15     :     REM VOLUME
160 :
170 FOR I = 100 TO 255 STEP RA+2
180 POKE SI+1,I
190 NEXT
200 POKE SI+4,16
```

```
210 :
220 FOR I = 1 TO 40
230 NEXT              :     REM  DELAY LOOP
240 :
250 POKE SI+4,17
260 FOR I = 100 TO 255 STEP RA
270 POKE SI+1,I
280 NEXT
290 FOR I=255 TO 100 STEP -RA
300 POKE SI+1,I
310 NEXT
320 :
330 GET A$:IF A$<>"" THEN 600
340 POKE SI+4,16
350 FOR I=1 TO 500
360 NEXT              :     REM  DELAY LOOP
370 POKE SI+4,17
380 GOTO 150
600 GOSUB 1000    :         REM  RESET SID
610 END
990 REM         RESETS SID REGISTERS
1000 FOR I = 0 TO 24
1010 POKE SI+I,0
1020 NEXT
1090 RETURN
```

Noisy Loop

This program shows what happens when you POKE a variety of waveforms.

```
5 REM           NOISY LOOP
6 REM  THIS IS BEEP  WITH
7 REM  LINES 170-200   CHANGED
8 REM  AND LINES 210-220 ADDED
9 :
10 REM  YOU MAY NEED TO TURN UP THE VOLUME
30 :
100 WV = 17           :     REM  SAWTOOTH
110 SI = 54272        :     REM  SID REGISTER
120 GOSUB 1000        :     REM  RESET SID
130 POKE SI+24,15     :     REM  VOLUME
140 POKE SI+1,25      :     REM  HIGH FREQ.
150 POKE SI+5,16      :     REM  ATTACK/DECAY
160 POKE SI+6,240     :     REM  SUSTAIN/REL
170 FOR I = 0 TO 138
180 POKE SI+ 4,I
190 PRINT I
200 FOR J = 1 TO 100
210 NEXT J,I
220 POKE SI+4,16
600 GOSUB 1000        :     REM  RESET SID
```

```
610 END
980 :
990 REM          RESETS SID REGISTERS
1000 FOR I = 0 TO 24
1010 POKE SI+I,0
1020 NEXT
1090 RETURN
```

Chord Organ

This program plays three-tone chords—eight different chords played from the keyboard. To sound a chord, hit any of the regular keys on the keyboard—the letters, numbers, or punctuation. Each key plays a different pitch. To change the type of chord being played (major, minor, 7th, and so on), hit any function key, with or without a shift. The top line of the display will show the type of chord and will change when a function key is hit. The rest of the display shows what keys have been hit by printing them.

There are several challenges related to this program. With a bit of programming effort, you can greatly enhance this program by storing music, instrument voices, and so forth. Play with it first, and then try your hand at some modifications.

```
5 REM          CHORD ORGAN
30 :
100 CL$ = CHR$(147)    :  REM CLEAR SCREEN
110 GOSUB 9000
200 GET A$:IF A$="" THEN 200
210 PRINTA$;SPC(1);
220 I = A%(ASC(A$))
230 IF I>8 THEN 400
240 ON I GOSUB 1000,1010,1020,1030,1040,1050,1060,1070 : GOTO 200
400 POKE SI+1,NH(I) :     POKE SI,NL(I)
410 POKE SI+8,NH(I+A1) : POKE SI+7,NL(I+A1)
420 POKE SI+15,NH(I+A2): POKE SI+14,NL(I+A2)
430 POKE SI+4,33: POKE SI+11,33: POKE SI+19,33
440 FOR J = 1 TO 50:      NEXT
450 POKE SI+4,32: POKE SI+11,32: POKE SI+18,32
460 GOTO 200
990 :
995 REM          THE FOLLOWING CHANGE CHORD TYPE
1000 A1=4: A2=7:     PRINT CL$+"1/3/5 CHORD":RETURN
1010 A1=4: A2=-5:    PRINT CL$+"1/3/5-LOWER":RETURN
1020 A1=3: A2=-5:    PRINT CL$+"1/MINOR/5-LOWER":RETURN
1030 A1=7: A2=10:    PRINT CL$+"1/5/7TH":RETURN
1040 A2=4: A1=12:    PRINT CL$+"1/3/OCTAVE":RETURN
1050 A2=4: A1=-12:   PRINT CL$+"1/3/OCTAVE LOW":RETURN
1060 A1=3: A2=-2:    PRINT CL$+"MINOR 7TH":RETURN
1070 A1=5: A2=9:     PRINT CL$+"-5/1/3":RETURN
8900 :
```

```
8990 REM        SETUP ROUTINE
9000 PRINT CL$+"SETTING UP"
9010 SI = 54272       :    REM  SID REGISTER
9020 FOR I = 0 TO 24  :    REM  CLEAR SID
9030 POKE SI+I,0
9040 NEXT
9050 POKE SI+24,15    :    REM  VOLUME
9060 POKE SI+5,4      :    REM ATTACK/DECAY
9070 POKE SI+12,4
9080 POKE SI+19,4
9090 POKE SI+6,241     :   REM  SUSTAIN/REL
9100 POKE SI+13,241
9110 POKE SI+20,241
9120 REM        COMMAND STRING
9130 Q$ = ""
9140 FOR I = 133 TO 140
9150 Q$ = Q$ + CHR$(I)
9160 NEXT
9170 Q$ = Q$ + "1234567890+-\QWERTYUIOP@*↑ASDFGHJKL:;=ZXCVBNM,./"
9180 LQ = LEN(Q$)
9190 DIM NH(80),NL(80)
9200 FOR I = 0 TO 80:  N=2703*(2↑((I-20)/12))
9210 IF N>65535 THEN N=65535
9220 NH(I) = INT(N/256)
9230 NL(I) = N - NH(I)*256
9240 NEXT
9250 A1=4
9260 A2=7
9270 DIM A%(256)
9280 FOR I = 1 TO LQ
9290 A = ASC(MID$(Q$,I,1))
9300 A%(A) = I
9310 IF I>8 THEN A%(A)=I+4
9320 NEXT
9340 PRINT CL$+"READY MAJOR CHORD 1/3/5"
9350 RETURN
```

Voice Maker

This program makes every sound that the 64 can make.

ALPHABETICAL LIST OF COMMANDS

Key	*Function*	*Shifted Function*
A	Increase ATTACK	Decrease ATTACK
B	Input ATTACK duration	
C	Input RELEASE duration	
D	Increase DECAY	Decrease DECAY
E	Enable/disable voice	
F	Filter frequency	Resonance

ALPHABETICAL LIST OF COMMANDS

Key	*Function*	*Shifted Function*
G	Filter type	
H	Enable/disable filter	
I	Change sign of INCREMENT	Input INCREMENT
K	Enable/disable screen update	
M	Ring MODULATE voice	
N	Increase NOTE frequency	
O	Set voice 3 output off	
P	PLAY note(s)	Continuous PLAY (autoplay must be off)
Q	Enable/disable automatic play	
R	Increase RELEASE	Decrease RELEASE
S	Increase SUSTAIN	Decrease SUSTAIN
T	Type of WAVEform	
V	Next VOICE	Increase VOLUME
W	Change pulse wave WIDTH	
Y	Synchronize voice	

The top row of keys (1 to -) acts as a one-octave keyboard.

A User's Guide to VOICE MAKER

When you first get VOICE MAKER running, it will take a few seconds to set itself up and then should display a standard screen of information, as shown below. The first line shows the current voice that is being controlled by voice-specific commands, such as frequency, wave type, ADSR, and so on.

VOICE MAKER Initial Screen

```
VOICE 1
INCRM 10  AD 100  RD 100 VOLUME 15
FREQUENCY 2145  2145  2145
WAVE TYPE 1  1  1
PULSE WIDTH 2048  2048  2048
ATTACK DECAY SUSTAIN RELEASE
 7      7      7       7
 7      7      7       7
 7      7      7       7
DISABLE    RING MOD  SYNCH
 0          0         0
 1          0         0
```

```
 1          Ø          Ø
FILTER FRQ 2145 FILTER RES 7
FILTER TYPE Ø        V3= Ø
FILTER ENABLE Ø  Ø  Ø
```

To use VOICE MAKER, first play the one-octave keyboard to hear the initial sound. Do this by hitting the top row of keys (1 to 0, as well as the + and −) or simply press the "P" key. You might notice that the keyboard seems a little slow in playing. That is to be expected because each note is resetting all of the controls in the SID chip, and there is quite a lot of calculation to do. Now change the envelope by typing shift–A seven times, which sets the attack duration to 0. Do the same to the delay by typing shift–D seven times. You may want to disable the display update by pressing the K key and then enable it after typing the shift–A seven times. This will speed things up. Also, disabling the autoplay (the Q key) will speed things up, but then the note will be sounded only if you hit a "note-playing key" (top row and P). Now raise the sustain level to 9 by hitting S two times, until the display shows the sustain level for the first voice to be 9 (if the display is not changing, press the K key). Similarly, raise the release time to 10 by hitting R three times. Try playing some notes again. If you have disabled the screen update and/or autoplay, hit the P key to be sure that the new settings are what you hear. The sound should now be like that of a guitar (we said, "like" a guitar). Now try changing the waveform type by pressing T. Note how the display changes on the fourth line, starting with WAVE TYPE. At a wave type of 2, or sawtooth, the notes sound like a piano. At 3, they are similar to a harpsichord. At 4, they sound like rifle fire. The same frequency and ADSR envelope can sound very different depending on the waveform type chosen.

We'll now change the frequency, first trying very low frequencies. The beginning INCREMENT factor is only 10 (see line 2). Let's increase this to 1000 by typing:

```
shift-I
1000
```

(that is, typing 1000 to the prompt "increment size"). Line 2 should now display 1000 as the INCRM. Change this to −1000 by hitting I. We're now ready to lower the frequency. Do this by hitting N once. This should change the frequency of voice 1 to 1145. Line 3 of the display should now be:

```
FREQUENCY 1145 2145 2145
```

Now get to wave type 2 by pressing T until you see the fourth line of the display show:

```
WAVE TYPE 2 1 1
```

and get continuous play by pressing shift–P (make sure automatic play, the Q key, is off). Then press N and you should hear a "putt-putt" sound—pretty strange, no? The sounds are of very low frequency in hertz. Now let's raise the frequency by hitting I and then N four times. This should make the frequency 4145.

Now let's try our hand at a second voice. Switch to voice 2 by hitting V. Set the ADSR for voice 2 to 0,0,15,12 by following the instructions used above (the A, D, S, and R keys). OK, do you hear a bell when you press P? If so, you've made a mistake, because voice 2 is still disabled (look down near the bottom left of your display). Let's enable it by typing E. This should change the 1 under DISABLE (two lines under it) to a 0, and you should hear two separate sounds whenever you play a note. Note how the sounds go up and down together in pitch as you play. Now, let's raise voice 2's frequency by typing shift–I, replying 10000, and then typing N N N . This should leave voice 2's frequency at 32145, and a very nice bell note should be coming from it. Finally, let's enable voice 3 by typing V E .

Some more unusual effects can be obtained by ring modulating voice 1 with voice 3, voice 2 with voice 1, and voice 3 with voice 2. This can be accomplished by typing M for the current voice. Try playing some notes now. Turn off the ring modulation of voice 1 and 2 and turn on modulation for voice 3. Play some notes now. Make sure that the voices are enabled, or ring modulation will not make any difference. Also make sure that all the voices have the same ADSR values.

The filter can be used to further change the sound. Raise the filter frequency to 3145 by typing F. Enable the filter by typing H. Try changing the filter type by typing G and using each type of filter (1 to 4). Do the same after you have enabled the filter for all the voices (type V until you have the right voice and type H). Another interesting sound can be created by reducing the filter frequency to 2145 and setting the filter type to 1. Then try removing the filter on 2.

Further experimentation with VOICE MAKER will yield interesting and bizarre sound effects. Once you have a particular set of sounds that you wish to use in a program, write down the values from the screen display and prepare the POKEs that correspond to these. We have already talked about these POKE locations in the beginning of this chapter. The location and meaning of all of them are in Appendix P of the *Commodore 64 User's Guide.* Take a look at BELL or CHORD ORGAN to see what POKEs are used. These POKEs should make a bit more sense now, and you should be able to modify them using the values that you have found with VOICE MAKER.

Since this is a long program, we have provided a list of the variables and arrays used in it to help you understand what you are typing. We also suggest that you type the program in sections, saving it to the disk *often.* It is better to type a long program like this one during a couple of sessions rather than in one long sitting.

VARIABLES USED

Arrays One Dim for Each Voice

F — FREQUENCY
W — PULSE WIDTH FOR PULSE WAVEFORM 0 to 4095
A — ATTACK 0 to 15
D — DELAY 0 to 15
S — SUSTAIN 0 to 15
R — RELEASE 0 to 15
T — TYPE OF WAVEFORM 1,2,3, or 4
SD — DISABLE VOICE 0 or 1
MR — RING MODULATE VOICE 3 0 or 1
YS — SYNCHRONIZE VOICE 3 0 or 1
FO — FILTER VOICE 0 or 1

Variables

IN — INCREMENT AMOUNT
AD — ATTACK DURATION : USED IN A FOR NEXT LOOP
RD — RELEASE DURATION : USED IN A FOR NEXT LOOP
FF — FILTER FREQUENCY 0 to 65,535
FR — FILTER RESONANCE 0 to 15
FT — FILTER TYPE 1,2,3, or 4
VL — VOLUME 0 to 15
V — VOICE NUMBER 1,2, or 3
V3 — VOICE 3 OUTPUT DISABLE
Q — AUTO PLAY ON/OFF
QQ PRINT ON/OFF
Z1,Z2,Z3,Z4 TEMPORARY VARIABLES

