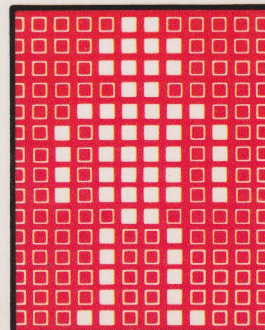
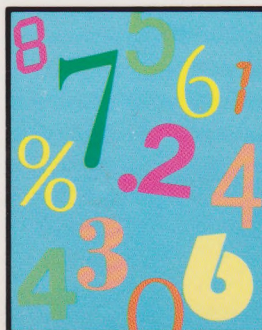
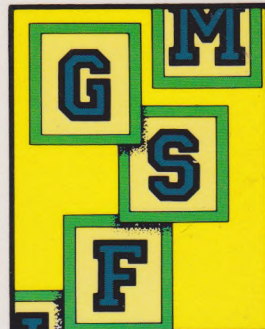
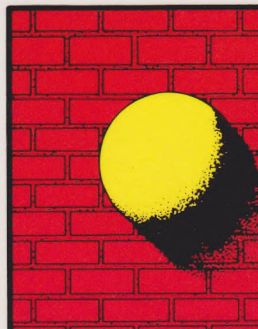
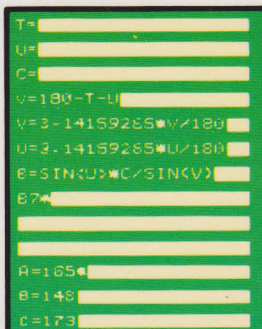


MASTERING YOUR COMMODORE 64™ THROUGH EIGHT BASIC PROJECTS

Includes ready-to-run programs on disk, with
spreadsheet, clock, music editor, screen editor,
games, and more



by the staff of MICRO magazine

MICRO magazine MASTERING YOUR COMMODORE 64™ THROUGH EIGHT BASIC PROJECTS €

MASTERING YOUR COMMODORE 64

Through Eight BASIC Projects

Robert M. Tripp
Editor

Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data
Main entry under title:

Mastering your Commodore 64 through eight basic projects.

"A Spectrum Book."

Bibliography: p.

1. Commodore 64 (Computer)—Programming.
2. Basic (Computer program language). 3. Computer programs. I. Tripp, Robert M., (Date). QA76.8.C64M37 1984 001.64'2 84-18284
ISBN 0-13-559543-6

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.

© 1984 by The Computerist, Inc.

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

A SPECTRUM BOOK

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

Original programs by Phil Daley, Robert M. Tripp, and Loren Wright.
Commodore 64 conversions and updates by Robert M. Tripp and Mark Maybury.
Layout and production by Jennifer Collins.
Illustrations by Frank Wyman.

ISBN 0-13-559543-6

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*

Table of Contents

Introduction	4
Using the Diskette	6
How to Read Program Listings	7
C64 Player	9
<i>Making Beautiful Music</i>	
Master	35
<i>A Simple Guessing Game</i>	
Word Detective	57
<i>A Word Guessing Game</i>	
Answer Machine	71
<i>The Intelligent Computer</i>	
Breakup	75
<i>An Animated Game</i>	
C64 Clock	87
<i>A Digital Time Display</i>	
Screen Editor	113
<i>Using Your Video Display</i>	
Sorting	137
<i>Information on Five Sorting Methods</i>	
MICRO Calc	161
<i>A Miniature Spread Sheet</i>	
Disk Menu	183
<i>A Useful Disk Utility</i>	
References	186

Introduction

Welcome to the world of The Commodore 64! You are about to embark on an incredible journey that will take you through the labyrinth of your microcomputer. You don't need to be a sophisticated programmer or have years of high-tech experience to enjoy this voyage; you just need an adventuresome spirit and a keen desire to learn. We hope you find this tour entertaining, stimulating, and useful.

Our itinerary includes explorations into many levels of computer programming. The absolute novice can simply 'RUN' and enjoy the programs, while the more experienced computerist can acquire new programming techniques. To get the most from this learning experience a rudimentary knowledge of BASIC programming would be helpful. This is neither a programming text nor a reference guide; teaching good programming by example is our intent. We have taken care to be consistent in our presentation of the material, including initial operating instructions, variable usage tables, and program listings in a structured format with descriptive text. We use figures, screen dumps, and short demonstration programs extensively. To become more knowledgeable about your Commodore 64, investigate the programs in the book and on diskette. A synopsis of your journey follows.

If you have ever dreamed of becoming a *note-able* musician, you will enjoy "C64 Player." This program provides you with a three-octave organ keyboard on which you can compose your own tunes or play some of your old favorites. Define your own instruments — from the sound of a guitar to an organ to a piccolo or even a sound never heard before! LOAD and SAVE the songs to diskette then regale friends with your renditions of *La Paloma*, *Allouette*, or *Loch Lomond*. Edit your songs directly on the screen, adding, deleting or changing notes, octaves, durations or voices. This will help you to fully understand the music capabilities of your Commodore 64. A musical feast awaits you as you learn music programming and the use of diskette data files.

Whether you are traveling alone or with someone else you will enjoy "MASTER," a simple guessing game for one or two players. Unlike the commercial version, this C64 game uses letters instead of colored pegs, but the rules are the same. We offer you three versions, and you can modify the program to play even more games. While trying to figure out your A's, B's, C's, and D's, you will have the opportunity to learn something about random numbers and flags.

And since there are always rules to every game, we have included "Word Detective," another guessing game that lets you figure out

what the rules are. The program introduces the concept of the computer handling letters and words, as well as numbers. A supplementary program, "The Answer Machine," allows you to ask the computer questions, which it will answer with a yes, no, or maybe. If you are endowed with a good imagination, the narrative can become quite involved and you may rival even Scheherazade or Aesop.

No journey would be complete without a sporting event of some kind; we take you to a ball game. "Breakup" uses a paddle, joystick, or the keyboard to provide you with a simple bouncing ball game. At the same time you learn how animation is accomplished with PEEKs and POKEs to the screen memory.

If you have the time, perhaps you would like to display it in giant-size digital numbers on your C64 screen. "C64 Clock" proceeds, step by step, from the simplest uses of the C64 clock to a deluxe clock that displays the time in big numerals that you can read from across the room. At the same time learn a few simple techniques for programming the cursor and using string primitives.

While traveling you may want to make pictures. We provide you with a "Screen Editor," which allows you to explore the built-in character graphic capabilities of the C64, adds a super line drawing facility, and, provides diskette support to save and restore your masterpieces.

Explore the two programs in "Sorting" that demonstrate several methods of sorting data. You will enjoy watching the C64's colorful character graphics sort bars of different lengths right in front of your eyes. The second program is a telephone directory that allows you to create new files, add, change, or delete old files, load existing files, save them to disk, and sort files in a useful manner.

Finally, sail through "MICRO Calc," a miniature spread-sheet program that makes complex, repetitive calculations a breeze. LOAD the program from the diskette and your video screen becomes a versatile calculator. Screens may be developed, saved on diskette and later loaded to perform a vast variety of useful calculations. You can use this program to experiment with BASIC functions.

We sincerely hope that you find this venture exciting and educational and that it leads you to further exploration of your Commodore 64. Now load your diskette and commence your journey. Bon voyage.

Using The Diskette

Programs may be LOADED from disk and RUN using the standard Commodore 64 commands. However, there is an easier way! Included on the *Mastering Your Commodore 64* diskette is a special program called MENU. This should be loaded as follows:

1. Insert the *Mastering Your Commodore 64* diskette into the disk drive and press the RESET key on your Commodore 64.
2. Type LOAD "MENU",8 and then press the RETURN key.
3. The MENU program will be loaded from the diskette. When the display says READY, type RUN and RETURN.
4. MENU will read the diskette directory and display all of the PROGRAMS. To execute a program (LOAD and RUN), type the PROGRAM NUMBER displayed with the menu and a RETURN. The selected program will be loaded and run from the diskette.
5. If you wish to load the program without running it, type a - (minus sign) and then the PROGRAM NUMBER.

```
DRIVE #: 0 DISK NAME: MASTERING

1+ PLAYER
2+ MASTER
3+ WORD DETECTIVE
4+ ANSWER MACHINE
5+ BREAKUP
6+ CLOCK
7+ SCREEN EDITOR
8+ BARS SORT
9+ PHONE BOOK
10+ MICRO CALC
11+ MENU

ENTER: # TO EXECUTE, - # TO LOAD
      + NEXT PAGE, C EXIT TO BASIC
SELECT PROGRAM ? _
```

6. The diskette is write protected to prevent accidental erasure of the programs. Use your own formatted diskette for saving files that you generate with the **PLAYER**, **PHONE BOOK**, **SCREEN EDITOR** or **MICRO CALC** programs.

How to Read Program Listings

This book contains a great number of program listings. To get the most from the material in the book, you should be able to follow these listings as you read the descriptive text and/or run the programs. Commodore computers allow cursor controls and other special characters to be programmed as characters within a string. In listings these appear as reverse field characters. For example, the 'clear screen' command is represented by a reversed heart. These reversed characters are often hard to identify in a printed form and do not have any relationship to the function that they represent. To make it easier to read and follow the listings, we have substituted bracketed abbreviations of the function names for these characters. These abbreviations are set off from all other text by starting with a { and ending with a }. If several abbreviations occur in a row, they will appear in one set of brackets and will be separated from each other by commas. If one abbreviated function is used several times in a succession, such as *move the cursor down six lines*, it will be represented by the abbreviated name followed immediately by the number of occurrences: {DOWN6,LEFT2}.

The following table includes the most commonly used commands. Commands not listed follow the same general rules and should be obvious from the listing context.

Expression	Function	Key(s)
{CLEAR}	Clear Screen	Shift CLR/HOME
{HOME}	Home Cursor	CLR/HOME
{DOWN}	Cursor Down	CRSR Up/Down
{UP}	Cursor Up	Shift CRSR Up/Down
{RIGHT}	Cursor Right	CRSR Left/Right
{LEFT}	Cursor Left	Shift CRSR Left/Right
{RVS}	Reverse On	CTRL RVS ON
{RVSOFF}	Reverse Off	CTRL RVS OFF
{SPACE}	Space	Space Bar
{YELLOW}	Yellow	CTRL YELLOW
{F1}	Function 1	F1
{F2}	Function 2	Shift F1



C64 PLAYER

Making Beautiful Music



Player enables you to play music on your Commodore 64. It converts it into a simulated organ with the organ keys being represented by the keys on the Commodore 64 keyboard — as shown above. Eight different 'voices' can be programmed — from a piccolo to a guitar to an oboe! The keyboard spans three complete octaves. Each note that is played is stored in memory so that it can be instantaneously replayed, edited, or saved on disk or tape for later use. You can load your song from disk or tape and replay it whenever you want to.

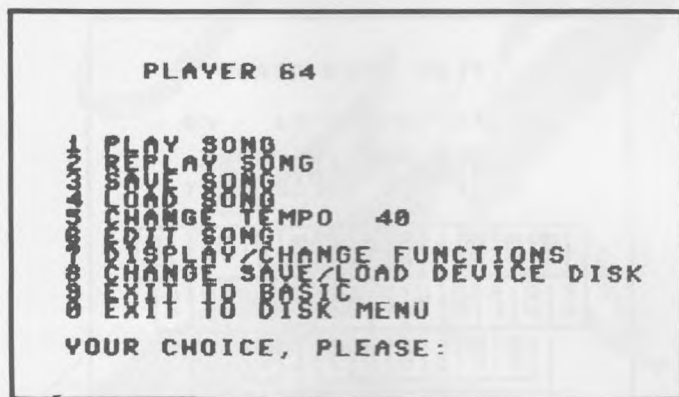
We have included a feature in Player that allows you to stop playing, go back to correct mistakes, replay the song from the beginning to the current note, and then continue playing additional notes. Absolute perfection is possible using the Song Editor, which allows you complete control over every note that has been played. You can change the voice, frequency, and/or duration of any note. You can even add and remove notes!

With the Player installed in your computer you can make your Commodore 64 an entertaining and instructive device.

Operating Instructions for Player Demonstration

The first time you use Player you should listen to the song provided on the diskette to appreciate how much can be accomplished. To do this, follow the simple steps listed here.

1. Execute PLAYER from the *Mastering the Commodore 64* diskette.
2. You will be presented with the following information.



3. Press the 4 key to select the 'LOAD SONG' option.
4. The display will ask 'WHAT IS THE NAME OF THE SONG?'.
5. Type SONG. This will cause the file named SONG.SEQ(a pre-recorded song) to be loaded from the diskette.
6. When the song has been loaded, the display will say 'LOADED SONG' for several seconds, and will then re-display the original menu.
7. Press the 2 key to select the 'REPLAY SONG' option.
8. You will now be treated to a rousing song played automatically on your Commodore 64 computer.

Using Player

After choosing the 'PLAY SONG' option from the menu, a representation of the Commodore 64 keyboard is printed on the screen in the format of a two keyboard organ. The bottom row of keys represents the lowest notes, starting with 'C' and ascending alphabetically. The second row of keys represents the sharps and flats (black keys) that correspond to the first row. Note that there are in-between keys on the second row of the Commodore 64 for every pair of first row notes. This is different than the normal organ keyboard and means that some of the second row keys do not sound when pressed (A, F, and K). These keys can be used to introduce rests into your song.

The third row of keys represents the second keyboard of the organ starting at middle 'B' and ascending to high 'C'. The top row of keys represents the sharps and flats corresponding to this second keyboard. When you have mastered the keyboard, you are well on your way to composing your own music.

At the beginning, the program waits for you to select a VOICE by pressing one of the function keys. This is one of the few times when a pause doesn't count. Once you start playing, the computer keeps track of every note and its length exactly as you play it. Practice a little bit to get the feel of the keyboard. It is not as simple as a piano, especially with the letters on the keys distracting you from what the true note is. The white keys on the display have the actual name of the note printed over the keyboard name of the key to help keep you oriented.

After you start a song, you may discover that you didn't mean to play a particular note. Fortunately there is a mistake-recovery method. As soon as you realize that you have made an error, press the space bar to pause momentarily. You will be presented with several options.

1. CONTINUE allows you to start playing the song at exactly the point where you stopped. This is a useful technique for the times when you become confused as to which note you want to play next; press the space bar to pause, regather your wits, and press 'C' to continue from where you stopped.
2. REPLAY will play the song up through the current note so that you can inspect your masterpiece as you input and make corrections if necessary. This option can be chosen as many times as you need it.
3. BACKUP is the option that allows you to make corrections. This allows you to remove one note at a time from the current song, all the way back to the beginning if you want. When you make a

mistake and press the space bar to pause, press the 'B' option and the note you are erasing will sound. Another 'B' will erase the next note, and so on. Then pressing 'C' will allow you to continue your song from the point to which you have backed up.

If, no matter how hard you try, you can't seem to get the song perfect, then the next step is to use the Song Editor. This part of the program is described in detail later in this chapter.

Other Menu Options

There are additional menu options that allow you to replay your song, load a song from or save a song to the disk, change the tempo of the song, change the storage device between disk and tape, edit the song, change the characteristics of each voice, exit to the disk menu, or quit.

Option two, REPLAY SONG will play the song currently in memory over the television or monitor speaker. The routine uses the current tempo for the speed at which to play the song. If there are no notes in the song, then 'NO SONG IN MEMORY' is printed and you are returned to the menu.

Option three is used to SAVE the current song on disk or tape. It will ask 'WHAT IS THE NAME OF THE SONG?'. Respond with the name that you want for the song. When the song is saved, you are returned to the menu. If there are no notes in the song, the current voice information will be saved.

Option four is used to LOAD a song from disk or tape. Remember that when you LOAD a new song, you will erase any song currently in memory; you must SAVE the current song (if you want to keep it for later use) before loading a new one. The 'WHAT IS THE NAME OF THE SONG?' prompt will appear. Type the name of the song you want and the Commodore 64 will load it from the disk or tape.

Option five permits you to CHANGE TEMPO. The minimum (fastest) setting allowed is '1'; the maximum (slowest) setting is 99.

Option six, EDIT SONG allows you to examine, change, insert and delete notes and voice changes in your current song. After 'playing' your selection from the keyboard, you can use this facility to make it perfect! (See Song Editor on page 14).

Option seven, DISPLAY/CHANGE FUNCTIONS lets you define the eight function keys to represent different musical voices. When first called, the current voice assignments are displayed. 'F#' is the function number; 'WF' is the wave form with 'P' for Pulse, 'T' for Triangle, 'S' for Sawtooth, and 'N' for Noise; 'AT' for attack, 'DC' for

decay, 'ST' for sustain and 'RL' for release; 'VOL' is the volume; and 'PULSE' is the percent of the time the Pulse wave form is positive. You can select any combination of these parameters for each of the eight function keys. Then, when you play your song, you select one voice at a time via the function key. You may change the voice function table at any time and then replay your song with different sounds. The current function table is saved with the song when you use the Save Song option; a new table is loaded when the Load Song is used. You can save just the function table by using the Play Song option, immediately pressing the RETURN to end the song. If you now 'save the song' to disk or tape, the function table will be saved, but there will be no song to save! Later, you can 'load the song'. This will merely load in the function table and will not disturb any song you have in memory. In this way you can easily change all of the voices without going through the 'change functions' option.

FUNCTION TABLE							
FW	WF	AT	DC	ST	RL	VOL	PULSE
1 2 3 4 5 6 7 8	P L S E S E S E	8 8 8 8 8 8 8 8	8 8 8 8 8 8 8 8	8 8 8 8 1 8 8 8	8 8 8 8 8 8 8 8	1 1 1 1 1 1 1 1	6
FW TO CHANGE [1-8] OR RETURN: ? _							

Option eight lets you switch between disk and tape. Although most of your song saving and retrieving will naturally be on the disk, the cassette tape is also supported. The currently selected device is displayed in the menu. It defaults to disk.

Option nine exits directly to BASIC if you want to examine and/or modify the PLAYER program. Any song that you have in memory will be lost, so be sure to save it first.

Option zero is used when you are finished with PLAYER and want to return to the DISK MENU.

The Song Editor

The Song Editor enables you to correct any minor (or major) mistakes that may creep into your performance. It allows complete control over every note in your song: the note, the octave, and the duration all can be changed to your specifications. You can even insert and delete notes and/or voice functions! Using the insertion feature, you can start from scratch and construct your own song without ever playing a note on the Player keyboard!

The first thing that the Song Editor does is list the notes and functions currently in memory which have either been entered from the keyboard (as in the example), or have been loaded in from a previously saved disk. The 'Next' and 'Previous' commands allow you to page forward and backward through the note tables so that you can look for the note (or notes) that you wish to change.

The '#' option allows you to change the current notes in memory, one at a time. It does not change any notes on the disk, so don't worry about making a mistake. Experiment with changing notes and then play them together to see how they sound.

Pressing the return at the 'NEW LENGTH:' prompt will return you to the main menu. Pressing 'F' will allow you to select a voice function. Pressing return at any other prompt will leave the remaining parts of the note unchanged and skip ahead to the next note, allowing you to change the lengths without re-entering the remaining values that you don't want to change.

Pressing the '/' and return at any prompt preserves that one value and jumps to the next prompt. For example, if you don't want to change a value, such as length, but do want to change the note or octave, press '/' and it will skip over the values you want to save without your having to retype them.

Note: When your song is absolutely perfect, you must save it on disk or cassette to preserve it for posterity. You can save it on top of the old version or save it under a new name.

The insert option moves all the notes, including the one whose number you entered, forward one position and then prompts for the new note values. After you have entered one new note you are back in the change routine for the next note. If you want to insert several notes at the same position, you will have to return to the note list each time. When you insert at the end of the song, insert mode is considered to be the same as an add mode and continues to add notes to the end of the song.

Delete immediately removes the note whose number you entered and returns you to the list of notes. All the notes to the end of the song

CURRENT SONG: SONG							
#	LEN	NOTE	OCT	#	LEN	NOTE	OCT
0	FUNCTION	3	0	15	12	FUNCTION	0
1	3	0	0	16	10	0	0
2	3	0	0	17	10	0	0
3	3	0	0	18	10	0	0
4	3	0	0	19	10	0	0
5	3	0	0	20	10	0	0
6	3	0	0	21	10	0	0
7	3	0	0	22	10	0	0
8	3	0	0	23	10	0	0
9	3	0	0	24	10	0	0
10	3	0	0	25	10	0	0
11	3	0	0	26	10	0	0
12	3	0	0	27	10	0	0
13	3	0	0	28	10	0	0
14	3	0	0	29	10	0	0
NEXT, PREVIOUS OR #							
?							

move backward, covering up the deleted note.

When you are prompted for a new note, the only notes that will be accepted are 'C ', 'C#', etc., as listed in the data statement (line 17000). A rest is indicated by ']'. The program looks up your note in the note table and converts it into the number that represents that particular note. If it can't find that note, it will tell you 'NOT FOUND' and return to the same note to give you another chance to change it.

Inserting from Scratch

Since we have not loaded any song, there are no notes in memory at this time. The choices available at this point are 'N' for the next page of notes in the song (all the notes won't fit on one page if the song is long), 'P' for the previous page of notes in the song, and '#' for the number of the note to change, insert, or delete. Choose note '1' at this time and press return.

From the next menu, choose 'I' for insert, since you will be constructing a song from the beginning. The prompt 'NEW LENGTH:' will appear and you can enter the values listed below to build a song. Press return after each value is entered.

At the next 'NEW LENGTH' prompt, press the return key. This will return you to the main menu. Press option '2' to play the song.

The first item of any song must be a voice function. After the prompt 'NEW LENGTH:' press the letter F to specify that a function is to be selected. The special prompt for selecting a function will appear

'FUNCTION 0'. The '0' indicates that a voice function must be selected. Valid voice functions are the digits 1 through 8.

Programming Concepts

Using a Menu to Make a Choice

1. Selecting by Number

When you run the PLAYER program, the first display that you see is a list, commonly called a 'menu', that tells you what actions are available. Each item on the menu is selected by pressing the number associated with it. The BASIC program steps required to make the choice are in lines 50 through 80.

```
50 GET A$:IF A$="" THEN 50
60 V=VAL(A$):IF V=0 THEN GOTO 20010
70 IF V<1 OR V>9 THEN 50
80 ON V GOSUB 1000,2000,3000,4000,5000,6000,7000,8000,9000,
  GOTO 40
```

Line 50 waits for a single character from the keyboard.