```
1 REM VOICE MAKER
5 GOTO 1000 :              REM SKIP AROUND
20 :
100 REM         KEYBOARD PLAYING
110 Z7 = 8*(2↑T(1))+SD(1)*8+MR(1)*4+YS(1)*2+1
120 Z8 = 8*(2↑T(2))+SD(2)*8+MR(2)*4+YS(2)*2+1
130 Z9 = 8*(2↑T(3))+SD(3)*8+MR(3)*4+YS(3)*2+1
140 POKE SI+4,Z7:POKE SI+11,Z8:POKE SI+18,Z9
150 FOR Z4 = 0 TO AD:NEXT
160 POKE SI+4,Z7-1:POKE SI+11,Z8-1:POKE SI+18,Z9-1
170 FOR Z4 = 0 TO RD:NEXT
180 RETURN
190 REM         END KEYBOARD PLAYING
200 :
1000 DIM F(3),W(3),A(3),D(3),S(3),R(3)
1010 DIM T(3),SD(3),MR(3),YS(3),FO(3)
1020 REM        BUILD COMMAND STRING
```

```
1030 Q$ = ""
1040 FOR I = 1 TO 32
1050 READ A
1060 Q$ = Q$ + CHR$(A)
1070 NEXT I
1080 Q$ = Q$ + "1234567890+-\"
1090 LQ = LEN(Q$)
1500 SI = 54272
1510 MF=64*1024
1520 MW=4096
1600 FOR Z1 = 0 TO 24:POKE SI+Z1,0:NEXT
1700 SD(2)=1:SD(3)=1:     REM DISABLE VOICES 2,3
1710 SD(1) = 0       :    REM START VOICE 1 ONLY
2000 FOR V = 1 TO 3
2010 F(V)=2145       :    REM 3RD OCTIVE C
2020 W(V)=2048       :    REM SQUARE WAVE
2030 T(V)=1          :    REM TRIANGE WAVE FORM
2040 MR(V)=0         :    REM NO RING MODULATE
2050 YS(V)=0         :    REM NO SYNCH
2060 FO(V)=0         :    REM NO FILTER
2070 A(V)=7:D(V)=7:S(V)=7:R(V)=7
2080 GOSUB 20030:GOSUB 20130:GOSUB 20220:GOSUB 20650:GOSUB 20850
2090 NEXT V
2100 FF=2145         :    REM MID C
2110 FR=7
2120 V3=0      :           REM NO DISABLE VOICE 3 OUT
2130 FT=0
2140 VL=15
2150 V=1
2160 IN=10
2170 AD=100:RD=100
2180 Q=1:QQ=0    :        REM AUTOPLAY, DISPLAY ON
2200 GOSUB 21020
2210 GOSUB 21120
2220 GOSUB 21540
2230 GOTO 10000:          REM PRINT THEN BACK HERE
4000 GET A$:IF A$="" THEN 4000
4010 FOR I=1 TO LQ:IF A$<>MID$(Q$,I,1) THEN NEXT:GOTO 4000
4020 X% = I
4030 IF X%>10 THEN X%=X%-10:GOTO 4060
4040 ON X% GOSUB 20000,20100,20200,20300,20400,20500,20600,
     20610,20700,20710
4050 GOTO 10000
4060 IF X%>10 THEN X%=X%-10:GOTO 4090
4070 ON X% GOSUB 20800,20810,20900,20910,21000,21100,21200,
     21300,21400,21500
4080 GOTO 10000
4090 IF X%>10 THEN X%=X%-10:GOTO 4120
4100 ON X% GOSUB 22000,22100,22200,22300,22400,22600,22700,
     23000,24000,24100
4110 GOTO 10000
4120 IF X%>2 THEN X%=X%-2:GOTO 4150
```

```
4130 ON X% GOSUB 25000,25100
4140 GOTO 10000
4150 FOR Z1 = 1 TO 3
4160 Z2 = F(Z1)*(2↑((X%-7)/12))
4180 POKE SI+7*(Z1-1)+4,0
4190 POKE SI+7*(Z1-1),Z2-INT(Z2/256)*256
4200 POKE SI+7*(Z1-1)+1,INT(Z2/256)
4210 NEXT
4220 GOSUB 140:GOTO 4000
9980 :
9990 REM        SHOW PARAMETERS
10000 IF QQ=0 THEN GOTO 10030
10010 IF Q=1 THEN GOSUB 23000
10020 GOTO 4000
10030 PRINT CHR$(147);:PRINT "VOICE";V
10040 PRINT "INCRM";IN;" AD";AD;" RD";RD;"VOLUME";VL
10050 PRINT "FREQUENCY";F(1);F(2);F(3)
10060 PRINT "WAVE TYPE";T(1);T(2);T(3)
10070 PRINT "PULSE WIDTH";W(1);W(2);W(3)
10080 PRINT "ATTACK DECAY SUSTAIN RELEASE"
10090 PRINT A(1);"     ";D(1);"    "S(1);"      ";R(1)
10100 PRINT A(2);"     ";D(2);"    "S(2);"      ";R(2)
10110 PRINT A(3);"     ";D(3);"    "S(3);"      ";R(3)
10120 PRINT "DISABLE    RING MOD  SYNCH"
10130 PRINT SD(1),MR(1),YS(1)
10140 PRINT SD(2),MR(2),YS(2)
10150 PRINT SD(3),MR(3),YS(3)
10160 PRINT "FILTER FRQ";FF;"FILTER RES";FR
10170 PRINT "FILTER TYPE";FT;"       V3=";V3
10180 PRINT "FILTER ENABLE";FO(1);FO(2);FO(3)
10190 IF Q=1 THEN GOSUB 23000
10200 GOTO 4000
19990 :
20000 F(V) = F(V) + IN
20010 IF F(V)>MF THEN F(V)=MF
20020 IF F(V)<0 THEN F(V)=0
20030 POKE SI+7*(V-1),F(V)-INT(F(V)/256)*256
20040 POKE SI+7*(V-1)+1,INT(F(V)/256)
20050 RETURN
20100 W(V) = W(V) + IN
20110 IF W(V)>MW THEN W(V)=MW
20120 IF W(V)<0 THEN W(V)=0
20130 POKE SI+7*(V-1)+2,W(V)-INT(W(V)/256)*256
20140 POKE SI+7*(V-1)+3,INT(W(V)/256)
20150 RETURN
20200 T(V) = T(V) + 1
20210 IF T(V)>4 OR T(V)<0 THEN T(V)=1
20220 Z1 = 8*(2↑T(V))+SD(V)*8+MR(V)*4+YS(V)*2
20230 POKE SI+7*(V-1)+4,Z1
20240 RETURN
20300 SD(V) = 1 - SD(V)
20310 IF SD(V)<>0 AND SD(V)<>1 THEN SD(V)=0
```

```
20320 GOTO 20220
20400 MR(V) = 1 - MR(V)
20410 IF MR(V)<>0 AND MR(V)<>1 THEN MR(V)=0
20420 GOTO 20220
20500 YS(V) = 1 - YS(V)
20510 IF YS(V)<>0 AND YS(V)<>1 THEN YS(V)=0
20520 GOTO 20220
20600 Z1 = 1:GOTO 20620
20610 Z1 = -1
20620 A(V) = Z1 + A(V)
20630 IF A(V)>15 THEN A(V)=0
20640 IF A(V)<0  THEN A(V)=15
20650 Z1 = 16*A(V)+D(V)
20660 POKE SI+7*(V-1)+5,Z1
20670 RETURN
20700 Z1 = 1:GOTO 20720
20710 Z1 = -1
20720 D(V) = Z1 + D(V)
20730 IF D(V)>15 THEN D(V)=0
20740 IF D(V)<0  THEN D(V)=15
20750 GOTO 20650
20800 Z1 = 1:GOTO 20820
20810 Z1 = -1
20820 S(V) = Z1 + S(V)
20830 IF S(V)>15 THEN S(V)=0
20840 IF S(V)<0  THEN S(V)=15
20850 Z1 = 16*S(V)+R(V)
20860 POKE SI+7*(V-1)+6,Z1
20870 RETURN
20900 Z1 = 1:GOTO 20920
20910 Z1 = -1
20920 R(V) = Z1 + R(V)
20930 IF R(V)>15 THEN R(V)=0
20940 IF R(V)<0  THEN R(V)=15
20950 GOTO 20850
21000 FF = FF +IN
21010 IF FF>MW OR FF<0 THEN FF=0
21020 POKE SI+21,FF-INT(FF/16)*16
21030 POKE SI+22,INT(FF/16)
21040 RETURN
21100 FR = FR + 1
21110 IF FR>15 OR FR<0 THEN FR=0
21120 Z1 = 16*FR+FO(1)+FO(2)*2+FO(3)*3
21130 POKE SI+23,Z1
21140 RETURN
21200 FO(V) = 1 - FO(V)
21210 IF FO(V)<>1 AND FO(V)<>0 THEN FO(V)=0
21220 GOTO 21120
21300 V3 = 1 - V3
21320 IF V3<>0 AND V3<>1 THEN V3=0
21340 Z1 = V3*128+FT*16+VL
21380 POKE SI+24,Z1
```

```
21390 RETURN
21400 PRINT CHR$(147) :PRINT "FILTER TYPE"
21410 PRINT"1=LOW,2=HIGH,3=REJECT,4=BANDPASS"
21420 INPUT FT:IF FT>7 OR FT<0 THEN GOTO 21400
21440 Z1 = V3*128+FT*16+VL
21480 POKE SI+24,Z1
21490 RETURN
21500 VL = VL + 1
21520 IF VL>15 OR VL<0 THEN VL=0
21540 Z1 = V3*128+FT*16+VL
21580 POKE SI+24,Z1
21590 RETURN
22000 V = V + 1
22020 IF V>3 OR V<1 THEN V=1
22090 RETURN
22100 INPUT "INCREMENT SIZE";IN
22190 RETURN
22200 IN = -IN
22290 RETURN
22300 INPUT "ATTACK DURATION";AD
22390 RETURN
22400 INPUT "RELEASE DURATION";RD
22490 RETURN
22600 RETURN
22700 RETURN
23000 Z7 = 8*(2↑T(1))+SD(1)*8+MR(1)*4+YS(1)*2+1
23010 Z8 = 8*(2↑T(2))+SD(2)*8+MR(2)*4+YS(2)*2+1
23020 Z9 = 8*(2↑T(3))+SD(3)*8+MR(3)*4+YS(3)*2+1
23040 POKE SI+4,Z7:POKE SI+11,Z8:POKE SI+18,Z9
23060 FOR Z4 = 0 TO AD:NEXT
23070 POKE SI+4,Z7-1:POKE SI+11,Z8-1:POKE SI+18,Z9-1
23080 FOR Z4 = 0 TO RD:NEXT
23090 RETURN
24000 Z1 = 8*(2↑T(1))+SD(1)*8+MR(1)*4+YS(1)*2+1
24010 Z2 = 8*(2↑T(2))+SD(2)*8+MR(2)*4+YS(2)*2+1
24020 Z3 = 8*(2↑T(3))+SD(3)*8+MR(3)*4+YS(3)*2+1
24040 POKE SI+4,Z1:POKE SI+11,Z2:POKE SI+18,Z3
24090 RETURN
24100 Z1 = 8*(2↑T(1))+SD(1)*8+MR(1)*4+YS(1)*2
24110 Z2 = 8*(2↑T(2))+SD(2)*8+MR(2)*4+YS(2)*2
24120 Z3 = 8*(2↑T(3))+SD(3)*8+MR(3)*4+YS(3)*2
24171 POKE SI+4,Z1:POKE SI+11,Z2:POKE SI+18,Z3
24190 RETURN
25000 Q = 1 - Q
25020 IF Q<>0 AND Q<>1 THEN Q=0
25090 RETURN
25100 QQ = 1 - QQ
25120 IF QQ<>1 AND QQ<>0 THEN QQ=0
25190 RETURN
29990 :
30000 DATA 78,87,84,69,77,89,65,193
```

```
30010 DATA 68,196,83,211,82,210,70,198
30020 DATA 72,79,71,214,86,201,73,66
30030 DATA 67,74,202,80,208,209,81,75
```

Modifying VOICE MAKER to Save Voice Information

As in SAVE/LOAD SPRITE, you might want to make VOICE MAKER save the settings for another program. Here's what to do:

1. Decide which key should be pressed to SAVE. One of the function keys would be good. To see the ASCII value of the function key, type:

   ```
   A$="F1"
   PRINT  ASC(A$)
   ```

   where you press the function key rather than type the characters F1. This should be 133 for function key F1. Now add that number to the last DATA statement, line 30030, after the 75. With function key F1, we would have:

   ```
   30030 DATA 67,74,202,80,208,209,81,75,133
   ```

   Note: You must change the 32 in line 1040 to 33 to allow for the new feature.

2. Now you must modify lines 4120 to 4130 to allow for one more option, and you must decide where to have the SAVE routine (26000 looks good). We might have:

   ```
   4120 IF X%>3 THEN X%=X%-3: GOTO 4150
   4130 ON X% GOSUB 25000,25100,26000
   ```

3. Last, you must save the information. The numbers (variables) that you need to save are in the video display that is made in lines 10000 to 10180. It is probably easier to just save all of them. Some easy editing of lines 10040 to 10180 could produce the following lines:

   ```
   26040 PRINT#8, VL
   26050 PRINT#8, F(1),F(2),F(3)
   26060 PRINT#8, T(1),T(2),T(3)
   26070 PRINT#8, W(1),W(2),W(3)
   26090 PRINT#8, A(1),D(1),S(1),R(1)
   26100 PRINT#8, A(2),D(2),S(2),R(2)
   26110 PRINT#8, A(3),D(3),S(3),R(3)
   26130 PRINT#8, SD(1),MR(1),YS(1)
   26140 PRINT#8, SD(2),MR(2),YS(2)
   26150 PRINT#8, SD(3),MR(3),YS(3)
   26160 PRINT#8, FF,FR
   ```

```
26170 PRINT#8, FT,V3
26180 PRINT#8, FO(1),FO(2),FO(3)
```

SAVE/LOAD SPRITE shows you how to open and close a file and store information in it. Some lines would have to be added to the lines above to do this, for example:

```
26000 INPUT "FILE NAME";SN$
26010 OPEN 8,8,8,"0:"+SN$+",S,W"
26280 CLOSE8
26290 RETURN
```

See lines 3100 to 3170 in SAVE/LOAD SPRITE for tape files.

4. In a program to use the information, you need to read the information back in the same order that it was put into the file. The easiest way is to change the PRINT#8 statements to INPUT#8. You will also need to use the information to set up the SID. If you examine VOICE MAKER, lines 2080 and 2200 to 2220, you will find the line numbers of the initialization routines, plus lines 100 to 180, which are needed to turn the voices on and off.

Note Frequencies

This program delivers the hertz you get for the POKE you made and vice versa. Try changing the 17 that is POKEd in line 5140 to other waveforms (33, 65, or 129). Notice the column formatting that is done in lines 5040 to 5080. Try playing with the 7 or 8 that sets the column width. Change it and see what happens. You could combine this type of column formatting with the centering routine used in CENTER TEXT in Chapter 5, "Ticker Tape." Proper use of these formatting routines will yield good-looking screens and printer outputs.

```
5 REM          NOTE FREQUENCIES
100 CL = 14318180  :      REM INTERNAL CLOCK
110 CS = CL / 14   :      REM COMPUTER CLOCK
120 CF = CS / (2↑24):     REM CONVERSION FACTOR
200 PRINT CHR$(147):      REM CLEAR SCREEN
210 PRINT "MUSIC FREQUENCY CONVERTER"
220 PRINT "IS THE INPUT HERTZ, POKES"
230 PRINT "OR IN SID FREQUENCY (H/P/S)?"
240 GET A$
250 IF A$="" THEN 240
260 IF A$ = "H" THEN GOSUB 2000
270 IF A$ = "P" THEN GOSUB 3000
280 IF A$ = "S" THEN GOSUB 4000
290 GOTO 200
1990 REM        HERTZ INPUT
```

```
2000 INPUT HF%
2010 SF = INT(HF%/CF):    REM HERTZ TO SID
2020 P1% = SF / 256
2030 P2% = SF - 256*P1%
2040 GOTO 5000
2990 REM        INPUT POKE VALUES
3000 PRINT "TYPE HIGH FREQUENCY,LOW FREQUENCY";
3010 INPUT P1%,P2%
3020 SF = 256*P1% + P2%
3030 HF% = SF*CF + 0.5
3040 GOTO 5000
3990 REM        INPUT SID FREQUENCY
4000 INPUT SF
4010 HF% = SF*CF + 0.5
4020 GOTO 2020
4990 REM        DISPLAY RESULTS
5000 PRINT CHR$(147):     REM CLEAR SCREEN
5010 PRINT SPC(4);"SID";SPC(10);"POKE"
5020 PRINT SPC(3);"FREQ.";SPC(6);"SI";
5030 PRINT SPC(5);"SI+1";SPC(4);"HERTZ"
5040 PRINT SPC( 7-LEN(STR$(SF)));SF;
5060 PRINT SPC( 8-LEN(STR$(P2%)));P2%;
5070 PRINT SPC( 7-LEN(STR$(P1%)));P1%;
5080 PRINT SPC( 8-LEN(STR$(HF%)));HF%;
5090 SI = 54272  :        REM SID REGISTERS
5100 POKE SI,P2%
5110 POKE SI+1,P1%
5120 POKE SI+5,0
5125 POKE SI+6,240
5130 POKE SI+24,15
5140 POKE SI+4,33
5300 PRINT:PRINT:PRINT
5310 PRINT "HIT ANY KEY TO CONTINUE,S TO STOP"
5320 GET A$
5330 IF A$="" THEN 5320
5340 POKE SI+4,16:POKE SI+24,0
5350 IF A$="S" THEN END
5360 RETURN
```

Challenges

1. Experiment with NOTE FREQUENCIES (or VOICE MAKER) to find sounds you can add to other programs. For example, SPRITE RACER could use a routine that creates motor sounds that change pitch with varying speeds.
2. Modify CHORD ORGAN, NOTE FREQUENCIES, or MUSIC BOX to input data files that contain the information needed to configure a voice to some instrument or other sound effect. Make a program that

will produce such data files. See SAVE/LOAD SPRITES as an example of reading and writing data files.

3. Add a sound menu to AMBULANCE to vary the parameters SP and RA. Display the parameters in SOUND EFFECTS also so you can hear their effects.
4. Try adding sounds to DICE, CARD DEALER, BINGO, any of the clock programs, or TIMED SCRAMBLER.
5. Try creating a simple animated cartoon with sound and sprite graphics.
6. Make up scales other than a 12-semitone equally tempered scale; for example, use 2 to the 1/23rd power and have 23 notes per octave.
7. You can use the three voices to do different things. One can be rhythm, using the noise waveform; one can be a "bass" pattern; and one can be the melody. Create a music maker that can play a song.

Appendix 1

A GUIDE TO COMMODORE PUBLIC DOMAIN SOFTWARE

Public domain software consists of computer programs that you are free to copy, provided you have access to them. The programs listed on the following pages were originally written by and for educators. They are generally available for "copy" on the user's own medium (disk or tape) at many computer stores handling the Commodore 64.

If you purchased your machine from a discount store, you may have difficulty finding access to these programs. Some computer stores, however, may be willing to assist you in making the copy at some nominal charge, usually a dollar per disk or less.

If you do not have a local distributor, your next best bet is to locate any of the computer magazines that contain Commodore 64 programming advice. These magazines contain ads or lists for "user groups" that also distribute these and thousands of other public domain programs. A user group consists of people who usually meet on a regular basis to share knowledge about a particular type of computer. There are Atari groups, Apple™ groups, Commodore groups, and so forth, in most large metropolitan areas.

Very often local user groups will meet to discuss common problems, for example, software and hardware evaluation, classes, and programming help. Many of these groups offer a collection of public domain programming to encourage your membership. You may find it hard to locate these groups. Try contacting your computer retailer or local library for information if your local outlet has no knowledge of such groups. The next appendix contains a partial listing of Commodore 64 user groups.

You may find that by the time of this book's publication, Commodore

will have many of these programs on the software racks of many computer stores. In all probability, the public domain programs will be offered only on disk because the cost of disk drives has come down dramatically in the last year. Tape users will probably have to use other means to obtain these programs.

Public domain software usually does not represent the best in programming skills. While there are some exceptions to this generalization, you must remember that writing a really "great" software program takes a long time. Most people who spend months, if not a year or more, on a program expect some financial reward for their labor. Thus, much of the programming that follows is "adequate" but is not the same quality as many commercial programs. However, there is much value in owning these programs. They offer the user "listable" copies of programs that he or she can legally modify. They are inexpensive and can be legally copied or traded with other new users.

All of the programs that follow were written for other Commodore machines, such as the VIC 20 or the Pet. Normally, such programs will not work on other computers without some modification. Commodore, however, has made this conversion for us so that all run properly on the 64.

Listing of Commodore 64 Public Domain Programs Available from Commodore

The format for the public domain listing is as follows:

ID—Program Title—Grade Level—Description

All listings are sorted first by disk ID, followed by program title, grade level, and program summary.

The grade level is Commodore's assessment of appropriate grade usage. In a few instances, the authors have substituted or amended entries.

E = Early Child	S = Senior
I = Intermediate	J = Junior
T = Trainable	C = College
P = Primary	

The program description is a very limited, one-line summary.

---- DISK AA ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
ANALYSIS 2	IS	STATISTICAL ANALYSIS: CALCULATES MEAN, AVERAGE, ETC.
ANALYSIS 1	IS	STATISTICAL ANALYSIS: CALCULATES MEDIAN, AVERAGE, ETC.
ANSWER BOX	PJT	REQUIRES QUESTION WORKSHEET. TEACHER SELECTED ANSWERS ARE IN DATA LINES.
BONDS	IS	CALCULATES BOND YIELD VALUES.
DOG	SC	EXPLORATORY SURGERY GUIDE ON A DOG.
FIGHT	SC	SIMULATION OF A TEACHER-STUDENT CONFRONTATION.
GRADES	SC	CALCULATES GRADES FOR UP TO 35 STUDENTS AND TEN TESTS.
LETTER	SC	SIMULATES DISAGREEMENT BETWEEN A TEACHER AND A PARENT.
MM PUNCTUATION	P	PUNCTUATION EXAMPLES.
MM SADSTORY	P	SENTENCE COMPLETION.
MM SHARE TIME	P	VOCABULARY DRILL.
MM VERB FORMS	P	VERB TUTORIAL.
MM VERB FORMS 2	P	VERB TUTORIAL, PART 2.
MM VERB FORMS 3	P	APPLYING VERB FORMS.
MM VERB FORMS 4	P	VERB FORMS.
MM VERB FORMS 5	P	VERB FORMS.

----- DISK AF -----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
FRENCH AID #1	JI	USER INPUTS AN ENGLISH WORD AND ATTEMPTS TO TRANSLATE IT TO FRENCH.
FRENCH AID #2	IS	USER INPUTS A FRENCH WORD AND TRANSLATES IT INTO ENGLISH.
FRENCH DRILL	JI	DRILL IN THE TRANSLATION OF ENGLISH WORDS INTO FRENCH.
FRENCH FWC	I	REVIEW OF FRENCH IRREGULAR WORDS. A FRENCH "HANGMAN" IS INCLUDED.

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
FRENCH QUIZ	S	TEST ON TRANSLATION - REPLACING NOUNS WITH PRONOUNS.
FRENCH VERBS	IS	TEST ON REGULAR AND IRREGULAR FRENCH VERBS (ADVANCED LEVEL).
FRENCH VERBS.2	J	TEST ON SELECTED VERB TENSES.
FW. SENTENCES	J	FRENCH VERB DRILL.
MELI-MELO	P	USER INPUT SENTENCE IS DISPLAYED RANDOMLY AND SLOWLY REASSEMBLED.
SCHOOL-MARM.	PJ	GENERAL KNOWLEDGE QUIZ MAKER.

---- DISK BA ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
ACCOUNTING	S	ACCOUNTING TUTORIAL.
AMORT. TABLE	IS	CALCULATE INTEREST AND AMORTIZATION TABLES ON A LOAN.
BONDS	IS	CALCULATES BOND YIELDS.
BUDGET ACCOUNT	SC	CONSTRUCTS A HOUSEHOLD BUDGET.
CALENDAR	SC	PERPETUAL CALENDAR - ANY MONTH, ANY YEAR.
CREDIT UNION	IS	DRILL ON CREDIT UNION INTEREST RATES.
DATES	SC	CALCULATE DAYS AHEAD OR BACK FROM A GIVEN DATE.
DEPRECIATION	IS	DEPRECIATION SCHEDULES - STRAIGHT LINE AND DOUBLE DECLINING.
FIFO	SC	FIRST IN-FIRST OUT ACCT. EVALUATION.
GROSS PAY	I	DRILL CALCULATION OF GROSS PAY.
HISTORY	IS	COMPUTER HISTORY QUIZ.
ICE CREAM	IS	BUSINESS SIMULATION.
INVESTMENTS	S	CALCULATES EFFECTS OF WITHDRAWAL AND DEPOSITS ON INTEREST EARNINGS.
LEMONADE	IS	SMALL BUSINESS SIMULATION.
LIFE TABLES	SC	CALCULATES LIFE INSURANCE AND ANNUITY TABLE FOR ANY INTEREST RATE.

---- DISK CA ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
BIG BINARY	SC	CONVERTS DIGITAL TO BINARY CODE.
COMMANDS	JS	INFORMATION AND DRILL ON PET COMPUTER.
COMP CONCEPT	IS	COMPUTER TUTORIAL WITH ANIMATION.
COMPUTING	JISC	TEST OF GENERAL COMPUTER TECHNOLOGY.
DISK COMMANDS	JISC	TUTORIAL ON THE PET DISK DRIVE.
DISK LISTER	U	UTILITY TO UPDATE MASTER DIRECTORY DISKETTE.
FEATURES QUIZ	PJIS	INSTRUCTIONS AND QUIZ ON THE PET COMPUTER.
GRAPH SUBROUTINE	S	DRAWS GRAPHS IN PET HI-RES.
HEX DEC	IS	CONVERTS HEXADECIMALS TO DECIMALS AND VICE VERSA.
HEX DEMO	SC	HEX CONVERSION DEMO.
HISTORY QUIZ	IS	COMPUTER HISTORY QWIZ.
HYPO. AUTO	IS	SIMULATION OF A MACHINE LEVEL LANGUAGE OPERATION.
KEYBOARD	PJ	KEYBOARD TEST.
PLOTTING	S	PLOTTING EXERCISES.
PROGRAM LISTER	S	ALPHABETIZER OF USER INPUT.

---- DISK CB ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
RND. GENERATOR	IS	DEMONSTRATES RANDOM NUMBER GENERATOR.
SIMULATION	S	COMPUTER FLOW CHART SIMULATION.
STRINGS	PJIS	DEMONSTRATES USE OF STRING VARIABLES ON THE PET COMPUTER.
TURTLE 1	JIS	SIMULATION OF LOGO TURTLE GRAPHICS ON THE PET COMPUTER.
TURTLE 2	JIS	SIMULATION OF LOGO TURTLE ON THE PET COMPUTER, PART 2.

---- DISK EA ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
A OR AN	PJ	DRILL ON THE INSERTION OF "A" OR "AN" BEFORE VARIONYMS OF WORDS.
APHORISMS	IS	CREATES APHORISM OF RANDOMLY CREATED WORDS.
B'BALL MADLIB	J	TUTORIAL ON PARTS OF SPEECH.
COMPOS. POETRY	J	COMPUTER COMPOSES POETRY.
CONCENTRAT. WORD	PJ	MEMORY MATCHING GAME.