Line 60 converts the keyboard character in A\$ to its numeric value in variable V. If V is 0, then control goes to line 20010 which reloads the Disk Menu.

Line 70 tests for a value less than 1 or greater than 9 and returns to line 50 to get another choice from the keyboard.

Line 80 uses ON V GOSUB to go to the subroutine whose position matches the number: the first subroutine address for a 1 (subroutine 1000), the second subroutine address for a 2 (subroutine 2000), and so forth for numeric values of 1 through 9.

2. Selecting by First Letter

If the space bar is pressed during the 'PLAY SONG', then another menu is displayed.

CHOICES:

CONTINUE
REPLAY
BACK UP 1 NOTE

YOUR CHOICE, PLEASE:

The first character of each item is displayed in reversed video to indicate that the letter is to be pressed on the keyboard to select that choice. Each letter is serviced by its own IF...THEN statement.

```
9360 GET A$:IF A$="" THEN 9360
9370 IF A$="C" THEN V=V-1:GOSUB 9200:GOTO 9500
9380 IF A$="R" THEN GOSUB 2000:GOTO 9310
9390 IF A$<>"B" THEN 9360
```

If there are only a few choices, as in this example, there is not a lot of code. If there were many choices, then the amount of code to service the letters could be significant.

Numeric Variables

Commodore 64 BASIC handles two kinds of numbers — integer and floating point. An integer number is a number without a decimal (123, 9999, 0, -4387) and so forth. A floating-point number has a decimal point (12.34, 98765.1, 0.0, -435.678) and so forth. Often it does not matter which type of number is being used, but it can make a significant difference in some programs.

Integers

An integer number in the Commodore 64 may be as large as +32767 or -32768. Each integer number that the program stores requires two *bytes* of memory. Most BASICs identify which information is stored in integer form by using a % after the one- or two-character symbol name. Examples in PLAYER are F1%, F2%, T%, W%. In PLAYER, W%, H% and L% are arrays — a large number of related values with a common reference. W%, for example, is the memory reserved to save the note number and duration of each note of the song. To conserve memory, we performed a few tricks. First, let's examine how *not* to program the storage area.

Each note of the song that you play produces two pieces of information that must be saved by Commodore 64 PLAYER:

1. The number of the note.
2. The length of the note.

We could define a two-dimensional array that contains a floating point number for each part (note number and duration) of each note: DIM W(299,1) would reserve space for 600 notes, two floating-point numbers per note. How much memory do you think this would take? Well, 600 times 2 is 1200. Is that the total number of bytes required? No! Each floating-point number requires five (5) bytes of memory. Therefore, it would take 5 * 1200 or 6000 bytes of memory!

One obvious way to save space would be to store each part of each note as integer values instead of floating-point values. Since each integer value requires only two (2) bytes of memory for storage, the total requirement for the 600-byte song would be 2 * 1200 or 2400 bytes. That is better, but we still can improve upon it!

You have to get a bit tricky to squeeze much more out of the song space, but there is nothing wrong with getting tricky when writing programs. In fact, that can be half the fun! To really squeeze the memory in Commodore 64 PLAYER we took advantage of the size of the number that a single integer value might hold. An integer requires two (2) bytes of memory, whether it contains 0, 32335, -32334, or whatever. The two values that we need to keep for each note played are: the note number (0 to 38); and the length (which is limited to 0 to 99 units). If only we could pack both of these individual values into a single integer number for storage and then unpack them when we needed to use them. Well, good news — we can!

The technique to pack the numbers is shown in line 9720.

```
9720 W%(V)=INT(NP:100+D)
```

This equation is not difficult.

W%(V) is the address of the integer where V is the position of the note in the song;

INT is the BASIC function that converts a floating point (five bytes, remember) into an integer number (only two bytes);

NP * 100 multiplies the note number (0 to 38) by 100;

D adds the duration value (0 to 99).

All we have done is multiply one of the parts that we need to save by enough to make sure it does not overlap the other part. This insures that we will be able to unpack the separate parts later. The unpacking is slightly more difficult than the packing, but conceptually it is simple. All we need to do is reverse the packing process. This is accomplished in the following lines:

```

9200 X=W%(V)
9210 IF X> 4999 THEN R=X-5000:RETURN
9220 R=10
9230 NP=INT(X/100)
9240 D=INT(X-NP*100)
9250 RETURN

```

Line 9200 copies the packed value of the note into X.

Line 9210 tests for a special value that is used to represent voice changes. A voice change is indicated by the voice number plus 5000. If the stored number is greater than 4999, then it is converted to the voice number by subtracting 5000.

Line 9220 sets a dummy value for the 'new' voice to indicate that the old voice should be maintained.

Line 9230 calculates the note number by dividing the stored number by 100.

Line 9240 restores the duration by subtracting the note number component from the stored number.

Line 9250 returns to the calling program.

The above lines of program have taken the single integer value and converted it back into two parts. The savings of this method result in a 600-note song requiring only 600 integer numbers to store it, at two bytes per number, for a total storage of 1200 bytes. Quite a reduction from the original 6000 bytes!

The Program

The Player program contains ten major functions, which are selected from a menu, three minor functions used during the playing of a song from the keyboard, and some support subroutines.

The main program (lines 10 - 80) calls subroutines to initialize the program, turn off the sound generators, and print the menu on the display. It waits for a character in the range of the menu (0 to 9), then goes to the appropriate subroutine to service the selected function.

The PLAY SONG subroutine (lines 1000 - 1150) calls a subroutine to print the music keyboard, gets input from the keyboard, calls a subroutine to pack information about the current note and store it in memory, and then, depending on which key you press, will do one of the following things:

Space goes to the 'Continue, Replay, Back Up' subroutine;

Carriage Return terminates the song by turning off the sound, putting a 0 in the next song location and returning to the main menu;

'f1' through 'f8' select a voice by calling subroutine 7800 that calculates the wave form, attack, decay, sustain, release and volume for the selected voice. It then bumps the song memory pointer and saves the voice number plus 5000 in the song memory. It uses subroutine 9900 to activate the voice with all of its parameters, and then continues in the PLAY SONG routine.

An *undefined key* is converted to a 'musical rest'. A *defined key* (i.e., a 'music key') is converted to its high and low sound values via a table lookup in line 1130. The note is saved in the song table and the routine is continued.

The REPLAY SONG subroutine (lines 2000 - 2090) prints the message 'PLAYING' and the song title, if there is one. If there is no song in memory, it prints the message 'NO SONG IN MEMORY', goes to a subroutine that produces a delay to enable the message to be read, and then returns to the main program.

If there is a song in memory, REPLAY calls subroutine 9200 to unpack the note number and duration for the next note from the song table. If the value of R returned by the subroutine equals 10, then it is a normal note. If the note has the value 0, this indicates the end of the song. The sound is turned off and control is returned to the main menu. If the note is not 0, then subroutine 9800 is used to output the note to the sound generator for the specified duration. The sound generator is then turned off and the song pointer incremented to the next note. If the maximum song length has been exceeded, then a return is made to the main program; otherwise the keyboard is checked and, if any key is pressed, a return is made to the main program.

If the value of R does not equal 10, then a voice change has been read. Subroutine 7800 is called to lookup the voice values and subroutine 9900 to output them to control the voice.

If no key is pressed, the next note of the song is unpacked and tested. If it is not the 'end of song' indicator (a zero value), the program continues playing the song. At the end of the song, REPLAY uses a subroutine to generate a brief delay and then goes to the main program.

The SAVE SONG subroutine (lines 3000 - 3070) uses a subroutine to print the message 'WHAT IS THE NAME OF THE SONG?' and

inputs a song name. It opens the current storage device for saving information, first outputting the voice table information and then the song information one note at a time. When it detects the zero note that signals the end of the song it closes the device, prints the message 'SAVED', and exits through the delay routine.

The LOAD SONG subroutine (lines 4000 - 4090) uses a subroutine to print the message 'WHAT IS THE NAME OF THE SONG?' and inputs a song name. It opens the current storage device for loading information and inputs the song information one note at a time. When it detects an empty note it closes the current device, prints the message 'LOADED', and exits through the delay routine.

The CHANGE TEMPO subroutine (lines 5000 - 5020) prints the current value of the tempo variable, then requests and inputs a new value. If the value is less than 1 or greater than 99, it is ignored. When a valid value is input, the routine returns to the main program.

The SONG EDITOR routines start at line 6000. The program prints the headings on the screen and keeps track of the song note number (V) and the position on the screen (J). V is converted to a string of three characters so that the numbers will be right-justified.

Subroutine 6900 is called to fetch the next set of values for the song memory array. It, in turn, calls 9200 to split the value into either its function value or into the note and duration. A valid note is converted into an octave and a note within the octave. D is the duration of the note and it is also converted to a string, since strings are easier to print in columns than are numbers. If J is less than 15, the numbers are printed in the first column; otherwise they are printed in the second column, up to thirty in all. If the end of the song is reached, an 'END' is printed on the screen.

At line 6150 the prompt 'Next', 'Previous', or '#' is printed, and the input checked to see if the program should print the next page, the previous page, or go to the edit routine when a number is encountered. A RETURN with no other input will return to the main PLAYER menu.

A number input leads to a request for a choice of 'Change', 'Delete', or 'Insert'. The change routine (line 6210) checks for input on each part of the note — length, value and octave or voice function. A return at the beginning returns to the page-print routine. A return elsewhere in the input routine skips ahead to the next note. A '/' input at any time preserves the current value and skips ahead to the next part of the note. The routine at line 6430 looks for a match of the input note

with allowable values and then converts the note to the number representation. The value for the song table is calculated using the note pointer NP, the duration D, and the octave O. It then returns to get the next note.

The previous page routine (line 6530) resets the J column counter and sets V back 31 counts and returns to printing a page. The next page routine (line 6540) resets the J column counter and sets V to the place where it left off.

The insert routine at line 6570 moves ahead all the notes from the number input to the end of the song and returns to the change routine. The delete routine at line 6610 moves back all the notes from the number input to the end of the song, effectively erasing that note.

DISPLAY/CHANGE FUNCTIONS (lines 7000 - 7850) provides for control of the voice table from the keyboard. Lines 7000 through 7120 convert the voice table values into the display. Lines 7130 through 7420 interact with the user to permit the voices to be redefined. First, the voice number to be changed is input, a number between 1 and 8. Then the various parts of the voice are individually listed and opened for modification. If a RETURN is pressed, then the current value of the parameter is left unchanged. Each change is checked for validity. If not a valid value, the question is asked again until either a good valued or a RETURN is input. Lines 7400 to 7420 pack the voice information and store it in the voice table, then return to the start to display the voice table and permit further modification.

Lines 7800 through 7850 unpack the voice table for voice I, creating values for the pulse width PW%, wave form WF%, volume VL%, attack AF%, decay DF%, sustain SF% and release RF%.

DEVICE SELECT takes place in lines 8000 through 8070. Line 8010 displays the current device; line 8020 requests a D for disk or T for tape or RETURN to keep the current device. If any other key in pressed, the question is repeated until a valid response is input.

The EXIT TO BASIC routine (line 9000) clears the screen and executes an END to return to BASIC.

The EXIT TO DISK MENU routine (lines 20000 - 20020) use a sneaky trick to load and execute the menu program from disk. Since Commodore 64 BASIC does not have any simple mechanism to load in another BASIC program, we had to invent a method. Line 20010 uses a PRINT command to clear the display and write the commands that a user would have to type in to load a program from the disk:

```
LOAD "MENU",8  
RUN
```

The Commodore 64 can execute instructions directly from the screen. All that is needed now is some way to pretend that the HOME key was pressed to set the cursor on the top line with the 'LOAD' message, and then a RETURN was pressed to cause that line to be executed, and finally, after the file had been loaded, another RETURN was given to execute the 'RUN' command. Well, this is exactly what line 20020 does. The Commodore 64's keyboard input buffer starts at location 631, where we POKE the code for the HOME key, 19. Then we POKE the value for a RETURN into the next two buffer locations. Now we have the HOME and two RETURN's that we require. All that is left is to tell the Commodore 64 that there are three (3) characters in its keyboard input buffer. We do this by POKEing the keyboard buffer counter at 198 with 3. The END causes us to leave the PLAYER program and return to the BASIC Editor which looks for a character in the keyboard buffer, finds the HOME that we put there, executes it, and so forth.

The PLAY SONG routine (lines 9300 - 9500) provides three additional commands for use while playing a song. When called by pressing the space bar, it first uses subroutine 9500 to turn off all sound generators and subroutine 9200 to unpack the current note. It prints its own menu and waits for a keyboard selection. On receipt of a 'C' it backs up the song pointer to the current note, unpacks the note, and turns off the sound generators before returning to the PLAY SONG routine.

The letter 'R' calls the REPLAY SONG subroutine, which plays the current song from the beginning to the current note, and then waits for another menu selection.

The letter 'B' causes the song note pointer to back up one position, unless it is already at the start of the song, and then go to the subroutines to unpack the note's values, output the note for the correct duration, turn off the sound generators when the note is done, and wait for another menu selection.

The support subroutines (lines 9100 - 11160) provide support for the main program and major subroutines.

Lines 9100 to 9110 ask "WHAT IS THE NAME OF THE SONG: " and accept the users input as variable B\$ which is used by the Load and Save routines.

Line 9200 unpacks the stored note information. If the value is greater than 4999, then it is a voice change. 5000 is subtracted from the stored value to generate the voice number. If the value is

less than 4999, then it is divided into the note pointer 'NP' and the duration of the note 'D'.

Line 9500 turns off the sound.

Line 9600 provides a several-second delay to permit you to view messages.

Lines 9700 to 9720 pack the current note and duration into a single number and save it in the song table.

Line 9100 prints 'WHAT IS THE NAME OF THE SONG?' and accepts a name from the keyboard.

Line 10000 prints the main menu.

Line 11000 prints the keyboard display.

Line 12000 performs a series of initialization functions. It defines CR\$ to equal the carriage return character; it dimensions all of the arrays; it zeros all of the sound generator locations and defines the important sound generator addresses; it restores the BASIC DATA pointer and then READs the values for the High (H%) and Low (L%) sound value arrays (lines 13000 - 13060), the data for the note pointer (NP%) array (lines 14000 - 14040), the initial data for the voice function table (lines 15000 - 15070), the letter names for the Wave form Table (WT\$) (line 16000), digits for the function code table (FK%) (line 16010) and the note names (N\$) (line 17000).

Line 14000: This data table has the pointers for the keyboard playing routine. Since the keyboard is not in note order and the note table values are in note order, it is necessary to convert from the keyboard code to the position in the note table. For example, 'Z', which is a 'C', has the keycode 90. This number is called the ASCII value and is a standard form of encoding the keys for a computer. Line 1120 subtracts 42 from 90, resulting in 48 as the keyboard position. (The first 42 keycodes are unused, and so we throw them away.) If you look at the table of pointers (starting at line 14000 and counting from zero), you will see that the 48th entry is '0'. Therefore, the note pointer for Z is 0. If you look at the note table data (starting at line 13000), you will see that the zeroth (first) entry pair in this table is 16 for the High value and 195 for the Low value. This is the value to be POKEd to make the sound 'C' in the fourth octave. This value was obtained from the Music Note Values table in the *Commodore 64 Programmer's Reference Guide*, page 384.

```

1 REM    PLAYER
10 POKE 53281,1
20 DV$="D":DN$="DISK":S=40:L=260
30 GOSUB 12000
40 GOSUB 10000
50 GET A$:IF A$="" THEN 50
60 V=VAL(A$):IF V=0 THEN GOTO 20010
70 IF V<1 OR V>9 THEN 50
80 ON V GOSUB 1000,2000,3000,4000,5000,6000,7000,8000,9000:
    GOTO 40
1000 V=-1:F=1:FR=10
1010 GOSUB 11000
1020 D=0
1030 GET A$:D=D+1:IF A$="" THEN 1030
1040 IF F=1 THEN 1060
1050 GOSUB 9700:POKE CR,CW
1060 IF A$=" " THEN GOSUB 9300:GOTO 1010
1070 IF A$=CR$ THEN POKE CR,CW:W%(V+1)=0:RETURN
1080 TP=ASC(A$):IF TP<133 OR TP>141 THEN GOTO 1110
1090 T%=TP-133:F%=FK%(T%):I=F%:GOSUB 7800:V=V+1:W%(V)=F%+5000
1100 POKE VL,VL%:GOSUB 9900(I,1):FR=F%:F=1:GOTO 1020
1110 IF FR=10 GOTO 1010
1120 NX=TP-42:IF NX>50 THEN NX=3
1130 NP=NP%(NX):POKE HI,H%(NP):POKE LO,L%(NP):
    POKE CR,CW+1:F=0
1140 V=V+1:IF V=L THEN W%(V)=0:RETURN
1150 GOTO 1030
2000 PRINT "{CLEAR}";TAB(IN);"{RED}PLAYING {BLACK} ";B$:V=0
2010 IF W%(V)=0 THEN PRINT "{DOWN,BLUE}
    NO SONG IN MEMORY{BLACK}":GOTO 9600
2020 GOSUB 9200:IF R=10 GOTO 2050
2030 I=R:GOSUB 7800:GOSUB 9900:POKE VL,VL%:V=V+1:
    IF V>L THEN RETURN
2040 GOTO 2080
2050 IF X=0 THEN 9500
2060 GOSUB 9800
2070 POKE CR,CW:V=V+1:IF V>L THEN RETURN
2080 GET A$:IF A$<>" " THEN POKE CR,CW:RETURN
2090 GOTO 2020
3000 GOSUB 9100:IF DV$="T" THEN 3020
3010 OPEN 1,8,2,"@0:"+"B$+",S,W":GOTO 3030
3020 OPEN 1,1,1,B$

```

```

3030 PRINT# 1,B$
3040 FOR I=0 TO 7:FOR J=0 TO 2:PRINT# 1,F%(I,J):NEXT:NEXT
3050 PRINT# 1,W%(V)
3060 IF W%(V)=0 THEN CLOSE 1:PRINT "{DOWN}SAVED ";B$:
      GOTO 9600
3070 V=V+1:GOTO 3050
4000 GOSUB 9100:IF DV$="T" THEN 4020
4010 OPEN 1,8,2,"0":"+B$+",S,R":GOTO 4030
4020 OPEN 1,1,0,B$
4030 INPUT# 1,B$
4040 FOR I=0 TO 7:FOR J=0 TO 2:INPUT# 1,F%(I,J):NEXT:NEXT
4050 INPUT# 1,W%:IF W%=0 THEN 4090
4060 W%(V)=W%:GOTO 4080
4070 INPUT# 1,W%(V)
4080 IF W%(V)<>0 THEN V=V+1:GOTO 4070
4090 CLOSE 1:PRINT "{DOWN}LOADED ";B$:GOTO 9600
5000 PRINT TAB(IN);"{BLACK}CURRENT TEMPO = {RED}";S:
5010 PRINT TAB(IN);"{BLACK}NEW TEMPO = {RED}";:INPUT S:
      PRINT "{BLACK}"
5015 IF S<1 OR S>99 THEN 5010
5020 RETURN
6000 PRINT "{CLEAR,RED}CURRENT SONG: {BLACK}";B$
6010 V=-1
6020 J=0:PRINT "{DOWN,RED} # LEN NOTE OCT #
      LEN NOTE OCT{BLACK,DOWN}"
6030 V=V+1:J=J+1:V$=STR$(V):IF LEN(V$)<3 THEN V$=" "+V$
6040 IF LEN(V$)>3 THEN V$=RIGHT$(V$,3)
6050 GOSUB 6900:IF X=0 THEN PRINT "END{HOME,DOWN19}":GOTO 6150
6060 IF R=10 THEN 6080
6070 PRINT V$;"{RED} FUNCTION ";R+1;"{BLACK}":GOTO 6100
6080 D$=STR$(D):IF D<10 THEN D$=" "+D$
6090 PRINT V$;" ";D$;" ";N$;" ";O$
6100 IF J<15 THEN 6030
6110 IF J=15 THEN PRINT "{HOME,DOWN4}";
6120 PRINT TAB(20);
6130 IF J<30 THEN 6030
6140 PRINT
6150 PRINT "{DOWN,BLUE,RVS}N{RVSOFF}EXT,{RVS}P
      {RVSOFF}REVIOUS OR {RVS}#{RVSOFF,BLACK}"
6160 A$="":INPUT A$:TP=V:V=VAL(A$):
      IF V=0 AND A$<>"0" THEN 6530

```



```

6170 PRINT "{CLEAR,BLUE,RVS}C{RVSOFF}HANGE, {RVS}D
      {RVSOFF}ELETE OR {RVS}I{RVSOFF}NSERT?{BLACK}"
6180 GOSUB 6640:IF A$="C" THEN 6210
6190 IF A$=CR$ THEN 6500
6195 IF A$=CHR$(13) THEN 6500
6200 GOTO 6560
6210 GOSUB 6900
6220 IF R=10 THEN 6280
6230 PRINT "{DOWN,RED}  FUNCTION ";R+1;"{BLACK}"
6240 A$="":INPUT "{DOWN,BLUE}NEW FUNCTION: {BLACK}";A$:
      IF A$="" THEN 6490
6250 IF A$="/" THEN 6500
6260 R=VAL(A$):IF R<1 OR R>8 THEN 6240
6270 W%(V)=R+4999:GOTO 6490
6280 PRINT "{DOWN,RED}  #  LEN NOTE OCT{BLACK}"
6290 IF W%(V)=0 THEN 6330
6300 V$=STR$(V):IF LEN(V$)<3 THEN V$=" "+V$
6310 D$=STR$(D):IF D<10 THEN D$=" "+D$
6320 PRINT V$;" ";D$;" ";N$;" ";O$
6330 A$="":INPUT "{DOWN,BLUE}NEW LENGTH: {BLACK}";A$:
      IF A$="" THEN 6360
6340 IF A$="/" THEN 6500
6350 D$=A$:IF VAL(D$)>99 THEN D$="99"
6360 A$="":INPUT "{BLUE}NEW NOTE: {BLACK}";A$:
      IF A$="" THEN 6390
6370 IF A$="/" THEN 6430
6380 N$=A$:IF LEN(N$)=1 THEN N$=N$+" "
6390 A$="":INPUT "{BLUE}NEW OCTAVE: {BLACK}";A$:
      IF A$="" THEN 6430
6400 IF A$="/" THEN 6430
6410 IF VAL(A$)>3 THEN A$="3"
6420 O$=A$
6430 NP=-1:FOR I=0 TO 12:IF N$=N$(I) THEN NP=I:I=12
6440 NEXT
6450 IF NP=-1 THEN PRINT "{GREEN}NOT FOUND{BLACK}":GOTO 6210
6460 IF NP=12 THEN NP=38
6470 D=VAL(D$)
6480 NP=NP+VAL(O$)*12:W%(V)=INT(NP*100+D)
6490 V=V+1:GOTO 6210
6500 V=TP-J
6510 IF V<=0 THEN V=-1
6520 PRINT "{CLEAR}":GOTO 6020