CONCENTRATION	PJ	VERSION OF POPULAR WORD GAME.
DEFMATCH	PJ	MATCHING DRILL BETWEEN SIX WORDS AND THEIR DEFINITIONS.
ENG. MONSTER	S	WORD ASSOCIATION GAME.
FLASHER	PJI	A WORD IS FLASHED ON THE SCREEN FOR A SHORT PERIOD OF TIME.
GRAMMAR	IS	TEST ON THE PARTS OF SPEECH.
HAIKU	JIS	HAIKU GENERATOR.

---- DISK EB ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
HANGMAN	J	WORD GUESSING GAME.
HANGMAN 2	JIS	FIVE CATEGORY WORD GUESSING GAME.
HOMOCONCENTRAT.	P	A CONCENTRATION TYPE GAME.
INIT DIGRAPH	D	MULTIPLE CHOICE DIGRAPH DRILL.
JOTTO	JI	USER ATTEMPTS TO MATCH INPUT WITH COMPUTER'S HIDDEN WORD.
LETTER	P	LETTER GUESSING GAME WITH COMPUTER GENERATED CLUES.
MADLIB	JI	NONSENSE STORY GENERATOR.
MATCHING	PJ	WORD DISTINGUISHING DRILL.
MEDIAL VOWELS	PJ	MULTIPLE CHOICE VOCABULARY TEST WITH THE USE OF MEDIAL VOWELS.

MISSPELLING 5	J	IDENTIFICATION AND CORRECTION OF MISSPELLED WORDS.
MISSPELLING 6	J	IDENTIFICATION AND CORRECTION OF MISSPELLED WORDS.
MM 2LADVF	P	DRILL TUTORIAL OF APPLYING VERB FORMS.

---- DISK EC ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
MM ADVBFORM	P	TUTORIAL ON ADVERBS.
MM CRCOMP	P	IDENTIFICATION OF TYPES OF QUESTIONS.
MM DARK WOOD	P	VOCABULARY DRILL.
MM HOMONYMS	P	WORDS THAT SOUND THE SAME.
MM LADVF	P	VERB FORMS.
MM MUGS	P	VOCABULARY STUDIES.
MM MUGS WM	P	VOCABULARY STUDIES.
SPD SPELLING4	P	SPEED SPELLING QUIZ.
SPD SPELLING5	P	SPEED SPELLING QUIZ.
SPD SPELLING6	P	SPEED SPELLING QUIZ.
SPD SPELLING7	I	SPEED SPELLING QUIZ.
SPD SPELLING8	I	SPEED SPELLING QUIZ.
SPEED READ 2	I	SPEED READING OF PHRASES.
SPELL MEAN 5	J	MULTIPLE CHOICE QUIZ OF WORD MEANINGS (GRADE FIVE).
SPELL MEAN 6	J	MULTIPLE CHOICE QUIZ ON WORD MEANINGS (GRADE SIX).

---- DISK ED ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
MM VERB FORMS 6	P	DRILL ON APPLYING VERB FORMS.

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
MM VERB FORMS 7	P	DRILL ON APPLYING VERB FORMS.
MM VERB FORMS 8	P	DRILL ON APPLYING VERB FORMS.
MM VERB FORMS 9	P	DRILL ON APPLYING VERB FORMS.
MM WORD MEANS	P	TECHNIQUES OF SENTENCE COMPLETION.
NEW TACHISTO	JI	SHORT PHRASES BRIEFLY FLASHED ON SCREEN - STUDENT MUST REPEAT PHRASE.
NOUNS	DT	TUTORIAL AND QUIZ ON NOUNS.
P'BLEM - P'NOUN	JI	STUDENT MUST PICK CORRECT PRONOUN FOR SENTENCES.
PETPITPATPOT	I	STUDENT MUST FIND THE WORDS BEGINNING WITH PET, PIT, PAT, OR POT.
PARTS SPEECH	JI	REVIEW OF NOUNS, ADJ., VERBS, AND PREPOSITIONS.
PLURALS	J	RULES ON FORMING PLURALS.
PRGM. LISTER	C	ALPHABETIZES ANY LIST TO A PRINTER.
READ LEV & EVAL.	PISC	ENTER ANY PASSAGE AND COMPUTER WILL DETERMINE THE READING LEVEL.
READER	IS	READING TESTER.
REMEMBERING	PJ	TEST ON MEMORY OF SHAPES, LETTERS, AND WORDS.

---- DISK EE ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
RHYMECONC.	JI	GAME ON LEARNING HOMONYMS.
RHYMING	P	TESTING ON DISTINCTION BETWEEN RHYMING AND NON-RHYMING WORDS.
ROMEO AND JULIET	IS	TEST ON SHAKESPEARE'S "ROMEO AND JULIET".
S'PG ERRORS 4	J	LOCATION OF MISSPELLED WORDS IN LIST.
S'PG ERRORS 5	J	LOCATION OF MISSPELLED WORD IN LIST.
S'PG ERRORS 6	J	LOCATION OF MISSPELLED WORDS IN LIST.
S'PG ERRORS 8	J	LOCATION OF MISSPELLED WORDS IN LIST.

S-HYPHEN	JI	SPELLING QUIZ ON HYPHENATED WORDS.
S-SPELL	JI	SPELLING DRILL.
SCHOOL-MARM	PJ	GENERAL KNOWLEDGE QUIZMASTER.
SCRAMBLE 4	J	TASK TO UNSCRAMBLE A WORD.
SCRAMBLE 5	I	TASK TO UNSCRAMBLE A WORD.
SCRAMBLE 6	J	TASK TO UNSCRAMBLE A WORD.
SCRAMBLE 7	J	TASK TO UNSCRAMBLE A WORD.
SCRAMBLE 8	I	TASK TO UNSCRAMBLE A WORD.

---- DISK EF ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
SHAKESPEARE QUIZ	IS	DRILL ON SHAKESPEARE PLAYS.
SNOWYDAYNOUNS	IS	STUDENT SELECTS NOUNS FROM A PICTURE.
SPD SPELLING2	P	SPEED SPELLING QUIZ.
SPD SPELLING3	P	SPEED SPELLING QUIZ.
MRK STATS	C	TEACHER STAT PACKAGE FOR GRADES.
MARKS	C	TEACHER GRADEBOOK - TAPE STORAGE.
NOTES	C	TEACHER GRADE PROGRAM.
SEX ED.	SC	SEX EDUCATION.

---- DISK EG ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
SPELL MEAN 7	I	MULTIPLE CHOICE TEST ON WORD MEANINGS.
SPELLING BEE	I	SPELLING WORDS ARE FLASHED ON SCREEN.

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
SPELLINGTUTOR	PJI	SPELLING WORDS ARE INPUT BY USER. COMPUTER OMITS LETTERS, ETC.
SWAP NEW ROM	D	ALPHABETIZER.
SYLLABLE	PJ	SYLLABLE DRILL.
SYNONYMS	JI	SYNONYM DRILL AND TUTORIAL.
T-HYPHEN	JI	USED WITH "S-HYPHEN" PROGRAM TO CREATE A TEST.
T-SPELL	JI	CREATES SPELLING WORD FILE FOR USE WITH "S-SPELL".
TWENTY QUESTIONS	PJ	GENERAL TEST TAKER MADE FOR THE PET COMPUTER.
TWO TO TOO	PJ	DISTINCTION BETWEEN THESE THREE WORDS.
UNSCRAMBLE	J	WORD UNSCRAMBLER.
VOCAB.	J	GRADE 6 VOCABULARY DRILL.
VOCABULARY 3	J	THIRD GRADE VOCABULARY QUIZ.

---- DISK EH ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
VOCABULARY 4	J	GRADE 4 VOCABULARY QUIZ.
VOWEL MAGIC	PJ	IDENTIFICATION OF VOWELS IN ANY WORD TYPED BY THE USER.
WORD GAME	J	USER GIVES SYNONYM OF DISPLAYED WORDS.
WORD HUNT	JI	WORD GAME USING A "WANTED POSTER" FORM-T.
WORD LADDER	J	THE USER MUST GUESS THE GIVEN WORD BY CHANGING ONE LETTER AT A TIME.
WORD POWER	JIS	VOCABULARY TEST.
WORD SEARCH	JIS	WORDS SUPPLIED BY THE USER ARE HIDDEN IN A CROSSWORD PUZZLE.

---- DISK 6A ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
A BLOCK	PJI	USER IS TO MATCH ARTICLES BASED ON ATTRIBUTES.
A-MAZING	IS	CONSTRUCTION OF MAZES.
ABSTRACT	JI	A GAME SIMILAR TO "BAGELS".
ACCELERATION	S	A PHYSICS GAME WHICH REQUIRES A CALCULATOR.
AFO	PJ	GAME TO DESTROY "APO" WITH A LASER.
APPAREIL JET	JIS	A SLOT MACHINE GAME.
ARROW	PJI	TASK TO GUIDE A SNAKE TO TARGET BOXES.
ARTILLERY	JIS	TASK TO FIRE A CANNON OVER A MOUNTAIN AT AN OPPOSING PLAYER.
ATARI	PJS	ALIEN SPACESHIPS MUST BE DESTROYED.
BAGEL	JIS	TASK TO IDENTIFY A 3 DIGIT NUMBER USING COMPUTER CLUES.
BATTLESHIP	IS	PLAYER VS. A COMPUTER WITH THE OBJECT TO SINK OPPONENT'S SHIPS.
BIORHYTHM	IS	BIORHYTHM FOR ANY GIVEN MONTH.
BLACK BOX	JI	USER MUST FIND THE LOCATIONS OF MISSING MARBLES IN THE BOX.
BLACKJACK	IS	COMPUTER BLACKJACK GAME WITH GRAPHICS.
BREAKOUT	PJI	BREAK THROUGH A WALL BY DIRECTING A BOUNCING BALL. (REQUIRES PADDLES)

---- DISK 6B ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
CHASE	JIS	FOUR ROBOTS CHASE THE USER THROUGH FOUR LEVELS OF PLAY.
CIVIL BATTLES	IS	A SIMULATION OF CIVIL WAR.
CRAPS	JIS	DICE ROLLING GAME SIMULATION.
CRAZY BALLOON	PJ	GUIDE A BALLOON THROUGH A PATH OF PRICKLY STARS.

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
CYCLON BATTLE	JI	SHOOT DOWN CYCLON FIGHTERS.
DAM BUSTERS	PJ	DESTROY A DAM WITHOUT BEING SHOT DOWN BY ITS DEFENDERS.
DUCKSHOOT	PJIS	SHOOT DOWN DUCKS PASSING A FIXED GUN POSITION.
ENDGAME	IS	MATHEMATICAL GAME.
FLECHE	JI	HAND-EYE COORDINATION GAME.
FOX AND HOUND	JIS	USER PLAYS THE COMPUTER USING CHECKER-LIKE MOVES.
FROG RACE	PJI	GAMBLING TYPE GAME BASED ON EIGHT FROGS JUMPING OUT OF A BOX.
GAME	JI	TIC-TAC-TOE GAME.
GOLIWOG	JI	GUESS THE LOCATION OF A GOLIWOG HIDING IN A GRID.
GUNNER	JI	TARGET SHOOTING GAME.
HAMLET	JIS	AN "OTHELLO" TYPE GAME.

---- DISK 6C ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
HAMMURABI	IS	USER MAKES ECONOMIC DECISIONS AFFECTING THE WELFARE OF HIS STATE.
HANGMAN 1	J	WORD GUESSING GAME.
HANGMAN	I	WORD GUESSING GAME.
HANGMAN 3	JIS	WORD GUESSING GAME WITH FIVE LEVELS OF DIFFICULTY.
HANGMATH	JI	A MATH WORD GUESSING GAME.
HELLO	JI	COMPUTER INTERVIEWS USER ABOUT VARIOUS LIFE ISSUES.
HI-Q	JIS	OBJECT OF GAME TO REMOVE PEGS BY JUMPING INTO EMPTY HOLES.
IN-ORDER	JI	GUESS A THREE DIGIT NUMBER WITH THE AID OF COMPUTER CLUES.
JOTTO	JI	USER THINKS OF A WORD AND PLAYER MUST GUESS IT.
LAKES-ENG.	JI	A HANGMAN TYPE GAME USING THE LAKE DISTRICT OF ENGLAND AS THE MYSTERY WORD.

LE PERDU	PJIS	FRENCH VARIATION OF HANGMAN.
LOGIBLOCKS	JISC	LOGIC GAME.
MAGIC SQUARE	IS	USER ATTEMPTS TO LIGHT ALL BUT THE CENTER SQUARE OF A 9 SQUARE BLOCK.
MASTERMIND	JIS	A POPULAR LOGIC GAME.

---- DISK GD ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
MASTERMIND 2	JIS	A POPULAR LOGIC GAME PLAYED WITH A FIVE COLOR CODE.
MASTERMIND 3	JI	A POPULAR LOGIC GAME PLAYED ON DIFFERENT LEVELS.
MATCHES	JI	USER PLAYS THE COMPUTER IN AN ATTEMPT TO REMOVE MATCHES FROM A PILE.
METEOR	PJI	TEST OF USER REACTION TIME TO DEPRESS A KEY WHEN A METEOR FALLS.
MISS. IMPOSSIBLE	PJI	TASK TO RECOVER WALLETS BETWEEN FALLING BOMBS.
MOUSE MAZE	IJS	PLAYER MUST MOVE A MOUSE THROUGH A MAZE TO REACH A PIECE OF CHEESE.
MUGWUMPS	JI	PLAYER MUST LOCATE THE MUGWUMP BASED ON COMPUTER CLUES.
PETALS...ROSE	PJIS	PLAYER MUST DISCOVER THE RELATIONSHIP BETWEEN A ROLL OF DICE AND ITS SCORE.
PICTURES	P	SMALLER PICTURES ARE MOVED ON THE SCREEN TO PRODUCE A LARGER ONE.
PIZZA	JI	MATH GAME TO LEARN THE USE OF COORDINATE GRIDS.
PLANET PROBE	PJ	SPACECRAFT LANDING SIMULATION.
PONG	JS	A BALL DEFLECTION GAME WITH THE OBJECT TO HIT A TARGET.
PUB SILLINESS	JI	ANOTHER VERSION OF MADLIB.

---- DISK GE ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
RAGING ROBOTS	JI	PLAYER MUST ESCAPE RAGING ROBOTS.
ROAD TRACK	JI	MOVE A BALL AROUND A TRACK TO THE END, AVOIDING COLLISION.

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
ROTATE	JI	PLAYER MUST PUT A WORD TOGETHER IN CORRECT ORDER BY ROTATING 4 LETTERS.
SNAKES	G	OBJECT IS TO LOCATE A SNAKE HIDDEN UNDER A GRID.
SHARK	IS	SHARK HUNT UNDER THE GRID.
SNERD	PJ	COMPUTER CONSTRUCTS A STORY BASED ON USER INPUT.
SNOOPY	IS	LINE NUMBER GAME IN WHICH SNOOPY MUST SHOOT DOWN THE RED BARON.
SPACE PILOT	JI	DESTROY ARMS DEPOT OF AN EVIL MAGICIAN BY AERIAL BOMBARDMENT.
SPACE WEIGHTS	JI	COMPARES PLAYERS ABILITY TO JUMP AND THROW ON OTHER PLANETS.
STAR WARS	JIS	PLAYER MUST DESTROY ENEMY FIGHTERS.
STARTREK	JIS	COMPUTER SIMULATION OF A SPACE MISSION.
STARTREK IV.	JI	THE ENTERPRISE PURSUES AND ATTACKS THE KLINGONS.
SUPERDRAW	PJ	GRAPHICS DRAWING PROGRAM.

---- DISK GF ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
TIC-TAC-PRO	PJ	TIC-TAC-TOE PLAYED AGAINST THE COMPUTER.
TORP BOMBER	PJI	SIMULATION OF A B-29 SUB HUNT.
TOWER	IS	GAME IS SIMILAR TO HANOI TOWERS.
TURTLE	I	PLAYER INSTRUCTS A ROBOT TURTLE (GRAPHICS ON SCREEN).
TURTLE 2	I	PLAYER INSTRUCTS A ROBOT TURTLE.
TWENTY QUEST.	PJ	COMPUTER GIVES A QUIZ BASED ON USER INPUT.
UP THE LADDER	P	MATH DRILL GAME.
WAREHOUSE	IS	USER MANAGES A WAREHOUSE BY FILLING ORDERS, ETC.
WESTWARD HO	JI	HISTORICAL SIMULATION OF WESTWARD MIGRATION.
YELLOW LIGHT	JI	GAME TESTS PLAYER REACTION TO A YELLOW TRAFFIC SIGNAL.

---- DISK HA ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