```

```

6530 IF A$="P" THEN V=TP-(J+30):GOTO 6510
6540 IF A$="N" THEN PRINT "{CLEAR}":J=0:V=TP:GOTO 6020
6550 J=0:RETURN
6560 IF A$="D" THEN 6610
6570 FOR I=1 TO L:IF W%(I)=0 THEN A$=STR$(I):I=L
6580 NEXT:FOR I=VAL(A$) TO V STEP -1
6590 W%(I+1)=W%(I):NEXT
6600 GOTO 6330
6610 W%(V)=W%(V+1):V=V+1
6620 IF W%(V)=0 THEN J=V:GOTO 6500
6630 GOTO 6610
6640 GET A$:IF A$="" THEN 6640
6650 RETURN
6900 GOSUB 9200
6910 N=A%(NP):IF NP=38 THEN NP=12:O=0:GOTO 6930
6920 O=INT(NP/12):NP=NP-O*12
6930 N$=N$(NP):O$=RIGHT$(STR$(O),1)
6940 RETURN
7000 PRINT "{CLEAR,BLACK,DOWN}";TAB(IN);"FUNCTION TABLE"
7010 PRINT "{DOWN2,BLUE}F# WF AT DC ST RL VOL PULSE{RED}"
7020 FOR I=0 TO 7: PRINT
7030 PRINT "F{LEFT}";I+1;:GOSUB 7800
7040 PRINT " ";WF$;" ";
7050 IF AF%<10 THEN PRINT " ";
7060 PRINT AF%;:IF DF%<10 THEN PRINT " ";
7070 PRINT DF%;:IF SF%<10 THEN PRINT " ";
7080 PRINT SF%;:IF RF%<10 THEN PRINT " ";
7090 PRINT RF%;:IF VL%<10 THEN PRINT " ";
7100 PRINT " ";VL%;:IF PW%<10 THEN PRINT " ";
7110 IF WF$="P" THEN PRINT " ";PW%;
7120 NEXT
7130 PRINT:PRINT "{DOWN,BLACK}F# TO CHANGE [1-8] OR RETURN: ";
7140 N$="":INPUT N$:IF N$="" THEN RETURN
7150 T=VAL(N$):IF T<1 OR T>8 THEN 7140
7160 I=T-1:GOSUB 7800
7170 PRINT "VOICE [T S P N]: ";WF$;
7180 N$="":INPUT N$:IF N$="" THEN 7210
7190 FOR J=0 TO 3:IF N$=WT$(J) THEN WF%=J:GOTO 7210
7200 NEXT:GOTO 7180
7210 PRINT "ATTACK [0-15]: ";AF%;" ";
7220 N$="":INPUT N$:IF N$="" THEN A%=AF%:GOTO 7240
7230 A%=VAL(N$):IF A%<0 OR A%>15 THEN 7220

```

```

7240 PRINT "DECAY [0-15]: ";DF%;" ";
7250 N$="":INPUT N$:IF N$="" THEN D%=DF%:GOTO 7270
7260 D%=VAL(N$):IF D%< 0 OR D%> 15 THEN 7250
7270 PRINT "SUSTAIN [0-15]: ";SF%;" ";
7280 N$="":INPUT N$:IF N$="" THEN S%=SF%:GOTO 7300
7290 S%=VAL(N$):IF S%< 0 OR S%> 15 THEN 7280
7300 PRINT "RELEASE [0-15]: ";RF%;" ";
7310 N$="":INPUT N$:IF N$="" THEN R%=RF%:GOTO 7330
7320 R%=VAL(N$):IF R%< 0 OR R%> 15 THEN 7310
7330 PRINT "VOLUME [0 - 15]: ";VL%;" ";
7340 N$="":INPUT N$:IF N$="" THEN V%=VL%:GOTO 7360
7350 V%=VAL(N$):IF V%< 0 OR V%> 15 THEN 7340
7360 P%=PW%:IF WF%< > 2 THEN 7400
7370 PRINT "PULSE ON PERCENT [1-99]: ";PW%;" ";
7380 N$="":INPUT N$:IF N$="" THEN P%=PW%:GOTO 7400
7390 P%=VAL(N$):IF P%< 1 OR P%> 99 THEN 7380
7400 F%(I,0)=P%*256+WF%*16+V%:F%(I,1)=A%*16+D%
7410 F%(I,2)=S%*16+R%
7420 GOTO 7000
7800 PW%=F%(I,0)/256
7810 WF%=(F%(I,0) AND 255)/16:WF$=WT$(WF%):
      CW=2{UP ARROW}(WF%+4)
7820 VL%=F%(I,0) AND 15
7830 AF%=F%(I,1)/16:DF%=F%(I,1) AND 15
7840 SF%=F%(I,2)/16:RF%=F%(I,2) AND 15
7850 RETURN
8000 DX$=DV$
8010 PRINT "{CLEAR,BLACK}CURRENT SAVE/LOAD
      DEVICE IS {RED}";DN$
8020 PRINT "{DOWN,BLACK}SELECT DISK (D),
      TAPE (T) OR RETURN"
8030 DV$="":INPUT "TO KEEP CURRENT DEVICE: ";DV$
8040 IF DV$="" THEN DV$=DX$:RETURN
8050 IF DV$="D" THEN DN$="DISK":RETURN
8060 IF DV$="T" THEN DN$="TAPE":RETURN
8070 GOTO 8010
9000 PRINT "{CLEAR}":END
9100 PRINT "{CLEAR}";"{BLACK}WHAT IS THE NAME OF THE SONG: "
9110 V=0:INPUT "{DOWN}      ";B$:RETURN
9200 X=W%(V)
9210 IF X> 4999 THEN R=X-5000:RETURN
9220 R=10

```

```

9230 NP=INT(X/100)
9240 D=INT(X-NP*100)
9250 RETURN
9300 GOSUB 9500:W%(V+1)=0:GOSUB 9200
9310 PRINT "{CLEAR}";TAB(IN);"{BLACK}CHOICES:"
9320 PRINT TAB(IN);"{RED,RVS}C{RVSOFF}ONTINUE"
9330 PRINT TAB(IN);"{RVS}R{RVSOFF}EPLAY"
9340 PRINT TAB(IN);"{RVS}B{RVSOFF}ACK UP 1 NOTE"
9350 PRINT:PRINT TAB(IN);"{BLACK}YOUR CHOICE, PLEASE: ";
9360 GET A$:IF A$="" THEN 9360
9370 IF A$="C" THEN V=V-1:GOSUB 9200:GOTO 9500
9380 IF A$="R" THEN GOSUB 2000:GOTO 9310
9390 IF A$<>"B" THEN 9360
9400 IF V=1 GOTO 9360
9410 V=V-1:GOSUB 9200:GOSUB 9800
9420 GOSUB 9500:W%(V)=0:GOTO 9360
9500 POKE CR,CW:RETURN
9600 FOR I=1 TO 900:NEXT:RETURN
9700 D=D/4+10:IF D>99 THEN D=99
9705 IF D<0 THEN D=1
9710 IF D<0 THEN D=1
9720 W%(V)=INT(NP*100/D)
9730 RETURN
9800 POKE HI,H%(NP):POKE LO,L%(NP):POKE CR,CW+1
9810 SD=((S*D)/4)-30:IF SD<0 THEN SD=1
9820 FOR I=1 TO SD:NEXT:RETURN
9900 POKE AD,F%(I,1):POKE SR,F%(I,2)
9910 PX%=PW%*41:POKE PH,INT(PX%/256):POKE PL,(PX% AND 255):
RETURN
10000 PRINT"{CLEAR,BLACK}":V=0:IN=6
10010 PRINT TAB(IN);"    PLAYER 64":PRINT
10020 PRINT TAB(IN);"{BLUE,DOWN2,RVS}1{RVSOFF} PLAY SONG"
10030 PRINT TAB(IN);"{RVS}2{RVSOFF} REPLAY SONG"
10040 PRINT TAB(IN);"{RVS}3{RVSOFF} SAVE SONG"
10050 PRINT TAB(IN);"{RVS}4{RVSOFF} LOAD SONG"
10060 PRINT TAB(IN);"{RVS}5{RVSOFF} CHANGE TEMPO {RED}";
S;"{BLUE}"
10070 PRINT TAB(IN);"{RVS}6{RVSOFF} EDIT SONG"
10080 PRINT TAB(IN);"{RVS}7{RVSOFF} DISPLAY/CHANGE FUNCTIONS"
10090 PRINT TAB(IN);"{RVS}8{RVSOFF} CHANGE SAVE/LOAD
DEVICE {RED}";DN$;"{BLUE}"
10100 PRINT TAB(IN);"{RVS}9{RVSOFF} EXIT TO BASIC"

```

```

10110 PRINT TAB(IN); "{RVS}Ø{RVSOFF} EXIT TO DISK MENU"
10120 PRINT TAB(IN); "{DOWN,BLACK}YOUR CHOICE, PLEASE:
      {DOWN2} ":RETURN
11000 IN=10:PRINT "{CLEAR,BLACK,DOWN}";TAB(IN);
      " PLAY WHEN READY"
11010 PRINT TAB(IN-1); "{BLUE,DOWN2}SELECT VOICE F1 - F8"
11020 PRINT TAB(IN); "{RED,DOWN}SPACE BAR TO PAUSE":
      PRINT TAB(IN); "RETURN TO END SONG"
11030 PRINT TAB(IN); "A, F, OR K FOR REST":IN=6:PRINT "{BLACK}"
11040 PRINT TAB(IN); "{=A,=R,↑C,=R,↑C,=R,↑C,=R3,↑C,=R,↑C,
      =R3,↑C,=R,↑C,=R,↑C,=R3,↑C,=S}"
11050 PRINT TAB(IN); "{↑-2,RVS} {RVSOFF,↑-,RVS} {RVSOFF,↑-,
      RVS} {RVSOFF,↑-3,RVS} {RVSOFF,↑-,RVS} {RVSOFF,↑-3,
      RVS} {RVSOFF,↑-,RVS} {RVSOFF,↑-,RVS} {RVSOFF,↑-3,
      RVS} {RVSOFF,↑-}"
11060 PRINT TAB(IN); "{↑-2,RVS}2{RVSOFF,↑-,RVS}3{RVSOFF,↑-,
      RVS}4{RVSOFF,↑-3,RVS}6{RVSOFF,↑-,RVS}7{RVSOFF,↑-3,
      RVS}9{RVSOFF,↑-,RVS}Ø{RVSOFF,↑-,RVS}+{RVSOFF,↑-3,
      RVS,POUND,RVSOFF,↑-}"
11070 PRINT TAB(IN); "{BLACK,RVSOFF,↑-,=Z,↑*,=E,↑*,=E,↑*,=X,
      ↑-,=Z,↑*,=E,↑*,=X,↑-=Z,↑*,=E,↑*,=E,↑*,=X,↑-,=Z,↑C,=X}"
11080 PRINT TAB(IN); "{BLACK,↑-,BLUE}F{BLACK,↑-,BLUE}G
      {BLACK,↑-,BLUE}A{BLACK,↑-,BLUE}B{BLACK,↑-,BLUE}C
      {BLACK,↑-,BLUE}D{BLACK,↑-,BLUE}E{BLACK,↑-,BLUE}F
      {BLACK,↑-,BLUE}G{BLACK,↑-,BLUE}A{BLACK,↑-,BLUE}B
      {BLACK,↑-,BLUE}C{BLACK,↑-}"
11090 PRINT TAB(IN); "{BLACK,↑-}Q{↑-}W{↑-}E{↑-}R{↑-}T{↑-}Y{↑-}
      U{↑-}I{↑-}O{↑-}P{↑-}@{↑-}*{↑-}"
11100 PRINT
11100 PRINT TAB(IN); "{BLACK,RVSOFF,=Z,↑*,=E,↑*,=E,↑*,=E,↑*,=E,
      ↑*,=E,↑*,=E,↑*,=E,↑*,=E,↑*,=E,↑*,=E,↑C,=X}":IN=7
11110 PRINT TAB(IN); "{BLACK} {↑-2,RVS} {RVSOFF,↑-,RVS}
      {RVSOFF,↑-3,RVS} {RVSOFF ↑-,RVS} {RVSOFF,↑-,RVS}
      {RVSOFF,↑-3,RVS} {RVSOFF,↑-,RVS} {RVSOFF,↑-2}"
11120 PRINT TAB(IN); "{BLACK} {↑-2,RVS}S{RVSOFF,↑-,RVS}D
      {RVSOFF,↑-3,RVS}G{RVSOFF ↑-,RVS}H{RVSOFF,↑-,RVS}J
      {RVSOFF,↑-3,RVS}L{RVSOFF,↑-,RVS}:{RVSOFF,↑-2}"
11130 PRINT TAB(IN); "{BLACK} {↑-,=Z,↑*,=E,↑*,=X,↑-,=Z,↑*,=E,
      ↑*,=E,↑*,=X,↑-,=Z,↑*,=E,↑*,=X,↑-}"
11140 PRINT TAB(IN); "{BLACK} {↑-,BLUE}C{BLACK,↑-,BLUE}D
      {BLACK,↑-,BLUE}E{BLACK,↑-,BLUE}F{BLACK,↑-,BLUE}G
      {BLACK,↑-,BLUE}A{BLACK,↑-,BLUE}B{BLACK,↑-,BLUE}C
      {BLACK,↑-,BLUE}D{BLACK,↑-,BLUE}E{BLACK,↑-}"

```

```

11150 PRINT TAB(IN);"{BLACK} {↑-}Z{↑-}X{↑-}C{↑-}V{↑-}B
      {↑-}N{↑-}M{↑-},{↑-}.{↑-}/{↑-}"
11160 PRINT TAB(IN);"{BLACK,RVSOFF} {=Z,↑*,=E,↑*,=E,↑*,
      =E,↑*,=E,↑*,=E,↑*,=E,↑*,=E,↑*,=E,↑*,=X,BLUE}":
      RETURN
12000 CR$=CHR$(13)
12010 DIM W%(L),H%(38),L%(38),NP%(50),F%(8,3),WT$(4),FK%(8)
12020 DIM N$(12),A%(38)
12030 LO=54272:FOR I=0 TO 24:POKE LO+I,0:NEXT
12040 HI=LO+1:PL=LO+2:PH=LO+3:CR=LO+4:AD=LO+5:SR=LO+6:VL=54296
12050 RESTORE
12060 FOR I=0 TO 38:READ H%(I):READ L%(I):NEXT
12070 FOR I=0 TO 50:READ NP%(I):NEXT
12080 FOR I=0 TO 7
12090 READ P%,V%,L%,A%,D%,S%,R%
12100 F%(I,0)=P%*256+V%*16+L%
12110 F%(I,1)=A%*16+D%:F%(I,2)=S%*16+R%
12120 NEXT
12130 FOR I=0 TO 3:READ WT$(I):NEXT
12140 FOR I=0 TO 7:READ FK%(I):NEXT
12150 FOR I=0 TO 12:READ N$(I):NEXT
12160 FOR I=0 TO 38:READ A%(I):NEXT
12170 RETURN
13000 DATA 16,195,17,195,18,209,19,239,21,31,22,96
13010 DATA 23,181,25,30,26,156,28,49,29,223,31,165
13020 DATA 33,135,35,134,37,162,39,223,42,62,44,193
13030 DATA 47,107,50,60,53,57,56,99,59,190,63,75
13040 DATA 67,15,71,12,75,69,79,191,84,125,89,131
13050 DATA 94,214,100,121,106,115,112,199,119,124,126,151
13060 DATA 134,30,142,24,0,0
14000 DATA 36,34,12,38,14,16,32,38,18,20
14010 DATA 22,38,25,27,38,30,15,38,38,38
14020 DATA 14,16,35,38,7,4,3,21,38,6,8,29
14030 DATA 10,38,13,11,9,31,33,17,23,1,24
14040 DATA 28,5,19,2,26,0,38,37
15000 DATA 6,2,15,0,9,0,0
15010 DATA 0,0,15,6,0,0,0
15020 DATA 0,1,15,0,9,0,0
15030 DATA 0,0,15,0,9,0,0
15040 DATA 0,0,15,0,0,15,0
15050 DATA 0,0,15,6,6,0,0
15060 DATA 0,1,15,6,0,0,0

```

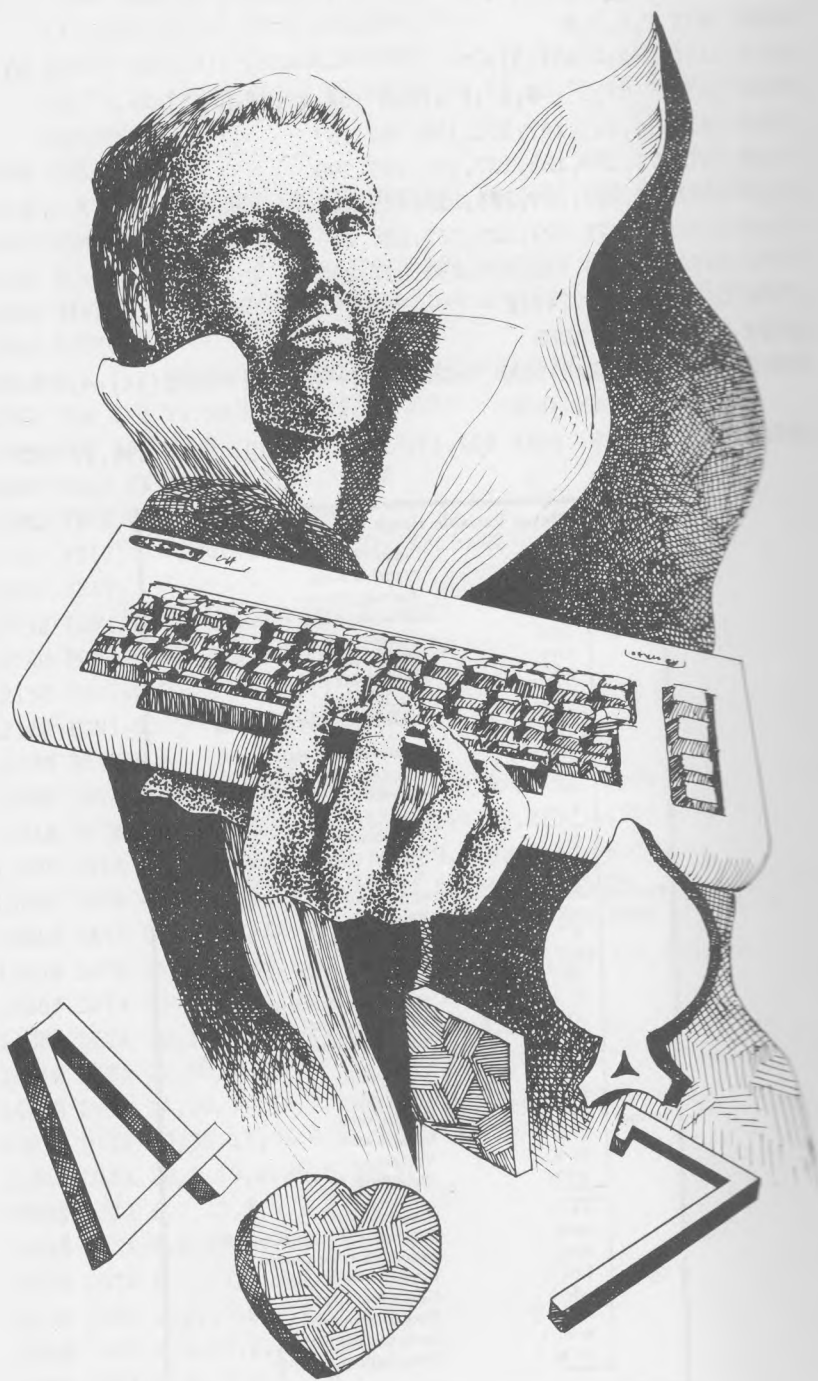


```

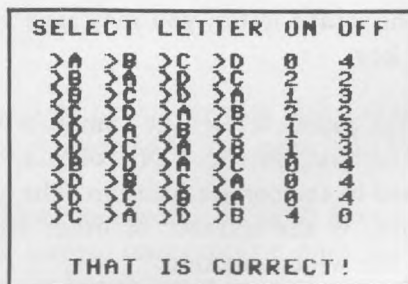
15070 DATA 0,3,15,4,4,4,4
16000 DATA T,S,P,N
16010 DATA 0,2,4,6,1,3,5,7
17000 DATAC ,C#,D ,D#,E ,F ,F#,G ,G#,A ,A#,B ,[]
17010 DATA135,143,147,151,159,163,167
17020 DATA175,179,183,187,191,195,199
17030 DATA201,203,207,209,212,215,217
17040 DATA219,221,223,225,227,228,229
17050 DATA231,232,233,235,236,237,238
17060 DATA239,240,241,0
20000 REM MENU LOADER
20010 PRINT "{CLEAR}LOAD "+CHR$(34)+"0:MENU"+CHR$(34)+" ,8":
      PRINT "{DOWN4}RUN"
20020 POKE 631,19: POKE 632,13: POKE 633,13: POKE 198,3: END

```

Player Variable Usage		
String	A\$	Input variable
	B\$	Name of song
	CR\$	Carriage return
	D\$	Duration string
	DNS	Device name
	DV\$	Device code
	N\$	Note string
	O\$	Octave string
Floating Point	V\$	Note number in song string
	CR	Current sound register
	CW	Current wave form
	D	Duration of note
	F	Flag
	I	Loop counter
	J	Vertical screen position
	L	Number of notes in array
	NP	Note code
	NX	Key code
	R	Voice number
	S	Tempo
	TP	Keyboard code
	V	Number of current notes
Integer	VL	Volume register
	X,Y	Temporary storage
	AF%	Attack value
	DF%	Decay value
	RF%	Release value
	SF%	Sustain value
Array	VL%	Volume
	WF%	Wave form
	F%()	Function table
	FK%()	Function key translation table
	H%()	Note high value storage
	L%()	Note low value storage
	.N\$()	Note name array
	NP%()	Note code storage
	W%()	Array to store song
	WT\$()	Wave form name table



A Simple Guessing Game



MASTER is a simple guessing game for one or two players. The commercial version of this game involves colored pegs. One player constructs a pattern of four colored pegs behind a screen, and it is up to the other player to guess the concealed pattern. The first player provides the second player with clues, telling him how many pegs have been guessed in the right position, and how many pegs are the right color but in the wrong position. The second player continues to guess until he has discovered the colors and correct positions of all four pegs. The number of guesses is the score, and the player with the lower score wins. The C64 uses letters instead of pegs, but the rules are the same. In fact, the MASTER program offers you a choice of three different game versions, and you can modify the program to play even more games.