ANCIENT HIST	S	TEST ON ANCIENT HISTORY.
ELECTION	IS	19TH CENTURY AMERICAN HISTORY SIMULATION.
FAMOUS PEOPLE	S	GENERAL TEST ON FAMOUS PEOPLE.
HISTORY QUIZ	S	TEST ON ANCIENT AND MEDIEVAL HISTORY.
MEDIEVAL HISTORY	S	TEST ON MEDIEVAL AND ANCIENT HISTORY.
MODERN HISTORY	S	TEST ON MODERN HISTORY.
PRESIDENT QUIZ	I	TEST ON U.S. PRESIDENTS.
TREND LINE	ISC	COMPUTER CALCULATES HISTORICAL TRENDS BASED ON USER INPUT.
WORLD WAR II	I	TEST ON WORLD WAR II.
WORLD WARS	I	TEST ON WORLD WARS I AND II.

---- DISK MA ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
ADD & SUB	IS	PROGRAM TEACHES THE ADDITION AND SUBTRACTION OF INTEGERS.
ADDITION RACE	J	ADDITION DRILL GAME.
ADDITION	J	RANDOM ADDITION PROBLEMS.
ADDS AND SUBS	PJ	ADDITION AND SUBTRACTION DRILLS.
AGENT BLOTTO	J	CODE BREAKING GAME USING MATH OPERATIONS.
ALGE VECTORS	S	ALGEBRAIC VECTORS.
AMORT'N TABLE	S	AMORTIZATION TABLES.
ANALYSIS	JIS	TEACHER UTILITY TO COMPUTE MEAN, MEDIAN, AND MODES OF STUDENT MARKS.
ANCOVA	S	TUTORIAL ON COVARIANCE ANALYSIS.

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
ANOVA	S	TUTORIAL ON VARIANCE ANALYSIS.
ARITHMETIC	JI	STUDENT DRILL ON BASIC MATH OPERATIONS.
ARTILLERY	JIS	SHOOT A CANNON OVER A MOUNTAIN; USER DETERMINES ANGLE AND POWDER.
ASTEROID	PJ	A TWO DIGIT ADDITION GAME.

---- DISK MB ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
AUTO ADD TCHR	J	ADDITION DRILL.
B.T.C. ADD	PJ	ADDITION DRILL WITH TIME LIMITS.
B.T.C. DECIMAL	JI	DECIMAL MULTIPLICATION DRILL WITH TIMER.
B.T.C. DIVIDE	JI	DIVISION DRILL WITH TIMER.
B.T.C. FRAC	J	FRACTION MULTIPLICATION AGAINST A TIMER.
B.T.C. MULT	PJ	MULTIPLICATION DRILL WITH TIMER.
B.T.C. PERCNT	JI	PERCENT TO FRACTION DRILL.
BAIRSTOW NTH	S	COMPUTER SOLVES THE N'TH ORDER OF POLYNOMIALS.
BALANCE	J	DRILL ON METRIC WEIGHTS.
BASE CHANGE	IS	CONVERSION OF ANY NUMBER FROM BASE TEN TO ANY BASE BETWEEN 2 AND 16.
BASIC STATIST	IS	DETERMINES STANDARD ERROR, MEAN, AND STANDARD DEVIATION.
BATTLESHIP	JIS	SINK ENEMY WARSHIPS ON A GRID.
BEADS	J	BINOMIAL DISTRIBUTION AID.
BIG ADD	J	ADDITION DRILL USING LARGE GRAPHIC NUMBERS.
BIG BINARY	IS	CONVERSION OF DECIMAL TO BINARY.
BIG DIVIDE	J	DIVISION USING LARGE GRAPHIC NUMBERS.

---- DISK MC ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
BIG MULTIPLE	J	RANDOM MULTIPLICATION DRILL WITH LARGE NUMERALS.
BIG SUBTRACT	J	SUBTRACTION DRILL WITH LARGE NUMERALS.
BIGTIME	PI	DIGITAL ALARM CLOCK.
BODMAS	PJ	MATH OPERATIONS DRILL ORDER.
BOMB ADD	PJ	USER SOLVES ADDITION PROBLEMS TO DEFUSE BOMB.
BONDS	S	CALCULATES VALUE OF SAVINGS BONDS.
BRAIN CRANE	PJ	ADDITION DRILL.
BRAIN CRANE -	PJ	SUBTRACTION DRILL.
BRAIN CRANE /	PJ	DIVISION DRILL.
CAR RACE MULT	PJ	TWO PERSON MULTIPLICATION CAR RACE.
CHANGEMAKER	J	SIMULATION OF MAKING PURCHASES IN A STORE (TOTAL, SALES TAX, AND CHANGE).
CHOICES	S	PROBABILITY UTILITY.

---- DISK MD ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
CLOCK	P	DRILL ON THE RELATIONSHIP BETWEEN DIGITAL AND CLOCK FACE TIME.
CO-ORDINATES	I	POINT GRAPHING.
COLLECTERM	IS	DRILL IN COLLECTING LIKE TERMS.
COUNT 1 TO 10	IS	DRILL IN COUNTING RATIONAL NUMBERS.
COUNT TEN	EP	GRAPHICS AID IN COUNTING TO TEN.
COUNT-FIVE	EP	TYPE ANY NUMERAL FROM ONE TO FIVE AND THAT NUMBER OF OBJECTS APPEAR.
CURVE FIT	SC	PLOTTING OF A POLYNOMIAL TO FIT USER INPUT OF POINTS.

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
DART	PJI	CHECKS ACCURACY AND SPEED OF BASIC MATH FUNCTIONS.
DATES	JISC	PROGRAM COMPUTES WHAT DAY OF THE WEEK A CERTAIN DATE WILL BE ON.
DECOMPOSITION	IS	TUTORIAL AND DRILL ON FACTORING OF TRINOMIALS BY DECOMPOSITION.
DEPRECIATION	IS	COMPUTES VARIOUS METHODS OF DEPRECIATION.
DERIV POLY	IS	USER SUPPLIES "X" OF A POLYNOMIAL AND THE COMPUTER COMPUTES FOR "Y".

---- DISK ME ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
DICE THROW	IS	COMPUTER GRAPHS DICE THROWS.
DIVISION DRILL	J	DIVISION DRILL WITH DIVISORS BETWEEN 1 AND 10.
DRILL S1	JI	METRIC CONVERSION DRILL.
DRILL	PJ	BASIC MATH DRILL.
DRILLS	PJ	BASIC MATH DRILL (+,-,/,X).
ELLIPSE-TRANS	S	COMPUTER DRAWN ELLIPSES AND TRANSFORMATIONS BASED ON USER INPUT.
ENDGAME	IS	MATH PUZZLE INVOLVING +,-,/,X.
EQN MANIPULAT	I	TUTORIAL ON THE MANIPULATION OF EQUATIONS.
EQUATIONS	I	EQUATIONS TUTORIAL.
EQUATIONS 2	IS	DRILL ON BALANCING EQUAL SUMS (USES MARBLES).
EXPONENT MULT	IS	DRILL ON THE MULTIPLICATION OF MONOMIALS.
EXPONENTS	PJ	QUIZ AND TUTORIAL ON THE MULTIPLICATION AND DIVISION OF EXPONENTS.
FACTEUR	IS	COMPUTER BREAKS ANY USER INPUT NUMBER INTO ITS PRIME FACTORS.
FACTOR TRINOM	I	QUADRATIC FACTORING.

---- DISK MF ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
FACTOR WHOLES	I	DETERMINES THE PRIME FACTORS OF WHOLE NUMBERS.
FACTORS	IS	PROGRAM RETURNS PRIME FACTOR OF USER INPUT.
FAST MATH	PJ	ADDITION/SUBTRACTION DRILL FOR TWO PLAYERS.
FLIP PROBLEM	IS	COIN FLIPPING EXPERIMENT TO DEMONSTRATE PROBABILITY.
FOIL PRACTICE	IS	DRILL ON MULTIPLYING BINOMIALS USING THE FOIL METHOD.
FRAC EST	JI	FRACTION ESTIMATION GAME.
FUN MACHINE	JI	USER MUST DETERMINE WHAT FUNCTION THE COMPUTER PERFORMED ON NUMBER INPUT.
FUNCTION	IS	USER CAN PLOT A NUMBER OF GRAPHS WITH DIFFERENT EQUATIONS.
GAUSS REDUCT	C	DETERMINATION OF VARIABLES BY USING A GAUSSIAN MATRIX OF COEFFICIENTS.
GEOMETRY	J	TEST ON GEOMETRIC SHAPES.
GEOMETRY TERMS	I	EXPLANATION OF GEOMETRIC TERMS WITH EXAMPLES.
GRAPH PLOT	S	USER DEFINED FUNCTIONS ARE PLOTTED ON A GRAPH.
GRAPHIQUE	S	A SIMULATION ON THE PROCESS OF DRAWING GRAPHS.

---- DISK MG ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
GUNNER	IS	USER MUST DETERMINE CORRECT ANGLES OF A CANNON TO DESTROY THE ENEMY.
HANGMATH	JI	A HANGMAN PROGRAM USING MATHEMATICAL TERMS.
HEXDEC	IS	CONVERTS DECIMALS TO HEX AND VICE VERSA.
HI-CALC	S	PLOTS A STRAIGHT LINE ON AN X-Y AXIS WITH TWO OR MORE INPUTS.
HI-LO	J	GUESS A NUMBER BETWEEN 1 AND 1,000,000.
HOW LONG	PJ	LENGTH RECOGNITION DRILL.

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
HOW MANY	EPT	COUNT 1 TO 10 SQUARES DISPLAYED ON SCREEN.
HURKLE	J	FIND "HURKLE" IN A 9 BY 9 GRID.
HYPERBOLA	S	PROGRAM CONSTRUCTS HYPERBOLAS BASEDON USER INPUT.
INT ADD	P	ADDITION DRILL WITH TIMER.
INTEGER & DEC	J	INTEGER AND DECIMAL ADDITION DRILL.
INTEGER ADD	J	ADDITION DRILL USING BOTH POSITIVE AND NEGATIVE NUMBERS.
INTEGER LINES	IS	PLOTTING OF THE POINT OF INTERSECTION OF 2 LINEAR EQUATIONS.

---- DISK M6 ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
INTEGERS	JI	DRILL ON INTEGERS WITH VARIOUS LEVELS OF DIFFICULTY.
INTERSECT	S	DETERMINES THE POINT OF INTERSECTION OF TWO LINES SUPPLIED BY THE USER.
IQ TEST	JISC	TEST ON MATHEMATICAL SEQUENCE.
LADDER MULT	PJ	MULTIPLICATION GAME.
LAST BOTTLE	PJI	A VERSION OF "NIM".
LAZER MATH	PJ	ADDITION GAME.
LIMIT CIRCLE	S	DETERMINES THE AREA OF A CIRCLE.
LINE GRAPH	ISC	PROGRAM WILL GRAPH UP TO 4 FUNCTIONS.
LINE OF BEST	S	PROGRAM DETERMINES THE BEST LOCATION FOR POINTS BASED ON USER INPUT.
LINEAR EQUA	IJ	PLOTS LINEAR EQUATIONS.
LINEAR SYS	S	PROGRAM SOLVES LINEAR EQUATIONS WITH USER INPUT OF 1 - 9 VARIABLES.
LONG DIVISION	I	INTEGER LONG DIVISION DRILL.

---- DISK MI ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
MAGIC SQUARE	JIS	ADDITION QUIZ GAME.
MAKING CHANGE	J	TEST ON MAKING CHANGE.
MATH DICE	ETP	DICE ADDITION GAME.
MATH DRILL	PJ	DRILL ON MATH EQUATIONS.
MATH QUIZ	PJ	TEST ON ONE AND TWO NUMBER ADDITION AND SUBTRACTION.
MATH TUTOR	PJ	DRILL ON INTEGER EQUATIONS.
MATHPACK	S	COMPUTER PERFORMS DIFFERENT MATH PROBLEMS.
MATRIX	S	TUTORIAL ON MATRIX MATH.
METER READING	JI	DRILL ON LEARNING TO READ METERS.
METRIC	JI	METRIC CONVERSION DRILL.
METRIC CON	IS	DRILL ON METRIC CONVERSION.

---- DISK MJ ----

PROGRAM TITLE	GR	PROG
MICROMATH	JIS	ADDITION AND SUBTRACTION DRILL OF INTEGERS.
MISSING NUMBER	EPT	USER MUST INPUT A MISSING NUMBER FROM A SEQUENCE OF NUMBERS.
MIXED NUMBERS	J	USER ADDS A SERIES OF MIXED NUMBERS AND REDUCES THE FRACTIONS.
MLA ARITH.	IS	A TEST ON COMPUTATION OF DECIMAL VALUES.
MONOMIAL MULT.	IS	MULTIPLICATION OF MONOMIALS WITH VARYING LEVELS OF DIFFICULTY.
MONSTER MULT.	PJ	MULTIPLICATION DRILL WITH PURSUING MONSTER.
MORTGAGE	S	COMPUTATION OF MORTGAGE TABLES.
MUNCHKIN MULT.	PJ	USER IS DRILLED ON ANY MULTIPLICATION TABLE OF HIS/HER CHOICE.

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
NUMBER GUESS	P	COMPUTER PICKS A NUMBER AND USER ATTEMPTS TO GUESS IT.
OPERATIONS	JI	A DRILL ON THE ORDER OF MATH OPERATIONS.
ORDERED PAIR	IS	A PROGRAM WHICH CREATES A TABLE OF VALUES FOR MODIFIED FUNCTIONS.
PARABOLA	S	PROGRAM CONSTRUCTS PARABOLAS BASED ON USER INPUT VARIABLES.

---- DISK MK ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
PERCENT	I	A DRILL ON CALCULATING PERCENTAGES.
PERCENT DRILL	JI	A DRILL ON PERCENT AND DECIMAL EQUATIONS.
PERIMETERS	J	A DRILL ON THE PERIMETERS OF RECTANGLES.
PI CALCULATOR	IS	CALCULATES PI TO THE THOUSANDS OF DECIMAL PLACES.
PIZZA	JI	MATH GAME TEACHING THE COORDINATE GRID CONSTRUCTION.
PLACE VALUE	J	USER PLAYS THE COMPUTER TO OBTAIN THE LOWEST SCORE IN A SUBTRACTION PROBLEM.
PLANES	S	GEOMETRY DRILL ON PLANES.
PLOT	IS	A SINGLE POINT PLOT ON THE SCREEN WITHOUT REFERENTS.
PLOTTING	S	A PLOTTING UTILITY.
POINTS	I	A DRILL ON POINT GRAPHING.
POLAR COOR.	S	TUTORIAL ON POLAR COORDINATES.
POLICE SUBT.	PJ	A MATH SUBTRACTION DRILL GAME.
POLY PLOT BAS.	S	PROGRAM PLOTS POLYNOMIAL CURVES ON THE SCREEN BASED ON USER INPUT.
POLYGON SECT.	S	A UTILITY PROGRAM WHICH CALCULATES THE PROPERTIES OF POLYGONAL SECTIONS.
POWER-FACT	IS	A UTILITY PROGRAM WHICH CALCULATES EXPONENTIALS AND FACTORIALS.

---- DISK MM ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
SHAPES	J	SHAPE RECOGNITION DRILL.
SIEVE	IS	PROGRAM GENERATES A LIST OF PRIME NUMBERS.
SIG-DIGITS	JI	DRILL ON SIGNIFICANT DIGITS.
SIGNIPONT DIG.	J	DRILL ON RECOGNITION OF A NUMBER OF SIGNIFICANT DIGITS.
SIMEQ. SOLVER	S	TUTORIAL ON SOLVING SIMULTANEOUS EQUATIONS.
SIMPLE SUBST.	IS	TUTORIAL IN THE EVALUATION OF MONOMIALS EQUATIONS.
SINE GRAPH	S	COMPUTER CONSTRUCTS SINE CURVES WITH USER VARIABLE INPUTS.
SKIER	J	ADDITION DRILL.
SLOPE AND INT.	IS	USER MUST SOLVE THE SLOPE AND INTERCEPT OF A GIVEN EQUATION.
SLOPE INTERCT.	SC	A TUTORIAL ON FINDING THE SLOPE AND X-Y INTERCEPTS OF LINEAR EQUATIONS.
SMALL MATH	J	SIMPLE ADDITION AND SUBTRACTION DRILL.
SNOOPY	JP	LINE MATH GAME BASED ON THE CARTOON "SNOOPY".
ST LINE PLOT	S	PROGRAM PLOTS AND ANALYZES USER INPUT OF A STRAIGHT LINE.
SUBTRACTION	J	SUBTRACTION DRILL.