Operating Instructions for MASTER

1. Load MASTER *via* the disk MENU.
2. Type 'RUN' and press RETURN.
3. In response to '1 OR 2 PLAYERS?' press '1'.
4. Select EASY game by pressing the '1' key again.
5. Read the rules of the game from the screen. You are trying to discover a secret pattern of four letters. In this EASY version of the game, each of the letters A, B, C, and D will appear exactly once in the pattern.
6. Type in a four-letter pattern using the letters A, B, C, and D. Your guess may use repeated letters, but the secret pattern in this EASY version of the game does not.
7. Until you type the fourth letter, you may start your guess over by pressing the '←' key.
8. After you press the fourth letter, the computer will respond with two numbers. The first, in the 'ON' column, is the number of letters you guessed in the correct position. The second number, in the 'OFF' column, is the number of other letters you guessed correctly, but in the wrong position.
9. Use this information to determine the secret pattern. Repeat the steps from number 6 above until you have correctly guessed the secret pattern.
10. When you have guessed the pattern correctly, the C64 will congratulate you and tell you how many guesses you have taken.

```
SELECT GAME :  
1 EASY  
2 MID  
3 HARD  
OR  
4 EXIT TO BASIC  
5 EXIT TO DISK MENU
```

Running the Program

Execute MASTER *via* the MENU. The screen will clear and the message '1 OR 2 PLAYERS?' will be displayed at the top. For the moment, select '1'. The two-player game is described later. Next you are offered a menu of game difficulty levels. Press '1', '2', or '3' to select a game. (You can change your choice for the next game, if you want.) The rules appropriate to the game you have selected are then displayed. The rules are printed here for reference.

In the EASY game, only A, B, C, and D are allowed, and no letter may be repeated in the secret pattern. Your guesses may include repeated letters, though.

In the MID game, only A, B, C, and D are allowed, but these letters may be repeated in the pattern.

In the HARD game, A, B, C, D, E, and F are allowed, and letters may be repeated.

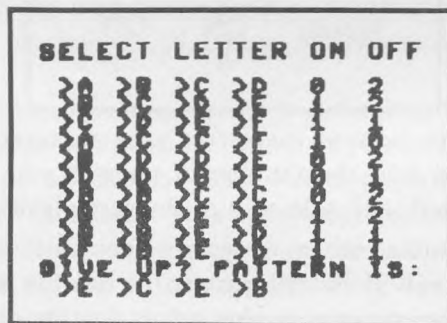
Press any key (except RUN/STOP) to continue. The computer now generates, at random, a secret pattern. The screen will clear and appear as below:

SELECT LETTER ON OFF

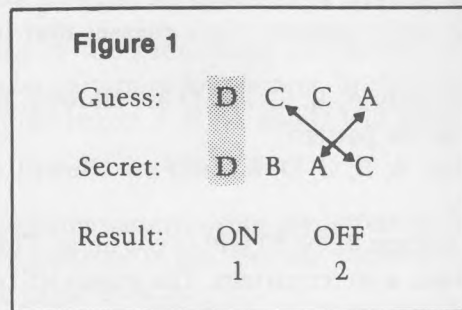
>

The flashing-square cursor appears right after the '>'. Only '?', '<', and the letters allowed in the game will be accepted from the keyboard.

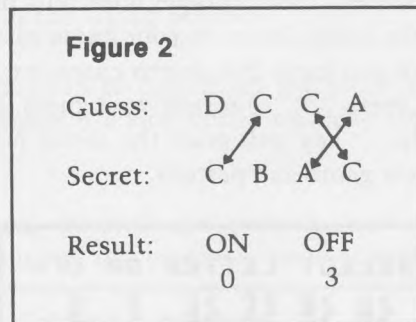
(The RUN/STOP key does work, though!) Acceptable characters will be printed on the screen; unacceptable ones will have no effect. As soon as you enter the fourth letter in your guess pattern, the program will process it. Until you enter that fourth character, though, you may change your mind. Press '<' to restart your guess. If at any time you want to give up, the '?' key will print the secret pattern and let you start over with a new game and pattern.



When you enter the fourth item in your guess pattern, the computer matches it against the secret pattern. In the 'ON' column is the number of letters guessed correctly, and in the right position; in the 'OFF' column is the number guessed right, but in the wrong position. Understanding the matching process will help you learn to play the game better. For instance, if you guessed 'D C C A', and the secret pattern is 'D B A C', the computer will return a '1' in the 'ON' column and a '2' in the 'OFF' column.



The 'D' is in the correct position (indicated by the shading in figure 1), but the 'C' and the 'A' (matches indicated with arrows), while they do exist in the secret pattern, were guessed in the wrong position. Only one of the C's in the guess is counted, since there is only one C in the secret pattern. If the secret pattern were 'C B A C' instead, the program would return '0' in the 'ON' column and '3' in the 'OFF' column. Both C's in the guess are now counted.



When you have guessed the secret pattern correctly, you will be congratulated and told the number of guesses you took. Then the program is restarted with selection of the game level.

As you play more and more games, you will begin to develop systems to help you guess the pattern as quickly as possible. One technique that is sometimes useful is substituting one character at a time.

Two-person Game

The two-person option allows a second player to input a secret pattern instead of having the computer come up with one. The player who will be guessing should look away from the screen while the other player inputs a pattern. The program tests for the letters allowed but it does not check for repetitions. Be sure to follow the repetition rule in effect. To go back to the one-person version press RUN/STOP, type RUN and press RETURN. This time answer '1' for the number of players.

Programming Techniques in MASTER

Random Numbers

In the one-player version of MASTER, the program is able to come up with a different secret pattern of letters each time the game is played. How is this done? The secret is in random numbers. BASIC is able to generate random numbers using the RND function.

A random number is one that is obtained without any predictability or repeatability. Rolling a die, flipping a coin, and spinning a roulette wheel are all means of obtaining random numbers in the real world.

Many programming applications require a source of random numbers. For statistics programs they can provide sample data to test a model, and in physics they can be used for applications such as demonstrating the behavior of gas particles.

Many game programs require random numbers. These may be used in the form of playing cards, dice, or locations of hidden treasures. In the 1-player version of MASTER, random numbers are used to generate the secret pattern of letters.

The BASIC function RND generates pseudo-random numbers in the range between 0 and 1. Pseudo-random means each succeeding number depends to some extent on the previous one. As a result, after many thousands of numbers, the sequence will start over. This makes statistics involving very large samples sometimes difficult, but it usually causes no problem in games, which use considerably fewer numbers.

The only problem we must avoid is generating the same sequence of random numbers every time the program is run. The technique used in MASTER is to be sure the RND function has a different starting number or 'seed' each time. This is accomplished with the statement 'I=RND(-TI)' in line 8050 as part of the initialization sequence. Using a negative number as the argument for RND causes a function of the argument to be used as the seed to start the sequence of random numbers. TI is the value of an internal clock that starts at zero when

the C64 is turned on and increments every sixtieth of a second. Since you are very unlikely to start the program at exactly the same moment each time, you are practically assured of getting a different seed each time. If you use a negative constant instead of $-TI$ in line 8050, you will generate the same sequence of random numbers each time. Run the program this way and you will be able to astound your friends with your psychic powers!

Now that we have a sequence of random numbers, how do we turn this sequence into the letter patterns for MASTER? Line 1020 does it all in one BASIC expression: $RN=INT(RND(1)*N+1)$, where N is number of letters allowed in the game. See figure 3 for a graphic illustration of how four random numbers are converted into the four letters of a secret MASTER pattern. $RND(1)$ produces numbers in the range of 0 to 1, but this does not include either 0 or 1 themselves. First we multiply the number by the number of letters allowed in the game. If we allow four letters ($N=4$), then we multiply the random numbers by four to get numbers in the range 0 to 3.999.... Next we add 1 to make it 1 to 4.999.... Then we use the BASIC INT function to remove whatever is to the right of the decimal point, leaving us with 1, 2, 3, or 4. These numbers are never actually converted to letters. Instead, the

Figure 3				
	Element #			
BASIC Function	1	2	3	4
RND(1)	.555877482	.689094948	.828839479	.0619696133
*4	2.22350993	2.75637979	3.31535792	.247878453
+1	3.22350993	3.75637979	4.31535792	1.24787845
INT()	3	3	4	1
LETTER	C	C	D	A

letters the player types for a guess are converted to numbers.

The same technique can be used to get random numbers over any range. For dice, multiply by six, take the integer, and add 1. For playing cards, multiply by 52, take the integer, and add 1. (Converting 1 to 52 into suits and ranks is another problem!)

You don't have to restrict yourself to a range of numbers or letters, either. Try the following short program. (You'll have to type NEW and press RETURN before you type it in. To use MASTER again, reload it from the disk.)

```

10 X=RND(-TI)
20 A$(1)="APPLE":A$(2)="PEAR":A$(3)="BANANA":A$(4)="GRAPE"
30 X=INT(4*RND(1)+1)
40 PRINT A$(X)
50 GOTO 30

```

A random sequence of fruits! Perhaps this is the beginning of a slot machine program? The strings defined in line 20 could be combinations of graphic characters and cursor controls that actually draw pictures, such as dice or playing cards, on the screen. See the "C64 Clock" chapter to learn about programming cursor controls into strings and positioning images on the screen.



Flags and Logic

One of the most powerful features of a computer is its ability to make decisions. MASTER uses the C64's decision-making ability throughout its program.

Every decision boils down to deciding whether an expression is true or false. The BASIC IF...THEN construction decides whether an expression is true or false. If the expression after the IF is true, then whatever appears on the line after the THEN is executed. If the expression is false, then the rest of the line is skipped and execution continues with the next line.

BASIC doesn't actually handle the words 'true' and 'false.' Instead, it assigns -1 to represent 'true' and 0 to represent 'false'. When evaluating expressions, any non-zero result is considered 'true'. To see this in action try the following example:

```

10 INPUT "A=";A
20 INPUT "B=";B
30 IF A=B THEN PRINTY "TRUE":GOTO 50
40 PRINT "FALSE"
50 PRINT A=B
60 GOTO 10

```

Run this program. Type in a value for A, press RETURN, type in a value for B, and press RETURN. If the number you entered for A equals the number you typed for B, then 'TRUE' will be printed, followed by -1. Otherwise 'FALSE' is printed, followed by 0. The number -1 or 0 is the value BASIC assigned to the expression 'A=B'. Line 190 in MASTER checks to see if the number of correct position matches (PM) is equal to the number of letters (NN) in the pattern. If so, the player has correctly guessed the pattern and the congratulation routine 6000 is executed before starting a new game by returning to line 100.

Now enter the following program example that demonstrates the use of a flag.

```
10 INPUT A
20 IF A THEN PRINT "TRUE":GOTO 40
30 PRINT "FALSE"
40 GOTO 10
```

Try a few numbers. Every number except 0 will result in 'TRUE' being printed. Entering 0 will produce a 'FALSE'. The 'A' in line 20 is evaluated just like any other expression. If it is non-zero then it is considered true.

A flag is a convenient device in a program. It can be either set (true or -1) or clear (false or 0). BASIC doesn't have a special variable type for flags, but either integer or floating point variables may be used that way. MASTER uses several variables as flags: RP, RQ, and the arrays PF(-) and PG(-). RP is set or cleared in the game selection routine (in line 7100, 7200, or 7300), depending on the game chosen. In line 1030, if RP is set (= -1) then lines 1040-1080, which prevent duplicate letters in the pattern, are skipped. RQ stays cleared unless a duplicate letter is found. If the flag is set, then the program returns to 1020 to determine a new number. Each element of the secret pattern has an element in the flag array PF(-), and each element in the guessed pattern has an element in the flag array PG(-). See the discussion under "Processing a Guess" for details of how these flags are used.

Another interesting use of a flag is in the display of the congratulation message (6080-6150). A FOR...NEXT loop is used to alternate the variable I between -1 and 0. The flag I is tested in lines 6090 and 6120. If the flag is set, then the reverse-field character is printed. When the flag is clear, the following message is printed in normal characters. This produces the alternating reverse-field effect.

The program has to make decisions in a number of other places, evaluating an expression to determine what to do next. The IF...THEN statement is used most commonly for decision making, but ON...GOSUB and ON...GOTO are also used. ON...GOSUB is used in

line 110 to decide whether to generate a random pattern in a 1-player game, or to let a player input a pattern.

Processing a Guess

As explained earlier, the match count is determined by first checking for exact position matches and then going through to check for out-of position matches. No element in either the secret or guess pattern may be used more than once in a match.

To avoid re-using pattern elements in matches, we need to program a way to "cross off" pattern elements that have been used in a match. In addition to the two arrays of the elements themselves, two corresponding flag arrays are used. The lines of the program involved in checking for matches are reproduced below.

```
3000 REM CLEAR FLAGS
3010 FOR I=1 TO NN
3020 PF(I)=0:PG(I)=0
3030 NEXT I
3040 REM CHECK FOR POSITION MATCHES
3050 FOR I=1 TO NN
3060 IF R(I)=GU(I) THEN PF(I)=-1:PG(I)=-1:PM=PM+1
3070 NEXT I
3500 REM CHECK FOR OTHER MATCHES
3510 FOR I=1 TO NN
3520 IF PG(I) THEN 3570
3530 FOR J=1 TO NN
3540 IF PF(J) THEN 3560
3550 IF R(I)=GU(J) THEN OM=OM+1:PF(J)=-1:PG(I)=-1:J=NN
3560 NEXT J
3570 NEXT I
3580 RETURN
```

At the beginning of the matching process, all the flags are cleared, or set to zero (3010-3030). As each match is detected, the flags corresponding to the matched elements are set (in lines 3060 and 3550). The flags are checked in lines 3520 and 3540. If the flag is set, then the matching process is skipped and the next element is checked. In addition, when a match is found in line 3550, the higher numbered elements in the guess pattern are skipped by setting the loop index J to its maximum value, NN. The NEXT J statement in line 3560 sees J equal to its maximum value and is fooled into thinking it's through with the specified repetitions. Control passes to the NEXT I statement in line 3570.

This process is graphically demonstrated in figure 4. I is the index into the secret pattern, while J is the index into the guess pattern. The boxes indicate the two elements currently being compared, PM is the number of position matches, and OM is the number of out-of-position matches. A shaded box indicates a match and a diagonal line through an element indicates that it has been used in a match already. First, the position matches are checked. The result is 1, with the D's in the first position crossed off. In the program, the flags PF(1) and PG(1) are set to -1.

Figure 4

Secret Pattern				Guess Pattern				PM	OM		
I				J				0	0		
1	D	B	A	C	1	D	C	C	A	1	0
2	B	B	A	C	2	B	C	C	A	1	0
3	B	B	A	C	3	B	C	C	A	1	0
4	B	B	A	C	4	B	C	C	A	1	0
2	B	B	A	C	2	B	C	C	A	1	0
2	B	B	A	C	3	B	C	C	A	1	0
2	B	B	A	C	4	B	C	C	A	1	0
3	B	B	A	C	2	B	C	C	A	1	0
3	B	B	A	C	3	B	C	C	A	1	0
3	B	B	A	C	4	B	C	C	A	1	1
4	B	B	A	C	2	B	C	C	A	1	2

Next, the out-of-position matches are checked. Since the first elements in each pattern have already been used, the comparison begins with the second elements. No match is found for the B, so the search continues with the third element of the secret pattern and the second element of the guess pattern. When the match is found with the fourth guess element, these two are crossed off, and the out-of-position match counter OM is incremented. A match is found immediately for the fourth secret pattern element, so the remaining two elements are skipped, and the counter incremented again. *One* position match and *three* out-of-position matches are reported to the player.

If you are still confused about how this works, try a different pattern and construct a table similar to figure 45. You might also try running through the program lines a specific example of your own.

Program Description

Initialization (10): Subroutine 8000 sets up a number of constants, and subroutine 7500 gets the number of players.

Program mainline (100-200): Subroutine 7000 gets the skill level for the game and displays the instructions for the game. Subroutine 5000 waits for a key to be pressed before continuing with the main program.

Line 110 uses the ON...GOSUB structure to determine whether to call subroutine 1000, which generates a random pattern, or subroutine 4000, which allows one player to input a pattern. NP can have only two values, 1 or 2. On 1, subroutine 1000 is called; on 2, subroutine 4000 is called.

GN is used to count the number of guesses. Line 130 calls subroutine 2000, which prints the header on the screen and receives the first guess. The second and subsequent guesses return to line 140 where the same subroutine is called at 2020, to avoid having the header reprinted for each guess.

A '?' indicates that the player has given up. A call is made to subroutine 9000, which prints out the secret pattern. GOTO 100 starts the player out with a new game.

Next the guess must be processed. Before each call to the processing routines, the match counters PM and OM are zeroed. Subroutine 3000 processes the guess, first checking for position matches and then for out-of-position matches. If PM (the number of position matches) equals NN (the number of elements in the pattern), then the player has guessed the pattern. Subroutine 6000 is the congratulations routine.

Line 180 prints out the results of the matching, with the position matches under the heading 'ON' and the out-of-position matches under the heading 'OFF'. When the TAB(19) expression is encountered in a PRINT statement the cursor moves to the nineteenth character position on the screen.

The guess counter GN is incremented and the program loops back to 140 for another guess.

Generate Random Numbers (1000-1110): This routine is called at the beginning of each one-player game to generate the secret pattern. In the supplied version of the game NN is always 4, so four numbers are generated. Line 1020 returns in RN an integer between 1 and the number of letters allowed in the game (N). If RP is non-zero, then repeats are allowed in the pattern. Lines 1040-1080 are skipped and RN is copied into R(I), the current element of the pattern. If repeats are not allowed (RP = 0), then each RN must be checked against the previous

elements in the pattern R(). In line 1040 RQ is set to 0 to indicate that no element has been found so far to match RN. If I = 1 then there aren't any numbers in the pattern and we can skip to 1090 and accept this RN. The FOR...NEXT loop on J (lines 1050-1070) goes from 1 to the previous element (IS11). If RN is found to match an existing element (RN = R(J)) then RQ is set to -1 to indicate a match has been found and J is set to I - 1 to terminate the FOR...NEXT loop. If no match is found, then the loop continues through all the previously assigned elements. RQ is tested in line 1080: if it is non-zero, then another RN must be calculated (return to 1020); if it is still zero, then we can accept the RN and install it in the current element R(I) of the pattern. The outside FOR...NEXT loop (1010 to 1100) continues until all of the elements required in the pattern have been calculated.

Process Guess (2000-2140): As discussed above under the program mainline, this routine is usually called at 2020, but the first time the call is made to 2000 to print the heading 'SELECT LETTER ON OFF'.

The routine consists of a big FOR...NEXT loop, where I starts with a value of 1 and ends with the value NN, the number of elements in the pattern. Within this loop, characters from the keyboard are accepted or rejected. The GET function returns with a character from the keyboard. If no key has been pressed, then the string T\$ is assigned a null value. As long as T\$ continues to be a null string, the program will keep looping on line 2060. As soon as a key is pressed, the program continues at line 2070. Normally, when the GET function is used, the cursor does not flash. POKEing a 0 into CF (a constant set to 204 in the initialization) starts the cursor flashing; POKEing a 1 turns it off. It must be turned off between GETs to avoid depositing cursor characters in unwanted places.

Two special characters " and '?' are tested. On ", the loop is terminated by setting I to NN and executing a NEXT statement. The GOTO 2030 starts the loop over again. If we had failed to terminate the loop (by omitting the I = NN and NEXT statements) the user would be able to crash the program by repeatedly hitting the " key. BASIC keeps track of each FOR...NEXT loop in an area of memory called the stack. If we don't terminate a loop, that information continues to occupy space on the stack. Repeated calls to 2030 with the " key will continue to build up new FOR...NEXT information on the stack until there is no room left. At this point the program crashes with an ?OUT OF MEMORY ERROR. The '?' character is dealt with similarly. The FOR...NEXT loop is terminated and a RETURN is made to the program mainline.

Other characters are converted in line 2100 to their numeric codes with the ASC function. The code for the letter A is 65, so subtracting 64 converts letters into numbers beginning with 1. If T is less than 1 or greater than the number of letters allowed in the pattern, then BK\$ (a constant defined as two [CL] characters) is printed and the program branches to 2060 to GET another character. If the character is accepted, then the appropriate colored letter block OB\$(T) is printed and the number T is stored in the current element of the guess pattern GU(I). RETURN takes the flow back to the mainline.

Matching Routines (3000-3580): These routines are described in more detail in the main text under the section "The Matching Process."

Lines 3010-3030 clear the flag arrays PG() and PF() by setting them to zero. Lines 3050-3070 advance, position by position, through the secret pattern R() and guess pattern GU() arrays checking for matches. If a match is found, the position match counter PM is incremented and the corresponding flags are set to -1.

Lines 3500-3580 check all the other possibilities for matches. The flags are used to cross off elements as they are matched. Some economy is achieved by skipping over crossed-off elements (lines 3520 and 3540) and by terminating the inside loop as soon as a match is found (JS5NN at the end of line 3550).

Input Pattern with Two Players (4000-4160):

After the instructions are displayed, this routine accepts letters one-by-one until the pattern is filled. It is similar to the guess-processing routine (2000-2130). Instead of filling the guess array, the secret pattern array R() is filled. See the description above for details.

ANY KEY WHEN READY (5000-5020): The string CN\$ is a constant defined in the initialization routine. The result is to print the message 'ANY KEY WHEN READY' at the bottom of the screen. The one-line GET loop must receive a key before a RETURN.

Congratulation Routine (6000-6190): 6010-6060 use the number of guesses GN to determine the congratulation message. The array G() is set up for each version of the game in 7100, 7200, or 7300. The messages MS\$() are set up as constant strings during initialization. By comparing GN to each cutoff value G() with the < = (less than or equal to) operation, the subscript MS for the array MS\$() is determined.

Lines 6070-6180 display the congratulation screen, alternating the message and the number of guesses between reversed and normal characters. The use of I as a flag is discussed above under "Flags." If

I = - 1 then the [RVS] character is printed. Its value alternates between - 1 and 0.

The string functions STR\$ and MID\$ applied to GN in line 6130 make the display of the number of guesses more attractive. The STR\$ function converts the number GN to a string of characters. Positive numbers leave a space in front of the numeric characters that normally would be occupied by the ' - ' character. To get just the numeric part of the string the MID\$ is used in a special way to get all the characters from the second position on. Normally the items included in the parentheses after MID\$ are the name of the string, the character position to start, and the number of characters to extract. If just the first two items are included, then the remainder of the string is the result. Specifying 2 for the second parameter converts the string of the number of guesses to the same string without the leading space.

Line 6140 is a FOR...NEXT loop that does nothing between the FOR and the NEXT! By adjusting the number after the TO, you can achieve a delay in the program of nearly any desired time. Here it controls the rate of the flashing.

The GET function is used in line 6170 in a way opposite to its use in 2050, 4090, 5000, and 7060. As long as no key is pressed (T\$ does not equal " "), the message continues to flash. When a key is pressed, the RETURN instruction is executed.

Select Game and Display Instructions (7000-7450): 7010-7055 display a menu listing the different games available. Line 7060 awaits a key, which is converted to a number and tested against the range of the menu in line 7070. If the key is out of range, then the program branches back to 7060 for another key. The ON...GOSUB instruction in line 7080 calls 7100 if T is 1, 7200 if T is 2, 7300 if T is 3, 7600 if T is 4 or 20000 if T is 5.

Each of these set-up-and-display routines establishes N (the number of letters allowed in the game), RP (the flag determining whether or not repeats are allowed), and G() (the array of guess number cutoff values for the congratulation messages). Then the name of the game is displayed. Next subroutine 7400, which displays parts of the instructions common to all games, is called. Finally, the rule regarding repeats is printed in the proper place.

Subroutine 7400 first prints the colored letter blocks corresponding to the number of letters allowed (N). If the number allowed is four, the first four letter blocks are printed. Then the portion of the directions common to all versions of the game is printed.

Get Number of Players (7500-7520): This subroutine is called once when the program is first run. It uses subroutine 5010 (5000 without

the ANY KEY WHEN READY message) to GET a key. Only 1 or 2 is accepted and the value is returned in NP.

End (7600): Exits to BASIC when called by T = 4 in the menu above.

Initialization (8000-8080): Sets up constants used in the program. See variable usage table for descriptions of the variables.

Print Pattern on Give-up (9000-9030): The secret pattern is printed out in the appropriate colored letter blocks, using the secret pattern array R(). Subroutine 5000 is used to wait for a key before starting a new game.

Menu Loader (20000 to 20020): This reloads and runs the disk menu program. Line 20010 clears the display and then prints a set of commands on the display, exactly as you would type them to LOAD "0:MENU",8 and after the program was loaded from disk would type RUN to execute it. Line 20020 POKes into the keyboard buffer a HOME (19) and two RETURN (13) codes. It POKes the number 3 into the keyboard character counter and then exits to BASIC with the END. BASIC now executes from the keyboard buffer the HOME to set the top of the display, the first RETURN to execute the LOAD and then the second RETURN after the MENU program has been reloaded to RUN it.

Customizing your MASTER Game

Adding an EXPERT Level

Because of the way MASTER is written, it is easy to add your own version to the game. As an example of how to do this, let's add an EXPERT game to the three choices we have already. Add or substitute the following lines to the program supplied.

```
2100 PRINT TAB(7) "{DOWN,RIGHT2,RVS}6{RVSOFF} EXPERT"  
7070 T=VAL(T$):IF T<1 OR T>6 THEN 7060  
7080 ON T GOSUB 7100,7200,7300,7600,20000,7350  
7350 N=8:RP=-1:G(1)=8:G(2)=12:G(3)=16:G(4)=20:G(5)=25  
7630 PRINT "{CLEAR} EXPERT GAME:"  
7370 GOSUB 7400  
7380 PRINT TAB(7) "{DOWN,RIGHT2}MORE THAN ONCE"  
7390 RETURN
```

This version of the game allows the first eight letters of the alphabet. The operation of the game itself is controlled by the values of N and RP in line 7350. The rest of the program changes involve adding

the game to the menu and displaying the rules. The value of N determines the number of letters allowed in the game. RP is a flag, which, if set, allows repeats of letters in the pattern (see the "Flags" section above). The array G(-) holds the cut-off numbers of guesses for each congratulation message. Adjust these values and program the appropriate messages, as in the example above, and you will be able to add your own game version.

Congratulation Messages

As part of the initialization routine, six congratulation messages are defined in lines 8060 and 8070. You can change these messages as well as the cut-off values G(-) defined in lines 7100, 7200, and 7300.

Number of Elements in Pattern

The number of elements is four for all versions of the game described so far. This number can be changed to practically any number, the only limitation being the width of the display. The number of elements in the pattern is determined by the value of NN in line 8050 of the initialization routine. Change the first part of line 8050 to read: 8050 NN = 3. Now run the program. Notice that everything works as before, except only three letters are generated in the secret pattern, and only three are expected in each guess.

To program more elements in the pattern, two additional changes must be made, both in line 8050:

```
8050 NN=6:FD$="{BLACK} > ":BK$="{LEFT2}":CR$=CHR$(13):
      CF=204:I=RND(-TI)
```

With this change, five or six elements can be accommodated without disturbing the rest of the display. Substitute in line 8050, as above: 8050 NN = 5 or 8050 NN = 6. To accommodate longer patterns, remove the TAB(7) portion of the PRINT statements in lines 2030 and 4060.

Patterns of 11 or more elements require a DIM statement in the initialization routine. For example,

```
8055 DIM R(NN),GU(NN),PF(NN),PG(NN)
```

along with one of the display adjustments above, sets up the game for 11 elements.


```

10 REM MASTER - A GUESSING GAME
20 GOSUB 8000: GOSUB 7500:
    REM SET UP CONSTANTS, SELECT # OF PLAYERS
100 GOSUB 7000: GOSUB 5000: REM SELECT GAME LEVEL, WAIT FOR KEY
105 IF T$="@" THEN 100
110 ON NP GOSUB 1000,4000: REM RANDOM OR PLAYER-INPUT PATTERN
120 GN=1
130 GOSUB 2000: GOTO 150: REM HEADER, ENTER GUESS
140 GOSUB 2020: REM ENTER GUESS (NO HEADER)
150 IF T$="?" THEN GOSUB 9000: GOTO 100
155 IF T$="@" THEN 100
160 PM=0: OM=0
170 GOSUB 3000: REM PROCESS GUESS, RETURN PM,OM
180 PRINT TAB(19){BLACK} "PM;OM
190 IF PM=NN THEN GOSUB 6000: GOTO 100
200 GN=GN+1: GOTO 140
1000 REM GENERATE RANDOM NUMBERS
1010 FOR I=1 TO NN
1020 RN=INT(RND(1)*N+1)
1030 IF RP THEN 1090
1040 RQ=0: IF I=1 THEN 1090
1050 FOR J=1 TO I-1
1060 IF RN=R(J) THEN RQ=-1: J=J-1
1070 NEXT J
1080 IF RQ THEN 1020
1090 R(I)=RN
1100 NEXT I
1110 RETURN
2000 REM PRINT HEADER
2010 PRINT "{CLEAR}          {RVS,PURPLE}SELECT LETTER
      {RVSOFF} ON OFF{DOWN}"
2020 REM PROCESS GUESS
2030 PRINT TAB(7);: FOR I=1 TO NN
2040 PRINT FD$;
2050 POKE CF,0: POKE CF+3,0
2060 GET T$: IF T$="" THEN 2060
2070 POKE CF,1
2080 IF T$="{BACK ARROW}" THEN PRINT CR$"{UP,SPACE19,UP}":
      I=NN: NEXT: GOTO 2030
2090 IF T$="?" OR T$="@" THEN I=NN: NEXT: GOTO 2140
2100 T=ASC(T$)-64: IF T<1 OR T>N THEN PRINT BK$;: GOTO 2040
2110 PRINT OB$(T);

```

```

2120 GU(I)=T
2130 NEXT I
2140 RETURN
3000 REM CLEAR FLAGS
3010 FOR I=1 TO NN
3020 PF(I)=0: PG(I)=0
3030 NEXT I
3040 REM CHECK FOR POSITION MATCHES
3050 FOR I=1 TO NN
3060 IF R(I)=GU(I) THEN PF(I)=-1: PG(I)=-1: PM=PM+1
3070 NEXT I
3500 REM CHECK FOR OTHER MATCHES
3510 FOR I=1 TO NN
3520 IF PG(I) THEN 3570
3530 FOR J=1 TO NN
3540 IF PF(J) THEN 3560
3550 IF R(I)=GU(J) THEN OM=OM+1: PF(J)=-1: PG(I)=-1: J=NN
3560 NEXT J
3570 NEXT I
3580 RETURN
4000 REM INPUT PATTERN
4010 PRINT "{CLEAR}          ONE PLAYER ENTERS"
4020 PRINT TAB(7) " PATTERN"
4030 PRINT TAB(7) "WHILE OTHER PLAYER"
4040 PRINT TAB(7) " LOOKS AWAY"
4050 PRINT TAB(7) "{DOWN}ENTER PATTERN:"
4060 PRINT TAB(7);: FOR I=1 TO NN
4070 PRINT FD$;
4080 POKE CF,0: POKE CF+3,0
4090 GET T$: IF T$="" THEN 4090
4100 POKE CF,1
4110 IF T$="{BACK ARROW}" THEN PRINT BK$: I=NN: NEXT: GOTO 4060
4120 T=ASC(T$)-64: IF T<1 OR T>N THEN 4080
4130 PRINT OB$(T);
4140 R(I)=T
4150 NEXT
4160 RETURN
5000 PRINT CN$
5010 GET T$: IF T$="" THEN 5010
5020 RETURN
6000 REM CONGRATULATIONS
6005 PRINT: PRINT "{DOWN,BLACK}          THAT IS CORRECT!"

```

```

6006 PRINT "{DOWN}          {RVS,BLUE}PRESS ANY KEY TO
      CONTINUE{RVSOFF}"
6008 GET T$: IF T$="" THEN 6008
6010 IF GN=1 THEN MS=1: GOTO 6070
6020 IF GN<=G(1) THEN MS=2: GOTO 6070
6030 IF GN<=G(2) THEN MS=3: GOTO 6070
6040 IF GN<=G(3) THEN MS=4: GOTO 6070
6050 IF GN<=G(4) THEN MS=5: GOTO 6070
6060 MS=6
6070 PRINT "{CLEAR}          ";
6080 FOR I=-1 TO 0
6090 IF I THEN PRINT "{RVS}";
6100 PRINT "{HOME,RED}"MS$(MS)
6110 PRINT TAB(7)"{DOWN,PURPLE}YOU TOOK {CYN}";
6120 IF I THEN PRINT "{RVS}";
6130 PRINT MID$(STR$(GN),2);"{RVSOFF,PURPLE} TRIES!"
6140 FOR J=1 TO 200: NEXT J
6150 NEXT I
6160 PRINT CN$
6170 GET T$: IF T$<>"" THEN RETURN
6180 GOTO 6080
6190 RETURN
7000 REM PROCESS INITIAL CONDITIONS
7010 PRINT "{CLEAR,BLACK}          SELECT GAME:"
7020 PRINT TAB(7)"{DOWN,RIGHT2,RVS}1{RVSOFF} EASY"
7030 PRINT TAB(7)"{DOWN,RIGHT2,RVS}2{RVSOFF} MID"
7040 PRINT TAB(7)"{DOWN,RIGHT2,RVS}3{RVSOFF} HARD"
7045 PRINT TAB(7)"{DOWN,RIGHT2} OR"
7050 PRINT TAB(7)"{DOWN,RIGHT2,RVS}4{RVSOFF}
      EXIT TO BASIC"
7055 PRINT TAB(7)"{DOWN,RIGHT2,RVS}5{RVSOFF}
      EXIT TO DISK MENU"
7060 GET T$: IF T$="" THEN 7060
7070 T=VAL(T$): IF T<1 OR T>5 THEN 7060
7080 ON T GOSUB 7100,7200,7300,7600,20000
7090 RETURN
7100 N=4: RP=0: G(1)=3: G(2)=5: G(3)=7: G(4)=10: G(5)=15
7110 PRINT "{CLEAR,BLACK}          EASY GAME:"
7120 GOSUB 7400
7130 PRINT TAB(7)"{DOWN,RIGHT2}ONLY ONCE"
7140 RETURN
7200 N=4: RP=-1: G(1)=4: G(2)=6: G(3)=8: G(4)=12: G(5)=18

```

```

7210 PRINT "{CLEAR}          MID GAME:"
7220 GOSUB 7400
7230 PRINT TAB(7) "{DOWN,RIGHT2}MORE THAN ONCE"
7240 RETURN
7300 N=6: RP=-1: G(1)=5: G(2)=7: G(3)=10: G(4)=15: G(5)=20
7310 PRINT "{CLEAR}          HARD GAME:"
7320 GOSUB 7400
7330 PRINT TAB(7) "{DOWN,RIGHT2}MORE THAN ONCE"
7340 RETURN
7400 PRINT TAB(7) "{BLUE,DOWN,RIGHT2}";: FOR I=1 TO N:
      PRINT OB$(I) " ";: NEXT: PRINT ALLOWED"
7410 PRINT TAB(7) "{DOWN,RIGHT2}EACH MAY BE USED"
7420 PRINT TAB(7) "{DOWN6,RIGHT2,BACK ARROW} TO CLEAR GUESS"
7430 PRINT TAB(7) "{DOWN,RIGHT2}? TO GIVE UP"
7440 PRINT TAB(7) "{DOWN,RIGHT2}@ TO RETURN TO
      MAIN MENU{HOME,DOWN4}"
7450 RETURN
7500 PRINT "{CLEAR}          {RED}MASTER {BLACK}-{BLACK} {BLUE}
      A GUESSING GAME{BLACK}"
7510 PRINT "{DOWN}          {BLACK,RVS}1{RVSOFF} OR {RVS}2
      {RVSOFF} PLAYERS?"
7520 GOSUB 5010: IF T$ < "1" OR T$ > "2" THEN 7510
7530 NP=VAL(T$): RETURN
7600 END
8000 REM SET-UP OF CONSTANTS
8010 POKE 53280,6: POKE 53281,1
8020 OB$(1)="{RED}A{BLACK}": OB$(2)="{BLUE}B{BLACK}":
      OB$(3)="{GREEN}C{BLACK}"
8030 OB$(4)="{PURPLE}D{BLACK}": OB$(5)="{CYN}E{BLACK}":
      OB$(6)="{YELLOW}F{BLACK}"
8040 CN$="{HOME,DOWN20,RIGHT9,RVS}ANY KEY WHEN READY"
8050 NN=4: FD$="{BLACK} > ": BK$="{LEFT2}": CR$=CHR$(13):
      CF=204: I=RND(-TI)
8060 MS$(1)="A PSYCHIC!": MS$(2)="EXCELLENT!":
      MS$(3)="VERY GOOD!"
8070 MS$(4)="GOOD": MS$(5)="FAIR": MS$(6)="TRY,TRY,TRY AGAIN!"
8080 RETURN
9000 REM PRINT PATTERN ON GIVE-UP
9010 PRINT CR$"{UP}          {RVS}GIVE UP?{RVSOFF} PATTERN IS:"
9020 PRINT TAB(7);: FOR I=1 TO NN:
      PRINT " > "OB$(R(I))"{RVSOFF}";: NEXT
9030 GOSUB 5000: RETURN

```

```

200000 REM MENU LOADER
20010 PRINT "{CLEAR}LOAD "+CHR$(34)+"0: MENU"+CHR$(34)+"",8":
      PRINT "{DOWN4}RUN"
20020 POKE 631,19: POKE 632,13: POKE 633,13: POKE 198,3: END

```

MASTER Variable Usage

Constants

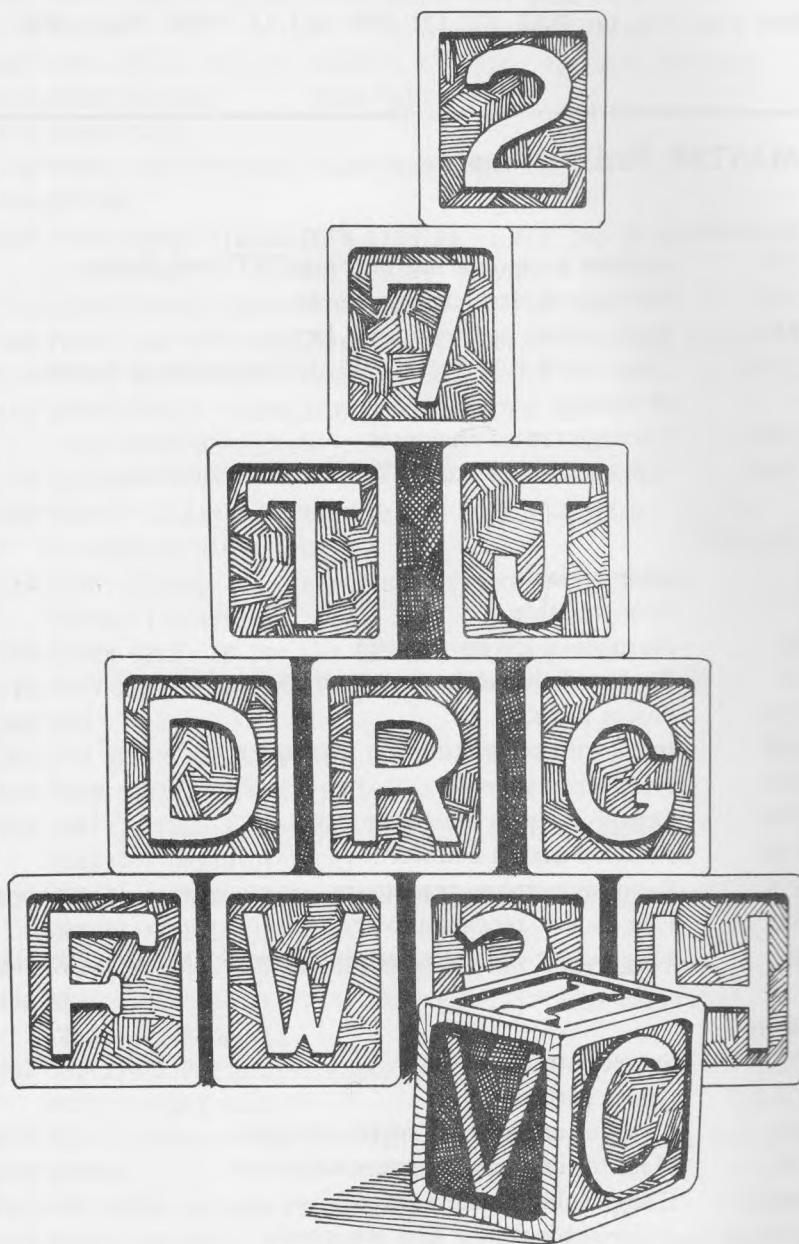
CF	Address for cursor flash during GET (= 0: flash)
NN	Number of elements in pattern
BK\$	Back up two spaces on bad key
CN\$	Print 'ANY KEY WHEN READY' message at bottom of screen
CR\$	Carriage return character
FD\$	"{BLACK,SPACE}" to GET next guess element

Variables

I	Loop variable
J	Loop variable
N	Number of letters allowed
T	Keyboard character converted to number
GN	Guess number
MS	Index into congratulation message array
NP	Number of players
OM	Out-of-position match counter
PM	Position match counter
RN	Random number, temporary
RP	Flag (= - 1: repeats allowed)
RQ	Flag (= - 1: indicates number already used in pattern)

Arrays

R(-)	Secret pattern
GU(-)	Guess pattern
PF(-)	Flags for crossing off secret elements
PG(-)	Flags for crossing off guess elements
MS\$(-)	Congratulation messages
OB\$(-)	Colored blocks for first six letters



Word Detective

A Word Guessing Game

WORD DETECTIVE
IF I LIKE A WORD, I MAKE IT RED
IF I HATE A WORD, I MAKE IT GREEN
FIGURE OUT MY RULE BY TYPING
IN DIFFERENT WORDS.
WHEN YOU HAVE FIGURED OUT THE RULE,
OR YOU GIVE UP, THEN TYPE:
THE RULE IS TO DISPLAY THE RULE.
TO DISPLAY ALL OF THE RULES, TYPE:
THE RULES ARE
PRESS F1 FOR DISK MENU, F3 FOR BASIC
YOUR WORD:

Word Detective is a game that challenges you to figure out why the computer likes some words and hates other words. By typing in different words, you attempt to figure out what rule the computer is using to decide whether it likes or hates a word. The game introduces the concept of the computer handling letters and words, as well as numbers. Detailed listings and instructions show you how to modify the program to use rules that you create.

Operating Instructions

1. Execute WORD DETECTIVE from the *Mastering the Commodore 64* disk menu.
2. You will be presented with the basic rules for WORD DETECTIVE.
3. Type in a word for WORD DETECTIVE to analyze.
4. When you press the RETURN at the end of the word, WORD will turn the word GREEN if it hates the word or RED if it likes the word.
5. Continue giving WORD one word at a time and try to guess the rule. When you think you know the rule, try some additional words to make sure that your hunch is correct.
6. When you are convinced that you know the rule, type 'THE RULE IS'. WORD will then display the rule.
7. Each time you play WORD, it randomly selects one of five rules. Can you guess what they are?
8. If you want to see all five rules, type 'THE RULES ARE'. Press F1 to get to the disk menu to run other disk programs. Press F3 to get back to BASIC.