---- DISK MN ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
TABLES	PJ	A MULTIPLICATION DRILL OF POSITIVE AND NEGATIVE NUMBERS.
TIC TAC PET	IS	PLAYER MUST SOLVE A MATH PROBLEM TO WIN A SQUARE.
TIMES TABLE	J	A DRILL ON MULTIPLICATION TABLES BETWEEN 1 AND 20.
TIMES	PJ	USER HAS ONE MINUTE TO SOLVE AS MANY MULTIPLICATION PROBLEMS AS POSSIBLE.
TREASURE ADD.	P	ADDITION DRILL GAME.
TRI.CLASS-ANG.	I	DRILL AND TUTORIAL ON TRIANGLE CLASSIFICATION BASED ON INTERIOR ANGLES.

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
TRIANGLES	S	TRIGONOMETRY DRILL.
TRINOMIAL FAC.	IS	DRILL IN TRINOMIAL FACTORING.
UP THE LADDER	PJ	ADDITION GAME.
VECTOR	S	TUTORIAL ON VECTOR ALGEBRA.
VERNIER SCALE	IS	DRILL OF VERNIER SCALES.
ZERO IN.	PJI	COMPUTER SELECTS A NUMBER AND THE USER MUST GUESS IT.

---- DISK MT ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
MM ADVBFORMS1	P	DRILL ON THE CORRECT APPLICATION OF ADVERBS.
STADIUM QUIZ	S	TEST ON STADIUMS IN NORTH AMERICA.
METEOR	PJI	COMPUTER RECORDS REACTION TIME TO A FALLING STAR.
LIFESTYLES	ISC	PROGRAM ANALYZES INFORMATION ABOUT USER'S LIFESTYLE.

---- DISK RA ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
AFRICA & ASIA	JI	TEST ON AFRICAN AND ASIAN CAPITALS.
CANADA QUIZ	JI	TEST ON CANADIAN PROVINCES AND CAPITALS.
CANADA	JIS	TUTORIAL DRILL ON CANADIAN PROVINCES AND CAPITALS.
CAPITALS	JI	MATCHING DRILL ON WORLD CAPITALS AND THEIR RESPECTIVE COUNTRIES.
CO-ORD DIST.	JIS	PROGRAM CALCULATES THE DISTANCE BETWEEN ANY TWO POINTS IN THE WORLD.
ENGLAND MAP	PJI	PROGRAM PRODUCES AN OUTLINE MAP OF ENGLAND.
FRENCH TOPICS	S	TEST ON FRENCH TOPICS.

GEOG	JI	GEOGRAPHY TEST BASED ON A COMPUTER DRAWN MAP.
GEOG TEST	JIS	GEOGRAPHY TEST OF GREAT BRITAIN.
GEOGRAPHY	JI	GEOGRAPHY TEST.
ITALIAN QUIZ	S	TEST ON ITALIAN TOPICS.

---- DISK RB ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
KOPPEN	S	DRILL ON KOPPEN CLASSIFICATION SYSTEM FOR CLIMATES.
LAKES-ENG	JI	HANGMAN GAME USING LAKE NAMES IN ENGLAND AS MYSTERY WORDS. 2 SITES.
MILEAGE	IS	PROGRAM CALCULATES THE DISTANCE BETWEEN LATITUDE AND LONGITUDE OF 2 SITES.
NORTH EAST	JI	A HANGMAN TYPE GAME BASED ON SITES IN ENGLAND.
OCEAN QUIZ	S	TEST ON OCEANS.
SLOPE	IS	GEOLOGY COMPUTATION OF THE SLOPE OF A HILL.
STATES AND CAP	JI	TEST ON STATES AND CAPITALS.
STATES AND REG.	JI	AMERICAN GEOGRAPHY DRILL.
WORLD CAPITALS	JI	TEST ON WORLD CAPITALS.

---- DISK SA ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
ACCELERATION	S	PHYSICS GAME WHICH REQUIRES THE USE OF A CALCULATOR.
ACTINIUM DECAY	S	ACTINIUM DECAY CYCLE. REQUIRES USE OF PERIODIC TABLE.
AVORM	PJI	USER NAMES THE OBJECT AS ANIMAL, VEGETABLE, OR MINERAL.
AZIMUTH & ALT	IS	USER MUST LOCATE EIGHT IMPORTANT STARS BASED ON ALTITUDE AND AZIMUTH.
BALANCE CHEM	S	PROGRAM BALANCES CHEMICAL EQUATIONS.
BALLISTICS	S	DRILL ON BALLISTICS PROBLEMS REQUIRES TRIG TABLES AND CALCULATOR.

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
BERNIE TOWER	I	SIMULATION OF CRUDE OIL SEPARATION BY BUBBLE TOWER METHOD.
BOHR ATOM	S	TUTORIAL ON BOHR ATOM STRUCTURE.
BOYLE'S LAW	S	SIMULATION AND GRAPH OF PRESSURE VARIATION ON GAS.
BUOYANCY	S	TEST ON DENSITY, FLOTATION, AND BUOYANCY.
CASCADE	J	SIMULATION OF UNDERGROUND WATER SEEPAGE.
CHARGE	IS	SIMULATION OF MILLIKAN'S OIL DROP EXPERIMENT.

---- DISK SB ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
CHEM	S	DRILL ON CHEMISTRY SYMBOLS, VALENCES, AND RATIOS IN WHICH THEY MIX.
CHEM EQUA.	S	A DRILL ON BALANCING CHEMICAL EQUATIONS.
CHEMIST QUIZ	S	DRILL ON CHEMICAL SYMIX.
CHEM EQUA.	S	A DRILL ON BALANCING CHEMICAL EQUATIONS.
CHEMIST QUIZ	S	DRILL ON CHEMICAL SYMIX.
CHEM EQUA.	S	A DRILL ON BALANCING CHEMICAL EQUATIONS.
CHEMIST QUIZ	S	DRILL ON CHEMICAL SYMBOLS, VALENCES, AND NAMES OF ELEMENTS.
CHEMIST	I	A TEST ON CHEMICAL RATIOS.
CIRCUIT	CS	SIMULATION ON ELECTRICAL CIRCUITS.
COMPOUND	S	DRILL ON IONIC COMPOUNDS.
COMPOUNDS	S	DRILL ON VARIOUS CHEMICAL FORMULAS.
DEFECT	S	TUTORIAL ON MASS DEFECT DEALING WITH A SINGLE ATOM.
E.M.T.	IS	EMERGENCY MEDICAL TRAINING DRILL.
ELECTRICAL PR.	S	DRILL ON ELECTRICAL PROBLEMS.

---- DISK SC ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
ELEMENT	S	TEST ON CHEMICAL ELEMENTS AND THEIR SYMBOLS.
ELEMENTS	S	DRILL ON CHEMICAL SYMBOLS OF THE ELEMENTS.
ENERGY	S	SIMULATION OF ELECTRONIC CONFIGURATION OF ANY ELEMENT.
ENV. PROFILE	IS	ENVIRONMENTAL PRIORITIZER TO ENVIRONMENTAL PROBLEMS.
ENZYME	S	ENZYME SIMULATION STUDY.
EQUATIONS	S	USER MUST BALANCE AN EQUATION WITH THE USE OF MARBLES ON A SCALE.
EQUIVALENTS	S	A DRILL AND TUTORIAL ON EQUIVALENTS AND NORMALITY.
FAMILY	S	GENETICS SIMULATION.
FOURIER PLOT	IS	DEMONSTRATION OF FOURIER PLOT.
FUSE	SC	DRILL ON THE RELATIONSHIP BETWEEN AMPERES AND POWER RATING.
GAS EQUATIONS	S	UTILITY INVOLVING BOYLE'S LAW.
GEIGERCOUNTER	S	GEIGER COUNTER SIMULATION.

---- DISK SD ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
GRAVITY QUIZ	S	A QUIZ ON GRAVITY.
HALF LIFE	S	HALF LIFE EXPERIMENTS.
HARMONICSPLY	S	DISPLAYS HARMONICS.
HEAT SOLVER	S	SOLVES FOR SPECIFIC HEAT AND FUSION HEAT PROBLEMS.
INORG CHEM	SC	INORGANIC CHEMISTRY DRILL.
INTERFERENCE	I	A SIMULATION ON THE INTERFERENCE OF WAVES.
ION	S	TEST ON ION CHARGES AND FORMULAS.

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
KINEMATICS	S	PROBLEMS CONCERNING THE MOTION OF A BALL THROWN VERTICALLY UPWARDS.
LOCKEY	S	STUDY OF ENZYME ACETYLCHOLINESTERASE.
MALARIA	IS	SIMULATION OF A POPULATION INFECTED WITH MALARIA.
MARBLE STAT.	IS	SIMULATION OF A PROBABILITY MACHINE.
METER READ	JS	INSTRUCTION ON HOW TO READ A METER.
METRIC VOLUME	J	DRILL ON METRIC VOLUME CONVERSIONS.

---- DISK SE ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
MICROSCOPY	IS	TUTORIAL ON THE OPERATION OF A MICROSCOPE.
MOLAR	S	COMPUTER CALCULATES MASS OF MOLAR BASED ON USER INPUT.
MOLECULE RACE	IS	A SIMULATION OF THE SPEED BETWEEN TWO MOLECULES.
MOLECULES	SC	STUDY OF MOLECULAR STRUCTURE.
MOLECULES 2	S	TEST ON MOLECULES AND THEIR SHAPES.
MOMENTUM	S	DRILL ON MOMENTUM PROBLEMS.
MOTION PROB.	S	DRILL ON MOTION PROBLEMS.
MOTORCYJUMP	JI	A MOTORCYCLE JUMP SIMULATION.
MULTIMICRO	S	TUTORIAL ON A MICROMETER GAUGE AND A MULTIMETER.
MUTANT	IS	PEPPER MOTH MUTATION STUDY.

---- DISK SF ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
NICHE	IS	USER MUST PLACE A VARIETY OF ANIMALS IN THEIR PROPER NICHE.
NOMENCLATURE	SC	CHEMISTRY DRILL ON COMPOUNDS.

OHM2	SC	TEST ON OHM'S LAW.
PEND 1	S	SIMULATION OF VARIOUS FACTORS ON A PENDULUM.
PERCENT	SC	CHEMISTRY UTILITY PROGRAM WHICH CALCULATES PERCENT OF COMPOSITION BY MASS.
PERIODIC PROB.	S	BAR GRAPH DISPLAY OF THE PERIODIC TABLE.
PERIODIC TABL.	S	DRILL ON LEARNING THE PERIODIC TABLE.
PET NCL REACT.	S	NUCLEAR POWER PLANT SIMULATION.
PH PROBLEMS	S	TUTORIAL ON DETERMINING THE "PH" OF VARIOUS SOLUTIONS.
PHOTOSYNTHES	S	A SERIES OF PHOTOSYNTHESIS EXPERIMENTS.

---- DISK SG ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
POLLUTION	S	A SIMULATION OF OXYGEN AND WASTE IN A BODY OF WATER.
RATE	IS	PROGRAM EXAMINES THE EFFECTS OF CHANGES IN RATE CONSTANTS OF REACTIONS.
REFLEX TIMER	EPJ	A TEST OF USER REFLEX TIME.
REG PWR SUP.	C	DESIGN REGULATED POWER SUPPLIES.
REMDL NOMENCL.	S	REMEDIAL CHEMICAL NOMENCLATURE PROGRAM.
RESISTORS	S	COMPARISON OF OHM'S LAW WITH RESISTORS.
RESOLV'N TIME	S	RADIATION TIME PROBLEMS.
RESONANCE	S	DRILL ON RESONANCE REQUIRES THE USE OF A CALCULATOR.
SC-NOTATION	IS	DRILL ON POWER NOTATION.

---- DISK SH ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
SIG-DIGITS	JI	DRILL ON SIGNIFICANT DIGITS.
SMPLEPENDULUM	S	SIMULATION OF SIMPLE PENDULUM EXPERIMENTS.

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
SPECIFIC HEAT	S	UTILITY TO AID IN MARKING OF LAB TEST ON SPECIFIC HEAT CAPACITY.
STOICH	S	PROGRAM TO SOLVE STOICHIOMETRIC CALCULATIONS.
TEMP. CONVERT.	S	TEST ON KELVIN AND CELSIUS TEMPERATURE CONVERSIONS.
TITRATE	S	TITRATION EXPERIMENT.
TWENTY QUEST.	PJ	USER SELECTS A CATEGORY AND IS ASKED TWENTY QUESTIONS.
USPOP.	IS	A SIMULATION OF U.S. POPULATION GROWTH.
VERNIER SCALE	JI	TUTORIAL ON READING A VERNIER SCALE.
WATER II	IS	WATER RESOURCE MANAGEMENT PROGRAM.
WAVES 3	SC	DOUBLE SLIT LIGHT INTERFERENCE EXPERIMENT.
WEATHER MAN	S	DETERMINES HUMIDITY INDEX, RELATIVE HUMIDITY, AND WIND CHILL FACTOR.
YOUNG	I	SIMULATION OF YOUNG'S SLIT EXPERIMENT.

---- DISK TA ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
BIG OHM'S LAW	IS	TEST OF OHM'S LAW.
CIRCUIT 3	IS	PROGRAM AIDS IN THE CALCULATION OF D.C. REGISTER WORK.
CIRCUIT 4	IS	SIMULATION OF CAPACITOR DISCHARGE THROUGH A RESISTOR.
DFW RESIST.	IS	DRILL ON PARALLEL AND SERIAL RESISTORS.
DRIVER EDUCAT.	IS	TEST ON DRIVER'S HANDBOOK.
ELECTRICAL PR.	S	TEST ON VARIOUS ELECTRICAL PROBLEMS.
FUSE	SC	DRILL ON AMPERES AND POWER RATING.
METER READ.	IS	INSTRUCTIONS ON HOW TO READ A METER.
MORSE CODE	S	TEST ON MORSE CODE.
OHM2	SC	TEST ON OHM'S LAW.
PHOTO LOG	ISC	PROGRAM ASSISTS IN ORGANIZING PHOTO INFO IN DEVELOPING ROLLS OF FILM.

---- DISK TB ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
SIMULATION	SC	SIMULATION OF A COMPUTER FOLLOWING A FLOW CHART.
RESISTORS	S	OHM'S LAW AND RESISTORS ARE REVIEWED.
RESIST TEST	IS	RESISTANCE CALCULATION DRILL.

---- DISK UA ----

PROGRAM TITLE	GR	PROGRAM DESCRIPTION
PRGM. LISTER	PC	PROGRAM ALPHABETIZES USER INPUT.
PLOT	IS	A SINGLE PLOT PROGRAM WITHOUT REFERENTS.
HOME ENERGY	S	PROGRAM HELPS IN THE HOME CONSERVATION OF ENERGY.
GRAPH SUBRTN.		GRAPHICS SUBROUTINE FOR PET COMPUTER.
GRAPH PRINT		CONSTRUCTION OF A BAR GRAPH BASED ON USER VARIABLES.
FEATURES QUIZ	T	TEST ON THE PET/CBM COMPUTER.
DUM 5.Ø		UTILITY PROGRAM TO PERFORM OPERATIONS ON A DISK.
DISK LISTER		UTILITY TO UPDATE MASTER DIRECTORY.
COPY D FILES		PROGRAM TRANSFERS FILES AND PROGRAMS BETWEEN DISKS.
CHECK DISK		THIS UTILITY CHECKS A DISK FOR BAD BLOCKS.
BAIRSTOW NTH	S	DETERMINES THE N'TH ORDER OF POLYNOMIALS.
ANALYSIS 2	JIS	PROGRAM COMPUTES THE MEDIAN AND AVERAGE OF STUDENT GRADES.
ANALYSIS	JIS	COMPUTES THE MEAN AND MEDIAN OF STUDENT GRADES.

Appendix 2

COMMODORE USER GROUPS

This listing of Commodore user groups is probably incomplete. You will find, however, periodic updates listed in the magazine *Compute!'s Gazette.* These groups are listed alphabetically by state. It is important to remember that most of the user groups exist with a minimum operating budget. If you expect a reply to your inquiries, it is necessary to include a stamped, self-addressed envelope.

Likewise, bear in mind that not everyone stays up to 2 A.M. playing with their computers. If you call, do so at reasonable hours.

ALABAMA

Huntsville Alabama Commodore Komputer Society (HACKS)
% Hal Carey
9002 Berclair Rd.
Huntsville, AL 35802
(205) 883-0223

CALIFORNIA

Commodore Users Group
% Gilbert Vela
4237 Plumeria Ct.
Santa Maria, CA 93455
(805) 937-4174

Pals
% Jo Johnson
886 So. K
Livermore, CA 94550

San Fernando Valley Commodore Users Group (SFVCUG)
% Thomas Lynch
21208 Nashville
Chatsworth, CA 91311
(213) 709-4736

San Luis Obispo VIC 20/64 Computer Club
1766 9th St.
Los Osos, CA 93402
(805) 528-3371

FLORIDA

Brandon User Group
% Paul Daugherty
108 Anglewood Dr.
Brandon, FL 33511
(813) 685-5138

Central Florida Commodore Users Group
% Stephen K. McHaney
P.O. Box 15949
Orlando, FL 32858
(305) 298-4709

Commodore Computer Club
% Chuck Fechko
P.O. Box 21138
St. Petersburg, FL 33742

Gainesville Commodore Users Group
% Louis Wallace
P.O. Box 14716
Gainesville, FL 32604

Miami 64 Users Group
% Dr. Eydie Sloane
P.O. Box 561589
Miami, FL 33256
(305) 274-3501

South Sarasota County Users Group
% Frank Topping
1859 Neptune Dr.
Englewood, FL 33533
(813) 666-2132

GEORGIA

VIC Educators Users Group
% Dr. Al Evans
Cherokee County Schools
110 Academy St.
Canton, GA 30114

IDAHO

Commodore Users
% L. Jones
548 E. Center
Pocatello, ID 83201
(208) 233-4294

S.R.H.S. Computer Club
% Barney Foster
Salmon River H.S.
Riggins, ID 83549

ILLINOIS

Commodore 64 Users Group
% Gus Pagnotta
Glen Ellyn, IL 60137

INDIANA

Commodore Hardware Users Group
% Tim Renshaw
9651 E. 21st St.
Indianapolis, IN 46229

NICE
% Eric Bean
927 S. 26th
South Bend, IN 46615
(219) 288-2101

PET/64 Users
% J. Brinson
10136 E. 96th St.
Indianapolis, IN 46256

IOWA

Commodore Computer Users Group of Iowa
Curtis Shiffer (President)
P.O. Box 3140
Des Moines, IA 52808

Quad City Commodore Computer Club
% John N. Yigas
1721 Grant St.
Bettendorf, IA 52722
(319) 355-2641

Siouxland Commodore Club
% Gary Johnson
2700 Sheridan St.
Sioux City, IA 51104
(712) 258-7903

KENTUCKY

The Commodore Connection
% Jim Kemp
1010 S. Elm
Henderson, KY 42420
(502) 827-8153

MARYLAND

Hagerstown User Group
% Joseph Rutkowski
23 Coventry Ln.
Hagerstown, MD 21740
(301) 797-9728

Long Lines Computer Club
% Tom Davis
323 N. Charles St.
Room 201
Baltimore, MD 21201
(301) 547-2570

VIC & 64 Users Group
% Tom DeReggi
21000 Clarksburg Rd.
Boyds, MD 20841
(301) 428-3174

MASSACHUSETTS

MASSPET Commodore User Group
% David Rogers
P.O. Box 307
East Taunton, MA 02718

The Boston Computer Society
% Commodore Users Group
Three Center Plaza
Boston, MA 02108
(617) 367-8080

MICHIGAN

South Computer Club
% Ronald Ruppter
South Junior H.S.
45201 Owen
Belleville, MI 48111

VIC for Business
% Michael Marotta
6027 Orchard Ct.
Lansing, MI 48910
(517) 394-2345

MISSOURI

The Commodore Users Group of St. Louis, Inc.
% Mark Cureton (C64 Subgroup)
P.O. Box 6653
St. Louis, MO 63125
(312) 752-1681

Mid-Missouri Commodore Club
1804 Vandiver Dr.
Columbia, MO 65201
(314) 474-4511

NEW JERSEY

ACGNJ Users Group
% Joseph Pylka
30 Riverview Terrace
Belle Mead, NJ 08502

NEW MEXICO

New Mexico Commodore Users Group
% Danny Byner
6212 Karlson N.E.
Albuquerque, NM 87113
(505) 821-5812

NEW YORK

Capitol District Commodore 64/VIC 20 Users Group
% William Pizer
363 Hamilton St.
Albany, NY 12210
(518) 436-1190

Folklife Terminal Club
% Bob Krebs
P.O. Box 2222-r
Mt. Vernon, NY 10551

Hudson Valley Commodore Club
1 Corwin Place
Lake Katrine, NY 12449
(914) 246-8936

KINGCOMP
% Dr. Lewis Levitt
1250 Ocean Ave.
Brooklyn, NY 11230
(212) 859a-3030

New York Commodore Users Group
% Ben Tunkelang
380 Riverside Dr., 7Q
New York, NY 10025

SUPERSTAR'S User Group
% Jean Coppola
151-28 22nd Ave.
Whitestone, NY 11357

VIC 20/Commodore 64 Users Group
% Pete Lobl
31 Maple Dr.
Lindenhurst, NY 11757
(516) 957-1512

NORTH CAROLINA

Commodore 64 Research Triangle Users Group
6164 Falls of Neuse Rd.
Raleigh, NC 27609
(919) 787-5748

Commodore User Group
% Tom Stidham
P.O. Box 749
Durham, NC 27702
(919) 489-4155

Micro Computer Users Club
% Joel Brown
Rt. #1, Box 134 A
Boonville, NC 27011

VIC 20 & 64
% Tim Gromlovits
Rt. 11, Box 686E
Hickory, NC 28601

OHIO

Commodore Computer Club of Toledo
% Gerald Carter
734 Donna Dr.
Temperance, MI 48182
(313) 847-0426

Commodore 64 and VIC 20 User Group
% Bill Novak
2299 W. 11th
Cleveland, OH 44113

Dayton Area Commodore Users Group
% Bernie Worby
4154 Tareyton Dr.
Bellbrook, OH 45305
(513) 848-2065

Licking County 64 Users Group
323 Schuler St.
Newark, OH 43055
(614) 354-1327

OKLAHOMA

Commodore Hobby Users Group
% Annette Hinshaw
Tulsa Computer Society
P.O. Box 1133
Tulsa, OK 74101
(918) 627-8994

Southwest Oklahoma Computer Club
% Gary Crowell
P.O. Box 6646
Lawton, OK 73505

OREGON

Oregon Commodore Computer Users Group
% John Jones
2134 N.E. 45th Ave.
Portland, OR 97213
(503) 281-4908

Astoria Commodore Users Group
% Kent Poulsen
Rt. 3, Box 75
Astoria, OR 97103
(503) 325-5685

Gresham VIC/64 Users Group
% Bob Brown
625 NW Battaglia
Gresham, OR 97030
(503) 666-4131

Lake Oswego PET/C64 Users Group
% John Jones
2134 N.E. 45th
Portland, OR 97213
(503) 281-4908

Lane Country Users Group
% Peter Rocksvold
584 Sierra
Eugene, OR 97402
(503) 688-1537

Mt. Hood Community College PET/64 Users
% T. Scheinman
4044 N.E. Couch
Portland, OR 97232
(503) 667-7196

Tillamook VIC/C64 Users Group
% George Shiveley
1009 5th St.
Tillamook, OR 97141
(503) 842-4515

Yamhill County Commodore Users
% Dan Linscheid
Rt. 2, Box 246
Sheridan, OR 97378
(503) 843-2625

Albany/Corvallis C64 Users Group
% Jon North
6277 Looney Ln. S.W.
Albany, OR 97321
(503) 928-5099

PENNSYLVANIA

Compstars
% Larry Shupinski
440 Manatawny St.
Pottstown, PA 19464

Pittsburgh Commodore Group
% Joel Casar
2015 Garrick Dr.
Pittsburgh, PA 15235

Westmoreland Commodore Users Club
% Jim Mathers
3021 Ben Venue Dr.
Greenburg, PA 15601
(412) 836-2224

RHODE ISLAND
Newport Commodore Club
% Dr. Matt McConeghy
10 Maitland Ct.
Newport, RI 02840
(401) 849-2684

TENNESSEE
Chattanooga Commodore Users Group
% Carolyn Bellah
P.O. Box 1406
Chattanooga, TN 37401
(615) 938-3773

Nashville Commodore Users Group
P.O. Box 121282
Nashville, TN 37212
(615) 331-5408

TEXAS
Commodore Houston Users Group
% Mary Howe
23502 Creekview Dr.
Spring, TX 77379
(713) 376-7000

UTAH
The Utah Commodore Users Group
% Rodney Keller
652 W. 700 N.
Clearfield, UT 84015
(801) 776-3950

VIRGINIA
Capitol Area Pet Enthusiasts (Commodore 64 Subgroup)
% William Tyler
607 Abbotts Ln.
Falls Church, VA 22046
(703) 532-8087

Fredericksburg Area Computer Enthusiasts
% Michael Parker
P.O. Box 324
Locust Grove, VA 22508

Southwestern Virginia Commodore Users
% Jim Richardson
Rt. 1, Box 12A
Elk Creek, VA 24326
(703) 655-4144

Tidewater Commodore Users Group
% Fred Monson
4917 West Grove Rd.
Virginia Beach, VA 23455

WASHINGTON

Longview, WA Commodore Users Group
% Stephen Jones
626 26th Ave.
Longview, WA 98632
(206) 577-8317

Commodore Computer Club
P.O. Box 1471
Oak Harbor, WA 98277
(206) 675-4815

NW Commodore User Group
% Richard Ball
2565 Dexter N. #203
Seattle, WA 98109
(206) 284-9417

PSACE
% Ken Tong
1800 Taylor Ave. N.
Seattle, WA 98109

Spokane Commodore Users Group
% Terry Voss
W. 1918 Boone Ave.
Spokane, WA 98000
(509) 327-7207

WISCONSIN

Commodore 64 Software Exchange Group
% Edward Rosenberg
P.O. Box 224
Oregon, WI 53575
(608) 835-5538

Table 1
Some Important CHR$ Values

See Appendix E of your *Commodore 64 User's Guide* for a complete listing.

CHR$	*CHR$*
(5) = White	(137) = F5 Function Key
(8) = Disable Commodore Key	(138) = F6 Function Key
(9) = Enable Commodore Key	(139) = F7 Function Key
(13) = Return	(140) = FB Function Key
(14) = Switch to Lower Case	(141) = Clear Screen
(18) = Reverse on	(142) = Switch to Upper Case
(19) = Clear/Home	(144) = Black
(20) = Instant Delete	(145) = Cursor
(28) = Red	(146) = Reverse off
(29) = Right Cursor	(147) = Clear/Home
(30) = Green	(156) = Purple
(31) = Blue	(158) = Yellow
(32) = Space	(159) = Cyan
(133) = F1 Function Key	(160) = Space
(134) = F2 Function Key	
(135) = F3 Function Key	
(136) = F4 Function Key	

Table 2
Colors

Current display color can be changed by POKEing a color value (0–15) into the screen color memory register (646).

0 = Black	CHR$(144)	POKE 646, 0
1 = White	CHR$(5)	POKE 646, 1
2 = Red	CHR$(28)	POKE 646, 2
3 = Cyan	CHR$(159)	POKE 646, 3
4 = Purple	CHR$(156)	POKE 646, 4
5 = Green	CHR$(30)	POKE 646, 5
6 = Blue	CHR$(31)	POKE 646, 6
7 = Yellow	CHR$(158)	POKE 646, 7
8 = Orange		POKE 646, 8
9 = Brown		POKE 646, 9
10 = Light Red		POKE 646, 10
11 = Gray 1		POKE 646, 11
12 = Gray 2		POKE 646, 12
13 = Light Green		POKE 646, 13
14 = Light Blue		POKE 646, 14
15 = Gray 3		POKE 646, 15

Table 3
Sound

See pages 152–154 of the *Commodore 64 User's Guide* for a table of values to be POKEd into the HI and LO FREQ registers to get specific musical notes. On pages 163–164 of that guide is a table similar to the following. Be wary of the table of ADSR settings on page 164 for simulating musical instruments: They don't sound right to us.

Address	*Add This to 54272*	*Interpretation of Location*
54272		Low frequency note (voice number 1)
54273	1	High frequency note (voice number 1)
54274	2	Lo pulse (voice number 1)
54275	3	Hi pulse (voice number 1)
54276	4	Waveform (voice number 1)
54277	5	Attack/decay (voice number 1)
54278	6	Sustain/release (voice number 1)
54279	7	Play a note (low frequency voice number 2)
54280	8	Play a note (high frequency voice number 2)
54281	9	Waveform (voice number 2)
54282	10	Pulse rate (hi pulse—voice number 2)
54283	11	Pulse rate (lo pulse—voice number 2)
54284	12	Attack/decay (voice number 2)
54285	13	Sustain/release (voice number 2)
54286	14	Play a note (low frequency voice number 3)
54287	15	Play a note (hi frequency voice number 3)
54288	16	Pulse rate (lo pulse—voice number 3)
54289	17	Pulse rate (hi pulse—voice number 3)
54290	18	Waveform (voice number 3)
54291	19	Attack/decay (voice number 3)
54292	20	Sustain/release (voice number 3)
54296	24	Volume control (all three voices)

Table 4
Some Input/Output Information

Joystick Control

Location to be Peeked	*Meaning*
56320	Joystick port 2
56321	Joystick port 1