Programming Concepts

While we normally think of computers as devices for dealing with numbers (referring to them sometimes as 'number crunchers'), they also can deal effectively with alphabetic material. One of the major uses of microcomputers today is as a word processor — a machine that can replace the typewriter and offer many kinds of support to the writer. Word processors can provide editing features such as character insert, character delete, line insert, line delete; they can move blocks of data, and more. Spelling-checking programs can eliminate most spelling errors. Text can be stored on cassette or disk, sent from one computer to another, and printed out in various formats. BASIC has a number of commands to assist in dealing with non-numeric information, and the most important ones are demonstrated in WORD DETECTIVE.

Processing ALPHABETIC Data

A CHARACTER is a single unit of information such as a letter (A d X), or a digit (0 1 2), a punctuation mark (! ? '), a mathematical symbol (= < > +), a special symbol (\$ # [@), or various graphic symbols and control characters. BASIC can handle characters in a number of ways:

1. It can reference them directly, as in a PRINT string, with a string of characters enclosed in quotation marks: PRINT "THESE ARE CHARACTERS".
2. It can refer to them as a variable: C\$ = "LETTER".
3. It can place them in arrays: A\$(3) = "7", etc.

The STRING Concept

The Commodore 64 can work with a single letter, a word, a sentence, or a large portion of text as a single entity. This collection of letters (which can include digits and most special characters) is called a 'STRING'. BASIC keeps track of strings separately from other types of information, such as floating point and integer numbers. It names them with a one- or two-character name, like the numbers, but adds the dollar sign (\$) to show that a string of characters is being referenced. Examples of string names include A\$, Z4\$, and BT\$. A string may have no characters, in which case it is called a NULL string. A string may have a single character or may be very long, often limited only by the amount of memory the system contains.

In WORD DETECTIVE we demonstrate several different ways to handle strings. An examination of some of these methods should provide a basic understanding of the string-processing capabilities of BASIC.

```
120 GET A$:IF A$="" THEN 120
```

The GET statement accepts a single character from the keyboard if one is present. If no key is pressed, then the variable, A\$, will be 'empty' and will equal the 'null' string or "". The program will continue to loop at line 120 until some key is pressed. Lines 130 to 165 test for special control characters and branch off to processing routines. Otherwise, control drops through to line 170.

```
170 Z$=Z$+A$
```

The individual characters that are input by the GET statement in line 120 are combined to make a single word by adding them together in line 170. This line takes the current contents of the Z\$ string, which is initially set empty by Z\$="" in line 115, and adds to it the character in the string A\$.

```
130 IF A$="{F1}" THEN 20000: REM F1 FOR DISK MENU
```

The entire contents of the Z\$ string may be tested by comparing it to a string that is specified by a pair of quotation marks. If the Z\$ string matches the string "THE RULE IS" exactly, character by character, then the subroutine at 3000 will be executed. If there is not an exact match, then the next instruction will be executed.

In addition to dealing with the entire text string, as in the INPUT, IF, or PRINT statements, BASIC can deal with portions of the total string. To work with part of the string, BASIC needs functions to isolate that part of the string that is of interest. Three functions are provided for this purpose:

1. LEFT\$(Z\$,N), which will isolate the LEFT N characters of the Z\$ string;
2. RIGHT\$(Z\$,N), which will isolate the RIGHT N characters of the Z\$ string;
3. MID\$(Z\$,M,N), which will isolate the MIDDLE N characters of the Z\$ string starting at the character in position M.

Let us assume that the INPUT to the Z\$ string was the word "SAMPLE".

```
210 F$=LEFT$(Z$,1)
```

F\$ is set equal to "S", the first character in "SAMPLE". The LEFT\$ function (Z\$,1) says to take one character from the left end of the string Z\$.

```
220 L$=RIGHT$(Z$,1)
```

L\$ is set equal to "E", the last character in "SAMPLE". The RIGHT\$

function (Z\$,1) says to take one character from the right end of the string Z\$.

```
3000 V$=MID$(Z$,K,1)
```

V\$ is set equal to the character at position K in the string Z\$: 'S' when K is 1, 'A' when K is 2, and so on up to 'E' when K is 6. The MID\$(Z\$,K,1) says to take one character from the K position in the string Z\$.

In the above examples, a single letter has been isolated from the entire string. Substrings (parts of strings) may be isolated in the same manner. The number 1 would be changed to the length of the substring that was to be isolated — 2 for two characters, 3 for three characters, and so forth up to the length of the string.

Strings may be handled in arrays, just like numeric data. The array name must have a \$ to indicate that the data is alphanumeric.

```
9040 READ X$(I),Y$(I):NEXT
```

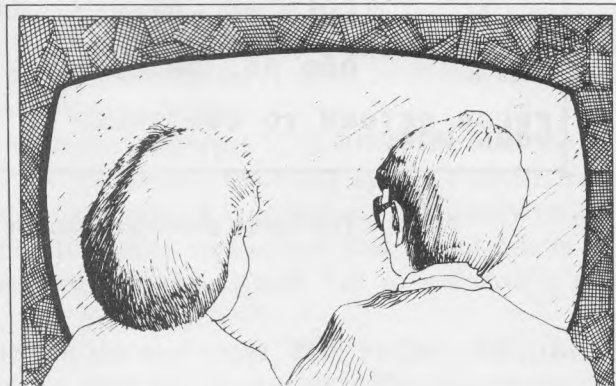
This will READ the strings of DATA starting at line 10000 into two arrays, X\$(I) and Y\$(I).

```
10000 DATA "ODD NO. OF LETTERS","EVEN NO. OF LETTERS"
```

X\$(0) will be set to equal "ODD NO. OF LETTERS" and Y\$(0) will be set equal to "EVEN NO. OF LETTERS". These individual strings, which are the written rules for the game, are then output by referencing them in a PRINT statement.

```
3100 PRINT "{BLUE}I HATE WORDS WITH{GREEN} ":  
      PRINT "          "Y$(R-1):PRINT
```

After printing the 'canned' message "I HATE WORDS WITH", the PRINT Y\$(R-1) statement will output the string referenced by R-1. For example, if R = 1, then R-1 will equal 0, and Y\$(0) contains the string "EVEN NO. OF LETTERS" as shown above, and this is what will be printed.



The figure below displays the five rules presented in Word Detective.

```

WORD-DETECTIVE RULES:
RULE # 1
I LOVE WORDS WITH
      ODD NO. OF LETTERS

RULE # 2
I LOVE WORDS WITH
      5 LETTERS OR FEWER

RULE # 3
I LOVE WORDS WITH
      VOWELS => CONSONANTS

RULE # 4
I LOVE WORDS WITH
      FIRST LETTER < LAST

RULE # 5
I LOVE WORDS WITH
      DOUBLE LETTERS

YOUR WORD:
  
```

```

YOUR WORD: DIFFICULT (B)
YOUR WORD: LONGEST (R)
YOUR WORD: BOON (R)
YOUR WORD: SIMPLE (R)
YOUR WORD: FIRST (M)
YOUR WORD: SECOND (N)
YOUR WORD: GLOUSY (N)
YOUR WORD: GOOD (N)
YOUR WORD: THE RULE IS RULE # 1
I HATE WORDS WITH
      EVEN NO. OF LETTERS
I LOVE WORDS WITH
      ODD NO. OF LETTERS
PRESS RETURN TO CONTINUE: ?
YOUR WORD:
  
```

The figure above is a sample run.

Program Description

At line 20, the program goes to subroutine 9000 to initialize. The initialization procedure consists of the following steps:

Line 9010 DIMensions two string arrays, X\$ and Y\$.

Line 9020 RESTOREs the DATA pointer for the following READ statements so that the READ will start at the first DATA statement, and POKEs location 53281 with the value 1 to set the screen background color to white.

Lines 9030 to 9040 use a FOR...NEXT loop to READ the DATA statements into the string arrays dimensioned above.

Lines 9050 to 9160 are simply a series of PRINT statements that display the operating instructions for the program.

Line 9200 initializes the Random Number Generator (RND) by using the current time in the Commodore 64 built-in timer (TI) as the seed value and then RETURNs the program to the mainline code.

The mainline of the program, lines 100 to 1000, uses the random number generator to select the current 'RULE', accept the word from the user, analyze the word for a number of characteristics, and then evaluate the word relative to the rule.

Line 110 generates a random number and then makes sure it is not the same number as the previous number. If the new RULE number is the same as the old one, then another number is generated until a new RULE is selected.

Line 115 initializes Z\$ to the null string and prints the prompt 'YOUR WORD: ' Line 120 GETs a character into A\$ from the keyboard.

Line 130 goes to line 20000 to exit the program and run the disk menu.

Line 140 ENDs the program, consequently exiting to BASIC.

Line 160 tests if the input character is equal to CHR\$(13), the character string code for a return. If it is, then no character was input and the program goes to line 120 to input a character.

Line 170 sets Z\$ to the input string by adding each input character to itself as they are entered.

Line 180: If the user types 'THE RULE IS', then the GOSUB 3000 will cause the current rule to be displayed. On return from line 3000, the current rule number R is copied to a temporary variable X for the test in line 110, which makes sure that a new rule is selected each time. The program will GOTO 110 to generate a new rule and continue.

Line 185: If the user types 'THE RULES ARE', then the routine starting at line 4000 will display all of the rules on the screen.

Line 200: The program comes here on any regular word. Each of four variables is set to an initial value of 0. The variables are: J for Odd or Even, D for Double letter, V for Vowel counter, and C for Consonant counter.

Line 210 uses the LEFT\$(Z\$,1) function to get the leftmost (First) character of the string Z\$ into the string variable F\$.

Line 220 uses the RIGHT\$(Z\$,1) function to get the rightmost (Last) character of the string Z\$ into the string variable L\$.

Line 230 uses the LEN(Z\$) function to get the length of the string Z\$ into the numeric variable L.

Line 240 uses the INTeger function to get the integer portion of the length L, does a logical AND with the number 1, and if the result is not zero (which will be the case for any ODD number) sets the J variable to 1 to be used later in the program to determine if the string was an ODD or EVEN number of characters.

Line 250 simply tests for a single letter word that obviously could not have a double letter, and skips the next few lines on a single letter.

Line 260 sets up a FOR...NEXT loop to test for a double character. The MID\$(Z\$,K,1) portion of line 270 isolates one character in the word, the MID\$(Z\$,K1,1) isolates the next character in the word, and the IF function tests to see if these two consecutive characters are identical. If they are, it sets the double flag, D, to 1 for later testing; if not, it leaves the flag alone.

Lines 290 to 350 are a FOR...NEXT loop to count Vowels (A E I O U and Y) and Consonants (all non-vowels) in the word. Line 300 sets the V\$ string variable equal to the next character in the word. Lines 310 and 320 test for the vowels and line 340 adds one (1) to the V counter for each vowel found. Line 330 adds one (1) to the C counter for any character that is not a vowel.

All variables that are going to be used for the various RULE tests below now have been calculated:

F\$ is the First character

L\$ is the Last character

J = 1 for an odd number of characters
 = 0 for an even number of characters

L = the number of characters

V = the number of vowels

C = the number of consonants

D = 0 for no double letters
 = 1 for double letters

Line 1000 branches to a subroutine based on the number of the rule in R. It will go to subroutine 1100 on rule 1, 1200 on rule 2, ... , 1500 on rule 5.

Line 1100 tests for odd or even number of characters. If $J = 0$, which means that the word is disliked, it goes to line 2000 to display the word in green; otherwise the word is liked and goes to line 2100 to display the word in red.

Line 1200 tests for the length to be equal to or greater than 5 and, if it is, goes to line 2000 to turn the word green; otherwise it goes to line 2100 to turn the word red.

Line 1300 tests for more consonants than vowels and, if there are, goes to line 2100 to turn the word green.

Line 1400 tests for the first letter of the word to be earlier in the alphabet than the last letter. If so, it goes to line 2100 to turn the word red.

Line 1500 tests for double letters. If there are double letters, it turns the word red.

Line 2000 moves the cursor back one line to the typed word, writes two spaces to advance the cursor and remove the question mark, then changes the color to green.

Line 2010 prints the word (Z\$) in whatever color has been set then changes the color back to blue.

Line 2020 RETURNS, ends the subroutine, and returns to line 1000 and the GOTO 120 statement.

Line 2100 moves the cursor back to the word line and changes the color to red.

Line 2110 goes to line 2010 to complete the output of the red word.

Lines 3000 to 3040 handle the response when the user types 'THE RULE IS'. Line 3010 prints the message "RULE #", prints the current rule number, R, and then goes to subroutine 3100, which prints the RULE. Line 3020 prints an extra carriage return and the message 'PRESS RETURN TO CONTINUE'. Line 3040 waits for a RETURN from the keyboard and then RETURNS to the calling program.

Lines 4000 to 4050 handle the response when the user types 'THE RULES ARE'. Line 4010 PRINTs a clear screen character (represented by the reverse heart character), sets the color to black (represented by the solid black box), then the message 'WORD DETECTIVE RULES:', and then changes the color to blue (represented by the reversed arrow). The next PRINT provides an additional blank line on the display.

Line 4020 uses a FOR...NEXT loop to output each of the five rules. It first PRINTs the color to black (solid box), the message 'RULE —' and then outputs the current rule number, R.

Line 4030 goes to the subroutine at line 3110 to output the "I LOVE..." information, a blank line, and returns.

Line 4040 is the NEXT statement of the FOR...NEXT loop that began in line 4020. On R values of 1 to 4 this statement will go back to

line 4020. On R equals 5 (the last value specified in the FOR loop in line 4020) the program will go to the next sequential line.

Line 4050 goes to line 110 to select a new random number for the next game.

That's all there is to it. Word Detective is a simple game and a simple program. If you study the program and fully understand it, then you are well on your way to understanding BASIC programming on the Commodore 64.

Examples of String Functions

`Y$ = "DIFFICULT"`

The string `Y$` contains the word "DIFFICULT".

`LEN(Y$) = 9`

The `LEN`gth of the string `Y$` is 9.

`LEFT$(Y$,1) = "D"`

The `LEFT`most character of string `Y$` is "D".

`RIGHT$(Y$,1) = "T"`

The `RIGHT`most character of string `Y$` is "T".

`MID$(Y$,3,2) = "FF"`

The `MID`dle two characters starting at position 3 are "FF".

`ASC(Y$) = 68`

The `ASC`ii decimal value of the first character in the `Y$` string, "D", is 68.

`CHR$(73) = "I"`

The `CHa`Racter represented by the `ASCII` decimal value 73 is the letter "I".

Programming Projects

There are many ways you can change Word Detective. Let's look at a few.

Reversing the Rules

To reverse a rule (e.g., to make WORD like EVEN words and hate ODD words), two parts of the program must be changed. First, the test for ODD or EVEN must be reversed. This can be done by changing line 1100.

```
1100 IF J=0 THEN 2000
```

When J equals 0, the word is EVEN and, in the original version, goes to the 'hate word' routine at 2000. To reverse the test simply test for J equals 1. This will then branch to the 'hate word' routine when the word is ODD.

```
1100 IF J=1 THEN 2000
```

Second, the strings that will be displayed for 'THE RULE IS' and 'THE RULES ARE' must be reversed.

```
10000 DATA "ODD NO. OF LETTERS", "EVEN NO. OF LETTERS"
```

should be changed to read

```
10000 DATA "EVEN NO. OF LETTERS", "ODD NO. OF LETTERS"
```

Changing the Rules

Any rule can be eliminated and a new rule put in its place. The new rule can be based on the existing tests ODD/EVEN, LENGTH, VOWELS/CONSONANTS, FIRST/LAST LETTER, and DOUBLE LETTERS, or new tests can be added. Using the existing tests, the FIRST/LAST LETTER can be modified so that the word is 'liked' only when the FIRST letter equals the LAST letter.

```
1400 IF F$<L$ THEN 2100 becomes 1400 IF F$=L$ THEN 2100
```

An entirely new test may be added. For example, if only words with the letter A were to be 'liked', then insert a new test in the loop that goes through the entire string looking for vowels and consonants (lines 300 to 350). Add

```
305 IF V$%="A" THEN A=10
```

Initialize the variable A to 0 by adding :A=0 to the end of the existing line 200.

```
200 J=0:D=0:V=0:C=0:A=0
```

Now replace the Doubles test with the Letter A test. Change

```
1500 IF D=1 THEN 2100 to 1500 IF A=1 THEN 2100
```

Finally, change the strings at 10040 to reflect the new rule:

10040 DATA "LETTER A", "NO LETTER A"

Use your imagination! There are many rules, and you can even combine two rules to make a new one. For example, 'like' ODD letters in words that are seven (7) characters or more in length, 'like' EVEN letters in words that are six (6) characters or fewer in length, and 'hate' the other two possible combinations.

```
10 REM WORD DETECTIVE
20 GOSUB 9000 : REM INIT
100 REM MAIN PROGRAM
110 R=INT(RND(1)*5)+1:IF R=X THEN 110
115 Z$="":PRINT "{BLACK}YOUR WORD: {BLUE}";
120 GET A$:IF A$="" THEN 120
130 IF A$="{F1}" THEN 20000:REM F1 FOR DISK MENU
140 IF A$="{F3}" THEN END:REM F3 FOR BASIC
150 IF A$=CHR$(13) AND LEN(Z$)>0 THEN 180
160 IF A$=CHR$(13) THEN 120
165 IF A$=CHR$(20) AND LEN(Z$)<>0 THEN
    Z$=LEFT$(Z$,LEN(Z$)-1):GOTO 175
170 Z$=Z$+A$
175 PRINT A$;:GOTO 120
180 IF Z$ = "THE RULE IS" THEN GOSUB 3000:X=R:GOTO 110
185 IF Z$ = "THE RULES ARE" THEN GOSUB 4000:GOTO 115
199 REM PROCESS NORMAL WORD
200 J=0:D=0:V=0:C=0
210 F$ = LEFT$(Z$,1)
220 L$ = RIGHT$(Z$,1)
230 L = LEN(Z$)
240 IF INT(L) AND 1 THEN J=1
250 IF L = 1 THEN 290
260 FOR K = 1 TO L
270 IF MID$(Z$,K,1) = MID$(Z$,K+1,1) THEN D = 1
280 NEXT
290 FOR K = 1 TO L
300 V$ = MID$(Z$,K,1)
310 IF V$="A" OR V$="E" OR V$="I" THEN 340
320 IF V$="O" OR V$="U" OR V$="Y" THEN 340
330 C = C+1: GOTO 350
340 V = V+1
350 NEXT
```



```

999 REM SELECT TEST
1000 ON R GOSUB 1100,1200,1300,1400,1500:GOTO 115
1100 IF J=0 THEN 2000
1110 GOTO 2100
1200 IF L=< 5 THEN 2100
1210 GOTO 2000
1300 IF C> V THEN 2000
1310 GOTO 2100
1400 IF F$< L$ THEN 2100
1410 GOTO 2000
1500 IF D=1 THEN 2100
1510 GOTO 2000
1999 REM CHANGE WORD COLOR
2000 PRINT:PRINT "{UP}";"YOUR WORD: "; "{GREEN}";:
    REM GREEN FOR HATE
2010 PRINT Z$"{BLUE}"
2020 RETURN
2100 PRINT:PRINT "{UP}";"YOUR WORD: "; "{RED}";:
    REM RED FOR LOVE
2110 GOTO 2010
3000 REM THE RULE IS
3010 PRINT " {BLACK}RULE #";R:PRINT: GOSUB 3100
3020 PRINT:PRINT "PRESS RETURN TO CONTINUE: ";
3040 INPUT Z$:RETURN
3100 PRINT "{BLUE}I HATE WORDS WITH{GREEN} ":
    PRINT "          "Y$(R-1):PRINT
3110 PRINT "{BLUE}I LOVE WORDS WITH{RED} ":
    PRINT "          "X$(R-1);
3120 PRINT "{BLUE}": PRINT
3130 RETURN
4000 REM THE RULES ARE
4010 PRINT "{CLEAR,BLACK}WORD DETECTIVE RULES:{BLUE}":PRINT
4020 FOR R = 1 TO 5:PRINT "{BLACK}RULE #";R
4030 GOSUB 3110
4040 NEXT
4050 GOTO 110
9000 REM INITIALIZATION
9010 DIM X$(4),Y$(4)
9020 RESTORE:POKE 53281,1
9030 FOR I = 0 TO 4
9040 READ X$(I),Y$(I): NEXT

```

```

9050 PRINT "{CLEAR,BLACK}          WORD DETECTIVE":
    PRINT "{BLUE}"
9060 PRINT "{DOWN}IF I LIKE A WORD,
    I MAKE IT {RED}RED{BLUE}"
9070 PRINT "{DOWN}IF I HATE A WORD,
    I MAKE IT {GREEN}GREEN{BLUE}"
9080 PRINT "{DOWN}FIGURE OUT MY RULE BY TYPING"
9090 PRINT "  IN DIFFERENT WORDS."
9100 PRINT "{DOWN}WHEN YOU HAVE FIGURED OUT THE RULE,"
9110 PRINT "  OR YOU GIVE UP, THEN TYPE:"
9120 PRINT "{DOWN,BLACK}    THE RULE IS{BLUE}  TO
    DISPLAY THE RULE."
9140 PRINT:PRINT "TO DISPLAY ALL OF THE RULES, TYPE:"
9150 PRINT "{DOWN,BLACK}    THE RULES ARE{BLUE}"
9160 PRINT "{DOWN,RED}PRESS F1 FOR DISK MENU, F3 FOR
    BASIC{BLUE,DOWN}"
9200 R = RND(-TI):RETURN
10000 DATA "ODD NO. OF LETTERS","EVEN NO. OF LETTERS"
10010 DATA "5 LETTERS OR FEWER","6 LETTERS OR MORE"
10020 DATA "VOWELS => CONSONANTS","VOWELS < CONSONANTS"
10030 DATA "FIRST LETTER < LAST","FIRST LETTER => LAST"
10040 DATA "DOUBLE LETTERS","NO DOUBLE LETTERS"
20000 REM MENU LOADER
20010 PRINT "{CLEAR}LOAD "+CHR$(34)+"0:MENU"+CHR$(34)+"8":
    PRINT "{DOWN4}RUN"
20020 POKE 631,19: POKE 632,13: POKE 633,13: POKE 198,3: END

```

Word Detective Variable Usage

I,K	Loop counters
R	Random number rule
Z\$	Input variable
F\$	First letter of Z\$
L\$	Last letter of Z\$
V\$	Middle letter of Z\$
C	Number of consonants
V	Number of vowels
L	Length of word
J	Odd/even flag
D	Double letter flag

The Answer Machine

The Intelligent Computer

```
THE ANSWER MACHINE  
I KNOW EVERYTHING !  
YOU ASK QUESTIONS ABOUT ANYTHING,  
AND I WILL ANSWER WITH A  
YES, NO, OR MAYBE.  
TYPE F3 TO EXIT TO THE DISK MENU.  
TYPE F3 TO EXIT BASIC.  
END EACH QUESTION WITH A QUESTION MARK.  
?
```

The Answer Machine allows you to have a discussion with your Commodore 64, or to have your Commodore 64 tell you a story or answer your questions. All you have to do is type in a question that can be answered with a YES or NO; The Answer Machine will do the rest. To enter a question simply type your question, ending it with a question mark ('?'). The Answer Machine will provide you with an answer immediately. It may not always be right, but it is always fun!

Program Description

The Answer Machine may be one of the easiest programs you ever try! The routine uses only eight BASIC statements and five BASIC functions. The BASIC statements are END, GET, GOTO, IF...THEN, LET, ON...GOSUB, PRINT, and RETURN. The five BASIC functions are LEN, LEFT\$, ASC, INT, and RND.

Line 0 is a REMark statement.

Line 1 POKes location 53281 with the value 1 to set the screen background color to white so that the messages will all be visible. Line 1 also POKes location 53280 with the value 6 to make the boarder blue.

Lines 20 to 80 simply PRINT the instructions for The Answer Machine.

Line 100 LETs the string variable S\$ be equal to the NULL string, LETs the numeric variable T be equal to 0, and then PRINTs a question mark. Where is the LET in line 100? It is implied. S\$ = 0 is equivalent to LET S\$ = 0. The LET does not have to be specified and is normally omitted.

Line 110 attempts to GET a character from the keyboard into the string variable A\$. IF there is no character to get, THEN line 110 is executed again until there is a character in A\$.

Line 120 does a GOTO 20000 to call the disk menu, if the user hits the F1 key.

Line 130 ENDS the program and exits to basic if F3 is hit.

Line 140 PRINTs the character in A\$ and then adds the ASC value of the current character to a total counter T. A\$ is then added to the current string variable S\$. Then a test is made to see if A\$ is a question mark. If it is not then the program goes to line 110 to enter additional characters; otherwise it continues on to line 150.

Line 150 ends the program if S\$, the string input, is only a question mark.

Line 160 PRINTs a color change to red for the message output.

Line 170 converts T, which has the total of the ASCII values of all of the characters in the string S\$, into a number in the range 1 to 10.

Line 180 uses the ON...GOSUB to branch to a YES, NO, or MAYBE. On return from the subroutine the program does a GOTO 100 to start the next question.

Line 200 PRINTs the YES, 210 PRINTs the NO, and 220 generates a random number to select among 10 choices for the MAYBE.

Line 230 uses the ON...GOTO to branch to one of 10 possible responses.

Lines 300 to 390 are the PRINTs for each of the responses. Each ends in a RETURN, which will return to line 180 at the GOTO 100.

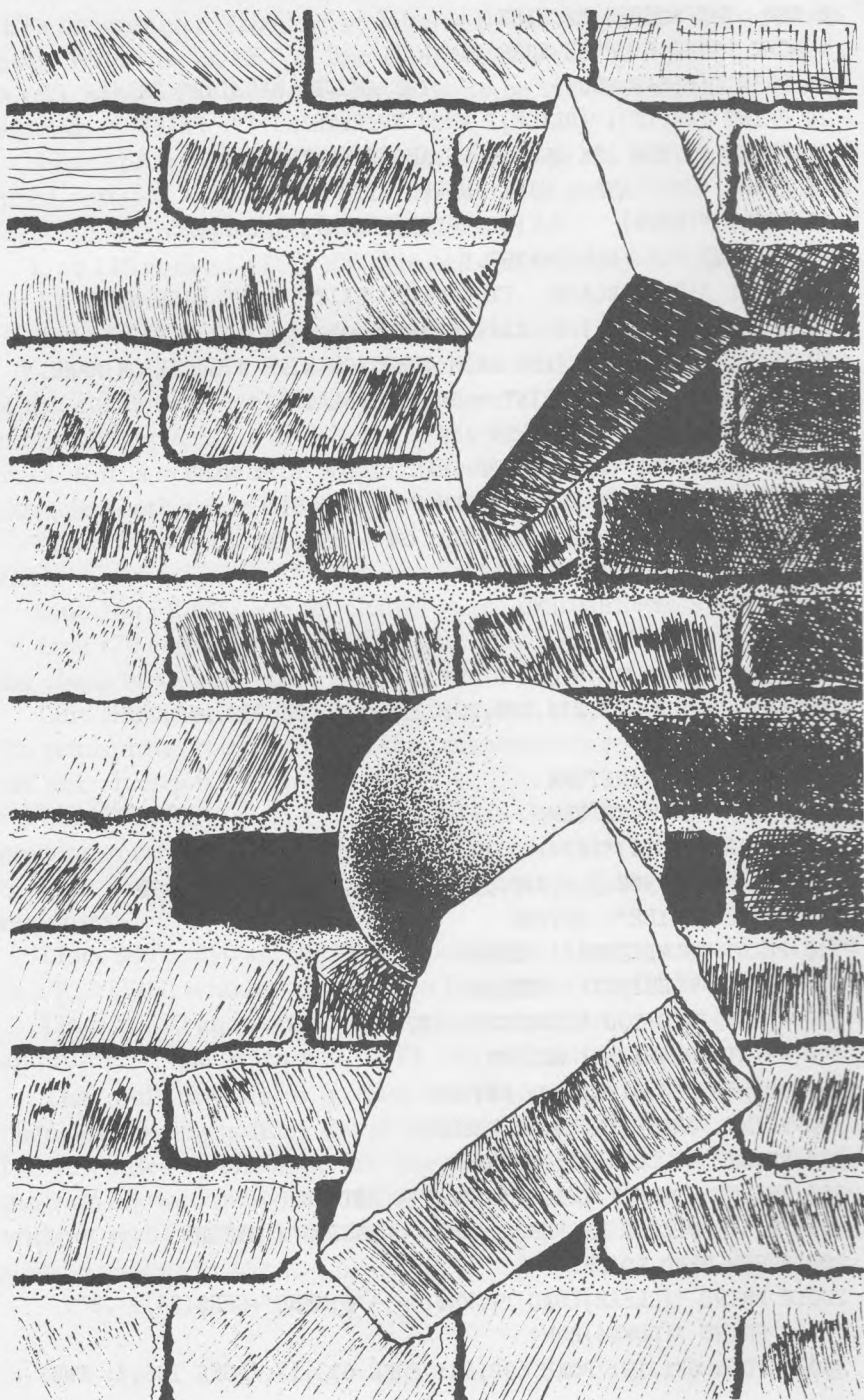
Lines 20010 and 20020 load the menu into the computer from disk and run it so that when you hit F1 you will exit to the disk menu.

That's all there is to it. You can change the responses easily by changing the text in lines 300 to 390, or change the number of YES and NO responses by changing the line numbers in line 180. This simple program shows how your computer can be made to *appear* intelligent without really having a whole lot of 'smarts'.

```

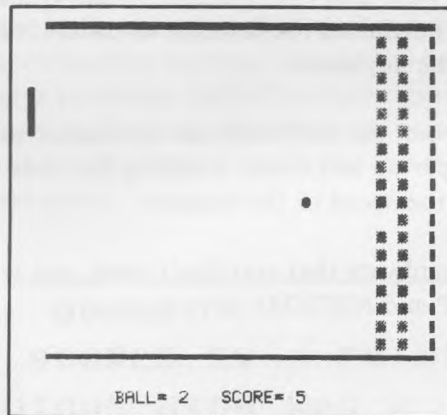
Ø REM THE ANSWER MACHINE
1 POKE 5328Ø,6: POKE 53281,1
1Ø PRINT "{CLEAR,RED} THE ANSWER MACHINE":PRINT
2Ø PRINT TAB(1Ø);"{BLACK}I KNOW EVERYTHING !": PRINT
3Ø PRINT " YOU ASK QUESTIONS ABOUT ANYTHING,"
4Ø PRINT " AND I WILL ANSWER WITH A"
5Ø PRINT "{DOWN} {BLUE}YES{BLACK}, {BLUE}NO
{BLACK}, OR {BLUE}MAYBE."
6Ø PRINT "{DOWN,BLACK} TYPE F1 TO EXIT TO THE DISK MENU."
7Ø PRINT " TYPE F3 TO EXIT BASIC."
8Ø PRINT "{DOWN,BLUE}END EACH QUESTION WITH A QUESTION MARK."
1ØØ S$ = "": T = Ø: PRINT "{BLACK}?";
11Ø GET A$:IF A$ = "" THEN 11Ø
12Ø IF A$="{F1}" THEN 2ØØØØ: REM F1 FOR DISK MENU
13Ø IF A$="{F3}" THEN END: REM F3 FOR BASIC
14Ø PRINT A$;: T = T + ASC(A$): S$ = S$ + A$:
IF A$ < > "?" THEN 11Ø
15Ø IF S$ = "?" THEN END
16Ø PRINT "{RED}"
17Ø T =T -INT(T/1Ø)*1Ø+1
18Ø ON T GOSUB 2ØØ,21Ø,22Ø,2ØØ,21Ø,22Ø,2ØØ,22Ø,2ØØ,21Ø:
GOTO 1ØØ
2ØØ PRINT "YES":RETURN
21Ø PRINT "NO": RETURN
22Ø R =INT(RND(Ø)*1Ø)+1
23Ø ON R GOTO 3ØØ,31Ø,32Ø,33Ø,34Ø,35Ø,36Ø,37Ø,38Ø,39Ø
3ØØ PRINT "MAYBE": RETURN
31Ø PRINT "SOMETIMES": RETURN
32Ø PRINT "POSSIBLY": RETURN
33Ø PRINT "ARE YOU KIDDING?": RETURN
34Ø PRINT "PERHAPS": RETURN
35Ø PRINT "NOT LIKELY!": RETURN
36Ø PRINT "INDUBITABLY!": RETURN
37Ø PRINT "OF COURSE": RETURN
38Ø PRINT "I CAN'T TELL YOU THAT": RETURN
39Ø PRINT "PLEASE RE-PHASE THE QUESTION": RETURN
2ØØØØ REM MENU LOADER
2ØØ1Ø PRINT "{CLEAR}LOAD "+CHR$(34)+"Ø:MENU"+CHR$(34)+"",8":
PRINT "{DOWN4}RUN"
2ØØ2Ø POKE 631,19: POKE 632,13: POKE 633,13: POKE 198,3: END

```



Breakup

An Animated Game



Get out your paddles or joystick and get ready to hit the ball with your bumper. By knocking down a wall of bricks, you can now show all your friends how dexterous you are.

Breakup is a simple animation display game that presents the principles of PEEKing and POKEing the screen memory to plot and erase characters on the screen, and to test for collisions. It includes a "ball" that moves around the screen, rebounds from struck objects, and knocks out blocks in the wall of bricks. It also includes a player-controlled "bumper" to keep the ball from going out-of-bounds and being lost, a defined playing field with three walls from which to bounce the ball, and two rows of blocks to be knocked off by your bouncing ball.

The game keeps score; 5 for each block and 1 for each bounce off the bumper. The previous highest score during each session is kept and displayed at the end of the game. More than 170 points qualifies you for a bonus of 50 points, new blocks, and an extra ball. You can choose to play with a joystick, paddles, or keyboard. If you choose the keyboard, the cursor keys F1 and F7 move the bumper up and down. If you hold down the shift key at the same time, it increases the bumper movement speed.

Operating Instructions

1. Execute BREAKUP from the disk menu.
2. First you are asked whether you have a joystick, paddles, or will play from the keyboard or if you wish to quit to BASIC or to the disk menu. Choose the correct response -J, P, K, Q or M.
3. If you choose K, P, or J then the program will display the playing field, the brick wall, and your bumper. When you are ready to start play, press the button on the joystick or paddle, or the space bar if you are using the keyboard.
4. If you have chosen the keyboard use the cursor keys F1 and F7 to move the bumper up and down. Holding the shift key at the same time increases the speed of the bumper.
5. If you specify hardware that you don't have, you will have to press the RUN/STOP and RESTORE keys to recover.

**C64 BREAKUP
KEYBOARD, PADDLE
JOYSTICK, BASIC OR MENU**

**SPACE BAR TO START
SHIFT TO INCREASE PADDLE SPEED
F1 FOR DOWN
F7 FOR UP
HIT ANY KEY TO CONTINUE**

The Program

The ball starts from a random position on the left side of the screen and travels to the right, hitting a block. This causes the block to disappear, adds five to your score, and the ball rebounds toward the left. Here is the challenge: You must position the bumper to keep the ball from travelling out-of-bounds and off the screen, thereby losing the ball. If you are successful, the ball will rebound back toward the blocks. If you miss the ball, the game freezes so that you can see how close (or how far) you were from the ball when you missed it. You are allowed three balls (four if you get the bonus), so try not to miss. You will notice that the angle and speed of the ball increase the closer you hit the ball to the ends of the bumper. Hitting the ball near the center of the bumper helps to restore the ball to the original direction.

The program stores the high score from game to game until the computer is turned off. Challenge your friends to a match and see who can get the highest score.

```
GAME OVER!  
YOU SCORED 27    POINTS!  
PREVIOUS HIGH WAS 0  
PRESS RETURN TO CONTINUE
```

To create animation on a computer screen, we need to know how the computer displays information on the screen and how we can change that information. The television is constantly updating the information being displayed, about 30 times per second. A computer that uses memory mapped video stores the screen information in RAM (Random Access Memory) so that the video display routine can have this information available at all times. In the C64, the start of the RAM display memory is normally at location 1024. To determine the memory address of any location on the screen, we can use the following function:

$$S0 + \text{INT}(X) + \text{COL} * \text{INT}(Y)$$

S0 is the starting address of the upper left corner of the display (normally 1024), X and Y are the Cartesian coordinates of the location, starting at 0,0 in the upper left corner of the screen, and COL is the number of columns on the screen (normally 40).

Appendix F of the *Commodore User's Guide* and Appendix B of the *C64 Programmer's Reference Guide* contain a table of the codes to be POKEd in screen memory to obtain specific characters. As an example, type the following mini program into your C64:

(First, type NEW to erase any current program from memory.)

```
10 S0 = 1024:C0 = 55296:COL = 40
20 X = 0:Y = 0
30 CH = 81:CL = 1
40 OF = INT(X)+COL*(INT(Y))
50 POKE S0+OF,CH
60 POKE C0+OF,CL
70 END
```

RUN the program and a small white ball should appear on your screen in the upper left corner. If you change the value of X, the location of the character would move across the screen from the left edge at 0 to the right edge at 39. Changing Y would move the character down the screen from the top at 0 to the bottom at 24. Changing CH changes the character displayed. 81 is the character code for a 'ball'. Other characters are: 83 for a 'heart', 01 for the letter 'A', 95 for a 'Pi' character, and so on. Changing CL changes the color of the character being printed. The value for CL is, for the main colors, the value 1 less than the number code on the keyboard color keys on the top row. See the color table below.

How do you erase a character? If you look up 'space' in the screen code chart you will find it has the value 32. POKE that number into the same location as you originally plotted the ball. This effectively erases whatever is on the screen position. Location 53280 controls the border color on the screen and location 53281 controls the background color. POKEing a number into these locations will change the border and background colors. The main colors are:

0 for black, 1 for white, 2 for red, 3 for cyan,
4 for purple, 5 for green, 6 for blue and 7 for yellow.

All of the color codes are listed in Appendix O of the *Commodore User's Guide* and Appendix G of the *C64 Programmer's Reference Guide*.

Paddles and Joysticks

Using paddles or a joystick helps to make the game more interesting and easier to play, but programming them is a little obscure, especially for the novice BASIC programmer. If you don't understand the following information on first reading, look at the BREAKUP program and see how it is done, then reread this section. Also, the *Commodore 64 Programmer's Reference Guide* has additional information of value on this topic.

There are five switches on a joystick which are connected to a piece of hardware in the C64 called a CIA (Complex Interface Adapter). Four of them are controlled by the joystick and one is for the fire button. You can PEEK the locations of the CIA to determine which joystick switches are closed. The variable values are 1 when a switch is not pressed and 0 when it is pressed.

The following lines in the program read the joystick UP and DOWN positions and set variables Q0 (up) and Q1 (down) accordingly:

```
2040 Q0=-(PEEK(PA) AND 1)=0:Q1=((PEEK(PA) AND 2)=0)
2050 IF Q0<>0 THEN S=-3:GOTO 2130
2060 IF Q1<>0 THEN S=3:GOTO 2130
2070 RETURN
```

where PA is the address of the CIA data register (56321), and S is set to -3 for fast upward movement and set to 3 for fast downward movement. Since the left and right joystick movements are not used by this program, they are not tested. Switch 0 is North, switch 1 is South, switch 2 is West, switch 3 is East, and switch 4 is the fire button. The fire button is read and tested in lines 4100 and 4110. It is read in line 4100 as BUT=PEEK(PA), just like the PEEKs in the example above. It is tested by the IF BUT<>239 portion of line 4110 which will be true if the fire button is pressed.

The C64 paddles are read in a different manner. A value representing the current position of the paddle is available by PEEKing location 54297. Since this interface is actually an analog-to-digital convertor, there will be some variation in the conversion value even when the paddle is not being moved. This is handled by checking for a change in the paddle value from its prior value. This is handled in lines 2000 through 2030.

```

2000 TS=INT(PEEK(54297)/17)+2
2010 IF TS< A+2 AND TS> A-2 THEN RETURN
2020 B=SP:GOSUB 1000
2030 A=TS:GOTO 2150

```

Line 2000 PEEKs the converter value that represents the current position of the paddle controller. It divides it by 17, takes the integer value of the result and adds 2 to convert the raw paddle position into a screen relative position. Line 2010 tests to see if the new value is two or more positions greater than or less than the prior position. If not, then the paddle has not moved far enough to cause a change by the program, and a RETURN is made to the main routine. If the change is sufficient, then in line 2030 the variable B, which is used to determine what character will be written on the screen, is set to be the space character. This will cause the old display paddle to be erased so that a new one may be drawn. Line 2030 simply sets the test value A to be equal to the current value of the paddle position, TS.

Program Description

Lines 1000-1020 (the subroutine to plot the bumper) calculate the screen position of the bumper and then POKE the locations either with a bar (B = 245) or a space (B = 32), depending on whether the subroutine is called to plot or erase the bumper. Lines 3070 and 3090 erase and draw the ball, respectively. Line 3080 looks at the screen location to see if a block has been hit. If so, the subroutine at line 5000 is called to update the score and put "english" on the ball if it was called because of bumper contact.

Lines 4050-4060 plot the block walls. If you remove lines 4050 and 4060, the game turns into "PONG" with the player scoring one point for each rebound from the bumper. Line 4140 picks a random spot along the left-hand side of the screen for the new ball to appear.

Line 3050 tests for out-of-bounds in the Y range and line 3100 tests for out-of-bounds in the X range. If the ball is out-of-bounds it reverses the ball direction. Line 3120 tests for hitting the bumper. If it has hit the bumper then the 5000 subroutine is called to update the score and put english on the ball, as previously mentioned.

As a whole, the routine moves rather slowly due to the limitations of the BASIC language. We employed several techniques to keep the program moving as fast as possible. Variables used most often (those concerning the ball and bumper) were initialized at the beginning of the program so that they would be placed in the beginning of the

variable storage area. The screen update subroutines were placed near the beginning of the program so that the interpreter would spend less time looking for the line numbers of the subroutines. In line 2010 (the bumper moving routine) a return is placed in the first line so that when the bumper is not to be moved, the program doesn't spend time replotting an unmoved bumper. That is why the ball moves much faster when you are not changing the bumper location on the screen.