```
A = PEEK(56320) AND 31
PRINT A
```

The AND 31 restricts the numbers being read between 0–31. If A exceeds 16, then the fire button on your joystick has been pressed. To determine the direction in which the joystick is being moved, a series of ANDs and IF–THEN commands can be used:

```
IF A THEN . . . (The fire button was pressed.)
IF A AND 8 THEN . . . (The joystick is tilted left.)
IF A AND 4 THEN . . . (The joystick is tilted right.)
IF A AND 2 THEN . . . (The joystick is tilted up.)
IF A AND 1 THEN . . . (The joystick is tilted down.)
```

Printer Output

Printer output commands vary between interfaces and printers. These two commands should be helpful for most users.

```
OPEN 23,4        :REM TELLS THE 64 THAT ALL OUTPUT
                  GOING TO 23 WILL GO TO THE PRINTER (4)
CMD 23: LIST     :REM PRINTER LISTS THE PROGRAM IN
                  MEMORY
PRINT#23         :REM CLEAR OUT THE LAST LINE
CLOSE 23         :REM TELL THE 64 AND PRINTER WE'RE
                  DONE
```

The above command sequence should work on the Commodore 1525 as well as printers connected with the CARDCO/? interface.

Table 5
Our Memory Sprite Number (SB) And The 64's

Our Memory Sprite Number	*Variable SB*	*CBM 64 Memory Sprite Number*	*Where in Memory It Is*
0	0	192	12288 to 12351
1	1	193	12352 to 12415
2	2	194	12416 to 12479
⋮	⋮	⋮	⋮
63	63	255	16320 to 16383

Table 6
Assigning Memory Sprites to Display Sprites

POKE these locations with the third column of Table 5 to assign a display sprite (0–7) to that memory sprite.

2040	0	2044	4
2041	1	2045	5
2042	2	2046	6
2043	3	2047	7

Table 7
Sprite Colors

The following chart will come in handy in figuring out the necessary sprite color setting POKE.

Display Sprite Number	*Memory Location*	*Sprite Colors (Numbers to POKE)*	
0	53248 + 39	0 = Black	8 = Orange
1	53248 + 40	1 = White	9 = Brown
2	53248 + 41	2 = Red	10 = Light Red
3	53248 + 42	3 = Cyan	11 = Dark Gray
4	53248 + 43	4 = Purple	12 = Medium Gray
5	53248 + 44	5 = Green	13 = Light Green
6	53248 + 45	6 = Blue	14 = Light Blue
7	53248 + 46	7 = Yellow	15 = Light Gray

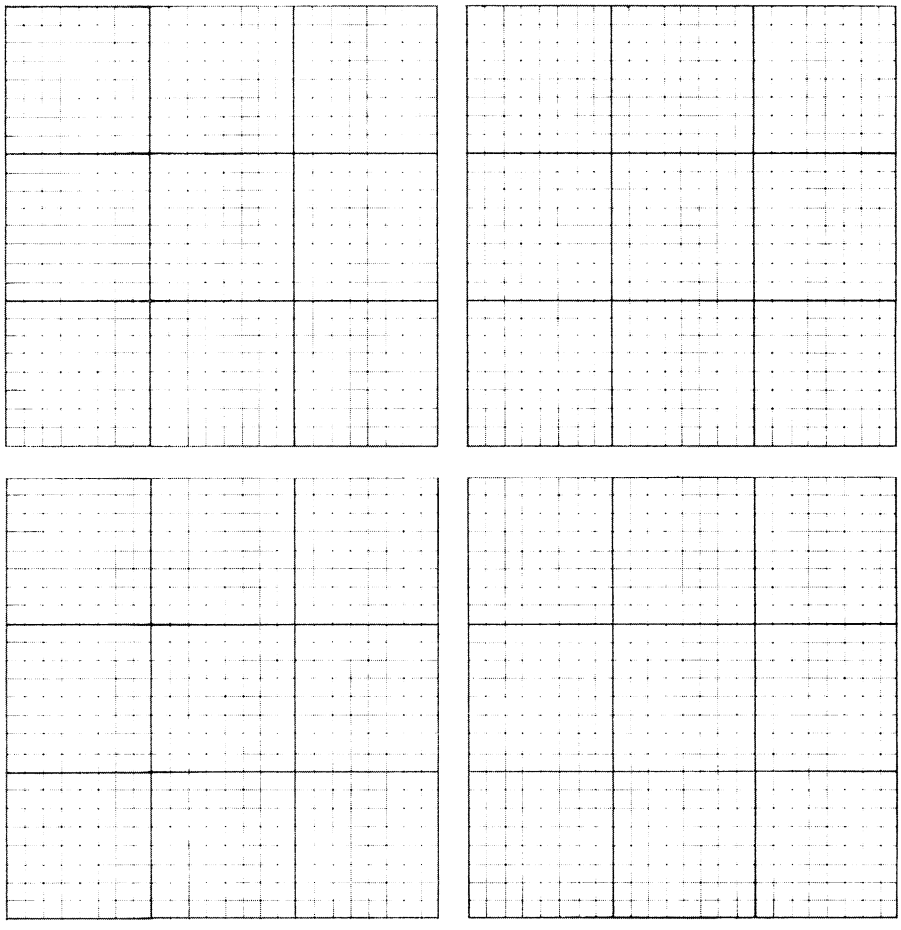
Table 8
Sprite Location

Examine the following chart for sprite vertical and horizontal memory locations.

Display Sprite Number	*Horizontal Location (+ 53248)*	*Vertical Location (+ 53248)*
0	0	1
1	2	3
2	4	5
3	6	7
4	8	9
5	10	11
6	12	13
7	14	15

Table 9
Blank Sprite Forms

The forms on this page are provided so you can more easily design your sprites. Copy this page and simply fill in the forms to design a sprite. Then set up your design as lines 200–220 in SPRITE EDITOR by filling in any character where you want a dot to be turned "on" or displayed in the sprite. Happy spriting!



INDEX

D

E

J

K

L

M

N

P

R

T

U

V

W

Z

Basic Fun for the Commodore 64 Beginner,
Arthur T. Denzau, Kent L. Forrest, and Robert P. Parks

Compatible with the Commodore 64; single disk-drive; color monitor is helpful.

COMPUTERS

The popular Commodore 64 computer's possibilities are endless—*if* you can slog through the user's manual!

This exciting new book gives novices the most *enjoyable* route for learning Commodore 64 computing! With *more than 100* easy-to-follow and ready-to-run programs, *Basic Fun for the Commodore 64 Beginner* lets you *learn by doing.* Drawing, coloring, keeping time, creating games, playing music ... these clever and *fun* programs let you capitalize on all of the Commodore 64's special features. You'll master concepts from the simple to the sophisticated as you practice writing, running, and modifying programs like "Dice Rolls," "Wild Screen," "Quick Draw," "Pizza," "Peekaboo," and "Ambulance." And, if you're adventurous, try the programming Challenges, which let you experiment with the techniques you've learned to create brand *new* effects.

Along the way, you get plenty of instructions and tips for working with the Commodore 64:

- how to use the screen editor
- how to use the disk drive
- what to do if a program won't run
- chapter-end lists of BASIC commands and programming techniques
- and much, much more.

Arthur Denzau, Kent Forrest, and **Robert Parks** are computer consultants and associate professors at Washington University in St. Louis, Missouri.

Cover design by Hal Siegel

Front plate photo courtesy of Michel Tcherevkoff

PRENTICE-HALL, Inc., Englewood Cliffs, New Jersey 07632

3

ISBN 0-13-061441-6