The screen POKes can be any character (see the screen code chart in the programming manual). Breakup uses 32 (the space) to erase the screen. The ball is 81, the blocks are 102, the bumper is 245. These could be changed easily to different shapes. The scoring subroutine at line 5000 can be changed to make the score advance either faster or slower. Also, the scoring could be changed to increase the bonuses or provide snappy comments at the conclusion of the game. The block-plotting routine at line 4050 could be modified to increase or decrease the number of columns of blocks. If you plot them nearer to the bumper, then the 15 in line 3080 will need to be decreased commensurately.

```

10 REM BREAKUP
20 X=0:Y=X:X1=X:Y1=X:S0=X:S1=X:S2=X:S3=X:DY=X:DX=X
30 CS=53280:POKE CS,1:POKE CS+1,1:GOTO 4000
1000 S1=S0+COL*A:S2=S0+COL*(A+1):S3=S0+COL*(A+2)
1010 POKE S1,B:POKE S2,B:POKE S3,B
1020 RETURN
2000 TS=INT(PEEK(54297)/17)+2
2010 IF TS<A+2 AND TS>A-2 THEN RETURN
2020 B=SP:GOSUB 1000
2030 A=TS:GOTO 2150
2040 Q0=-((PEEK(PA) AND 1)=0):Q1=((PEEK(PA) AND 2)=0)
2050 IF Q0<>0 THEN S=-3:GOTO 2130
2060 IF Q1<>0 THEN S=3:GOTO 2130
2070 RETURN
2080 GET A$:IF A$="" THEN RETURN
2090 S=0:IF A$="{F1}" THEN S=-1
2100 IF A$="{F7}" THEN S=1
2110 IF A$="{F2}" THEN S=-3
2120 IF A$="{F8}" THEN S=3
2130 B=SP:GOSUB 1000
2140 A=A+S
2150 IF A<2 THEN A=2

```

```

2160 IF A> 17 THEN A=17
2170 B=D:GOSUB 1000
2180 RETURN
3000 OK=1
3010 ON PDL GOSUB 2000,2040,2080
3020 X1=X:Y1=Y
3030 Y=Y+DY:IF Y> 19 THEN Y=19
3040 IF Y< 2 THEN Y=2
3050 IF INT(Y)=2 OR INT(Y)=19 THEN DY=-DY:POKE V3,17
3060 X=X+DX:IF X> 37 THEN X=37
3070 POKE S0+INT(X1)+COL*INT(Y1),SP
3080 IF X> 32 AND PEEK(S0+INT(X)+COL*INT(Y))=102
    THEN POKE V1,17:Q=5:GOSUB 5000
3090 POKE S0+INT(X)+COL*INT(Y),BL
3100 IF X> 1 AND X< 37 THEN 3150
3110 IF INT(X)=37 THEN DX=-DX:POKE V3,17:GOTO 3150
3120 IF X=1 THEN GOSUB 3200
3130 IF FLAG=1 THEN FLAG=0:DX=-DX:POKE V3,17:
    GOSUB 5000:GOTO 3150
3140 BALL=BALL-1:OK=0:POKE S0+INT(X1)+COL*INT(Y1),SP:RETURN
3150 POKE V1,0:POKE V3,0:RETURN
3200 IF (Y>=A OR INT(Y)=A-1 AND DY> 0) AND (Y<=A+2OR
    INT(Y)=A+3 AND DY< 0) THEN FLAG=1
3210 RETURN
4000 V1=54276:V3=54283:VOL=54296:PA=56321:COL=40:V2=54290
4005 POKE 54273,34:POKE 54272,75:POKE 54277,16:POKE 54278,64
4006 POKE 54280,72:POKE 54279,169:POKE 54284,16:POKE 54285,64
4010 POKE V1,0:POKE V2,0:POKE V3,0
4020 S0=1024:GOSUB 6000
4030 A=2:B=D:GOSUB 1000
4040 BALL=3:Q=1
4050 FOR X=33 TO 35 STEP 2:FOR Y=2 TO 19
4060 POKE S0+INT(X)+COL*INT(Y),102:NEXT:NEXT
4070 IF BNUS THEN 4090
4080 SCRE=-1:GOSUB 5000
4090 ON PDL GOTO 4100,4100,4130
4100 BUT=PEEK(PA)
4110 IF BUT<> 239 AND BUT<> 251 THEN 4100
4120 GOTO 4140
4130 GET A$:IF A$<> " " THEN 4130
4140 X=2:Y=INT(RND(1)*14)+2:DX=1:DY=1
4150 GOSUB 3000
4160 IF OK THEN 4150

```

```

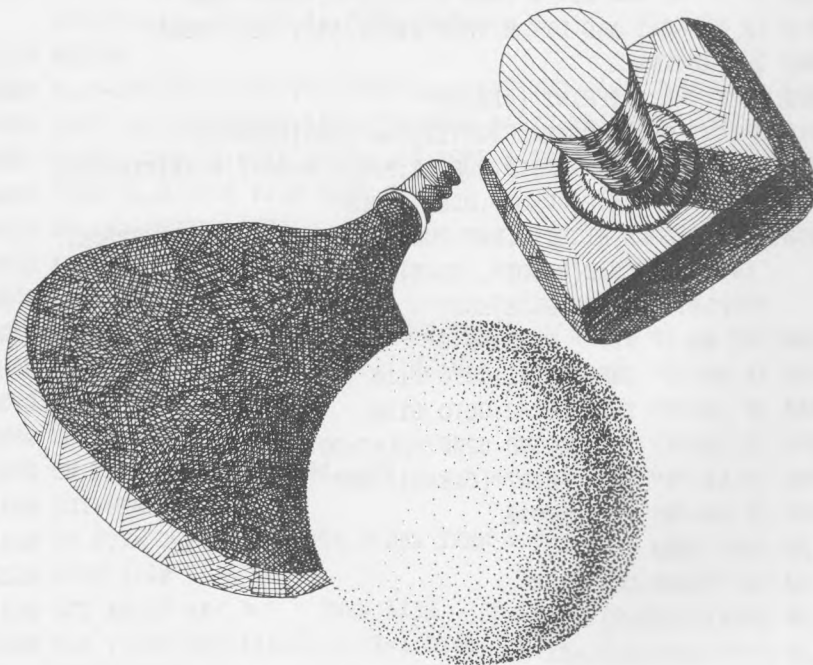
4170 POKE V3,0
4180 ON PDL GOTO 4190,4190,4220
4190 BUT=PEEK(PA)
4200 IF BUT<>239 AND BUT<>251 THEN 4190
4210 GOTO 4230
4220 GET A$:IF A$<>" " THEN 4220
4230 POKE S0+INT(X)+COL*INT(Y),SP
4240 IF BALL THEN SCRE=SCRE-1:GOSUB 5000:GOTO 4140
4250 PRINT"{CLEAR}":PRINT"{BLUE,DOWN5}GAME OVER!"
4260 PRINT"{DOWN}YOU SCORED{RED}"SCRE"{BLUE} POINTS!"
4280 A$="":PRINT"{DOWN}PREVIOUS HIGH WAS{GREEN}"HI"{BLUE}"
4290 PRINT:PRINT"{DOWN}PRESS RETURN TO CONTINUE":
      IF SCRE>HI THEN HI=SCRE
4300 INPUT A$
4310 GOSUB 6090
4320 GOTO 4030
5000 SCRE=SCRE+Q:PRINT V$"{PURPLE}BALL={GREEN}"BALL"
      {BLACK} SCORE={RED}"SCRE;
5010 IF NOT BNUS AND SCRE>170 THEN SCRE=SCRE+50:
      BALL=BALL+1:BNUS=-1:POKE V1,16:GOTO 4050
5020 IF Q=5 THEN DX=-DX:GOTO 5080
5030 IF Y<=A AND DY>0 THEN DY=-1.5*DY:GOTO 5080
5040 IF Y>=A+1 AND DY<0 THEN DY=-1.5*DY:GOTO 5080
5080 Q=1:RETURN
6000 POKE VOL,15:PRINT"{CLEAR}"
6010 PRINT"{GREEN,DOWN2,RIGHT12}C64 {RED}BREAKUP"
6020 PRINT"{DOWN5,RIGHT10,PURPLE,RVS}K{RVSOFF,BLUE}EYBOARD,
      {RVS,PURPLE}P{RVSOFF,BLUE}ADDLE"
6030 PRINT"{DOWN,RIGHT10,RVS,PURPLE}J{RVSOFF,BLUE}OYSTICK,
      {RVS,PURPLE}B{RVSOFF,BLUE}ASIC OR {RVS,
      PURPLE}M{RVSOFF,BLUE}ENU"
6040 GET A$:IF A$="" THEN 6040
6050 IF A$="P" THEN PDL=1:GOTO 6110
6060 IF A$="J" THEN PDL=2:GOTO 6110
6070 IF A$="K" THEN GOSUB 6200:PDL=3:GOTO 6110
6080 IF A$="B" THEN PRINT"{CLEAR}":END
6090 IF A$="M" THEN 20010
6100 GOTO 6000
6110 V$="{HOME,DOWN2}"
6120 PRINT"{CLEAR}"
6130 D=245:SP=32:BL=81
6140 FOR I=55296 TO 55296+1000:POKE I,6:NEXT

```

```

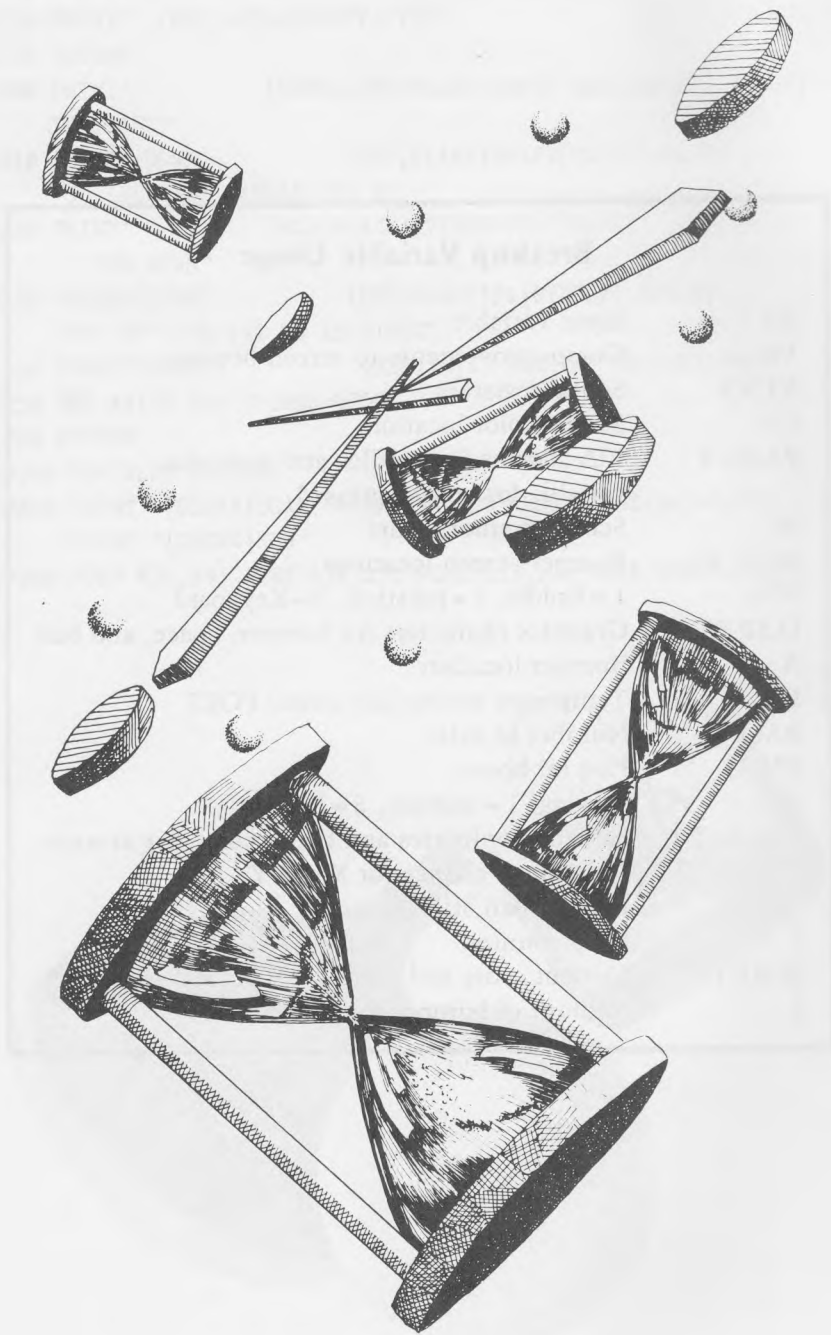
6150 PRINT "{YELLOW}  {=I36,=F}"
6160 FOR I=1 TO 18:PRINT TAB(38);" {=K} ":NEXT
6170 PRINT "  {RVS,=I36,RVSOFF,=V}"
6180 RETURN
6200 PRINT "          {DOWN2,RVS,BLACK}SPACE BAR{RVSOFF} {BLUE}
      TO START"
6210 PRINT "{DOWN}          {RVS,BLACK}SHIFT{RVSOFF,BLUE}
      TO INCREASE PADDLE SPE D"
6220 PRINT "          {RVS,BLACK}F7{RVSOFF} {BLUE}
      FOR DOWN"
6230 PRINT "{DOWN}          {RVS,BLACK}F1{RVSOFF} {BLUE}
      FOR UP":FOR I=1 TO 15 0:NEXT
6240 PRINT "{DOWN2}          {RED,↑H}IT ANY KEY TO CONTINUE"
6250 GET A$:IF A$="" THEN 6250
6260 RETURN
200000 REM MENU LOADER
20010 PRINT "{CLEAR}LOAD "+CHR$(34)+"0:MENU"+CHR$(34)+"",8":
      PRINT "{DOWN4}RUN"
20020 POKE 631,19: POKE 632,13: POKE 633,13: POKE 198,3: END

```



Breakup Variable Usage

A\$	Input variable
V\$	Cursor movements to screen bottom
V1,V3	Sound registers
CS	Screen color location
PA,BUT	VIA address for paddles and joysticks, variable for button pressed
S0	Screen memory start
S1,S2,S3	Bumper screen locations
PDL	1 = Paddle, 2 = Joystick, 3 = Keyboard
D,SP,BL	Graphics characters for bumper, space, and ball
A	Bumper location
B	Temporary storage for screen POKE
BALL	Number of balls
BNUS	Flag for bonus
Q	Scoring, 1 = bumper, 5 = brick
X,Y,X1,Y1	Screen coordinates and their temporary storage
DX,DY	Amount of change for X and Y
OK	Flag for ball still on screen
I	Loop counter
SCRE,HI	Current score and storage for highest score
S	Amount of bumper movement



C64 Clock

A Digital Time Display



The CLOCK program displays the time in large seven segment digits on your video screen. Features include AM/PM indication, seconds display, set time, and set alarm. Line changes and additions are given to add user entered seconds, to add a stop watch, to add a chime and to improve the appearance of the digits.

Operating Instructions

1. Execute the CLOCK program from the *Mastering Your C64* disk.
2. Set the time. You will be asked to enter an hour from 1 to 12. Type in a number and hit return. If you enter an illegal time, such as 13 or -1, then the program will wait until you enter a proper time. Do the same thing for minutes. Next enter A for AM or P for PM.
3. The time that you just entered will be displayed on the screen. When you wish the time to start hit any key. In order to get accurate time you should enter the time a minute ahead and wait for that time to come, then start the clock.
4. The clock will continue to display the correct time until you stop the program or turn off the computer.
5. To change the time or set the alarm, hit any key except 'A' or RUN/STOP. You will be presented with a short menu. Select 'A' to set the alarm.
6. The alarm is set the same way as the time. The alarm time will be displayed on the screen. Press RETURN to get back to the clock. Notice that the clock has maintained the correct time even while you were setting the alarm.
7. The alarm time is displayed on the top of the screen in red.
8. When the alarm goes off it will beep about once a second for a minute. You can turn off the alarm by hitting the 'A' key, which acts as a toggle to set the alarm or to turn it off.

C64 Clock Information

Take a look at a digital clock or watch. You will see that the numbers are composed of seven linear segments. Each of the ten digits is produced by lighting the proper segments in this seven-segment display. Figure 1 shows how each of the ten digits is produced using only these seven segments.

C64 CLOCK simulates the operation and appearance of a digital clock. When you run the program, you will see that the horizontal segments consist of two reversed spaces and the vertical segments consist of three reversed spaces. Each of the segments is the result of printing a string of characters on the screen. What characters are included in these strings? The spaces and the reverse-on and reverse-off characters are obviously included. To produce a vertical segment we must print three spaces in a vertical line. Between spaces the cursor must be moved back and down. This is accomplished by including cursor control characters in the string (*Editor's note:* Cursor controls appear as reversed field characters (clear screen is a reversed heart, cursor-down a reversed 'Q'). The listings in this book use bracketed abbreviations for these special functions. Clear screen is {CLEAR} and cursor-down is {DOWN}. See *How to Read Program Listings* on page 7 for a complete list.) When a cursor control character is printed, the corresponding cursor movement is executed.

Each of the strings for the two kinds of segments has a corresponding string that does not include the [RVS] and [OFF] characters. These are used to turn a segment off when the current digit does not include it. Each space removes the reversed space, making the segment appear to turn off. The four segment strings are defined as H\$(0), H\$(1), V\$(0), and V\$(1). The zeros designate 'off' segment strings. Each string is shown in figure 2, along with a graphic representation of how it is printed on the screen. An arrow indicates a cursor movement, with the arrow's head showing the cursor position at the end of each move. An open circle indicates where the cursor begins and a filled circle indicates where the cursor is positioned at the end.



Figure 1. How the ten digits look in seven segments.

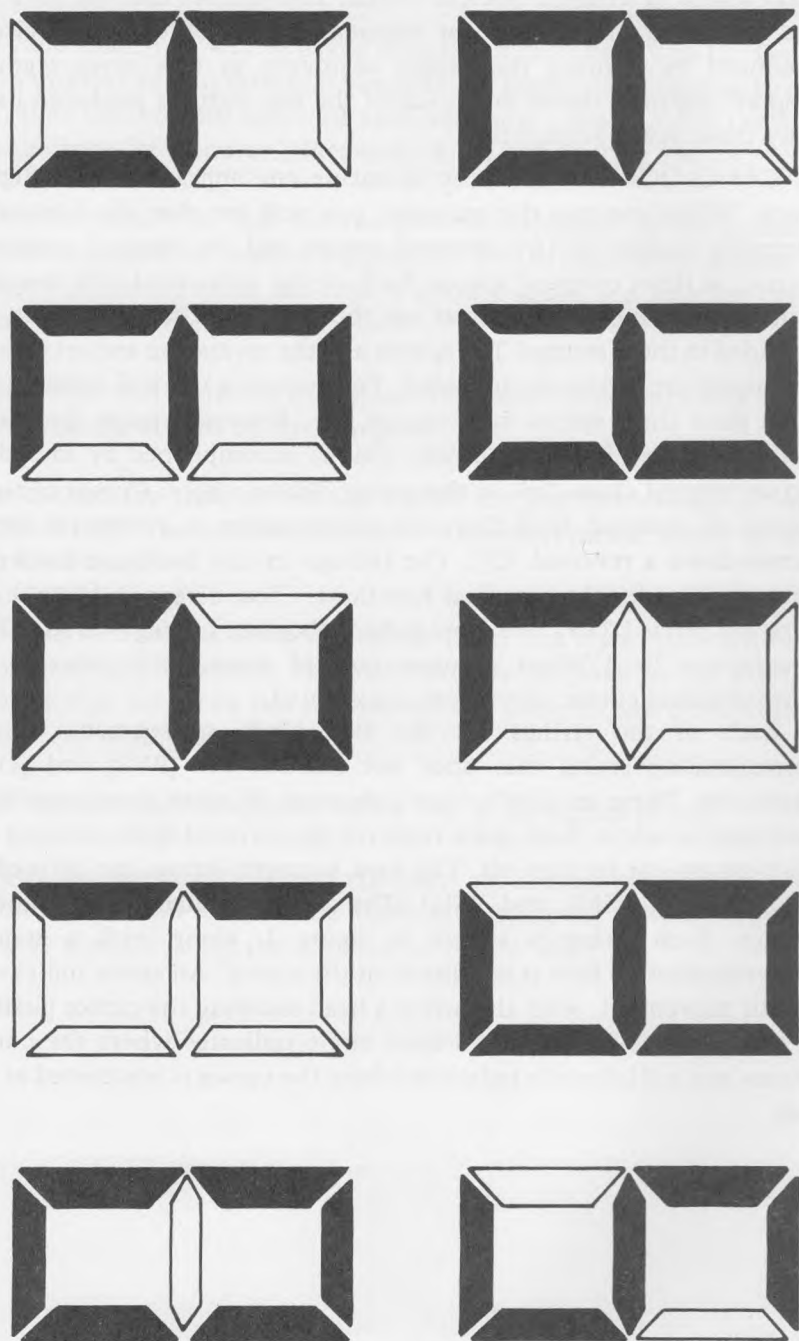
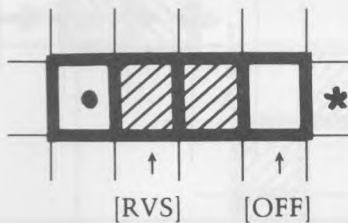
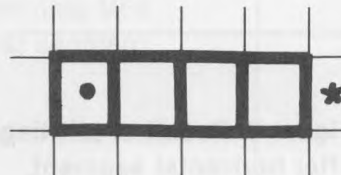


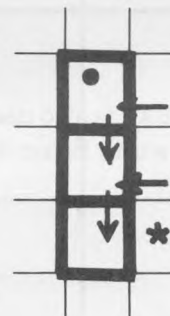
Figure 2: H\$(0),H\$(1),V\$(0),V\$(1) produce the on and off segments on the screen.

- Cursor starts here
- * Cursor ends here
- Cursor movement

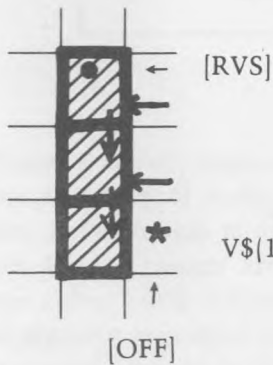
H\$(0) = " "



H\$(1) = " [RVS] [OFF] "



V\$(0) = " [CL][CD] [CL][CD] "

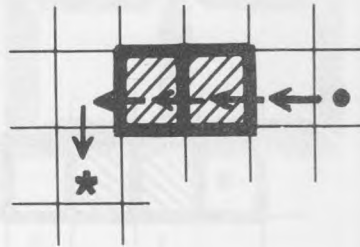


V\$(1) = "[RVS] [CL][CD] [CL][CD] [OFF]"

There are unseen strings other than the blank segments. These consist entirely of cursor movement characters. Two of these, BK\$ and UP\$, are used to move the cursor across the digit for the next segment. BK\$ moves the cursor from the end of a horizontal segment back to the position under the first space of the horizontal segment. We are now ready to print a vertical segment. BK\$ is shown with its printed results in figure 3.

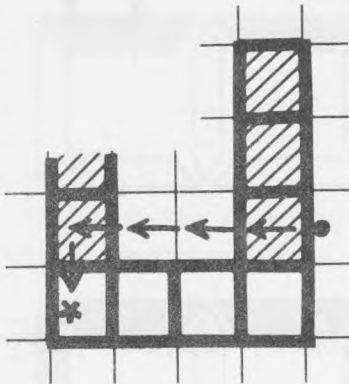
Figure 3: Result of printing BK\$ after horizontal segment.

BK\$ = "[CL][CL][CL][CL][CD]"



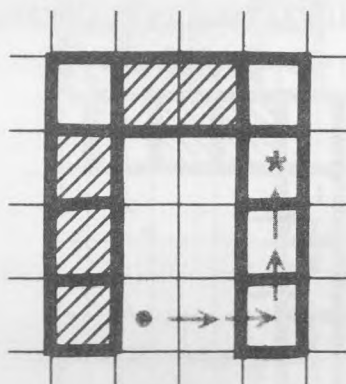
BK\$ is also used to ready the cursor to print a horizontal segment as shown in figure 4.

Figure 4: Result of printing BK\$ after right-hand vertical segment.



UP\$ is used to move the cursor from the bottom of a left-hand vertical segment to the top of a right one. Figure 5 shows UP\$ and the result of printing it after a vertical segment.

Figure 5: Result of printing UP\$ after left-hand vertical segment.



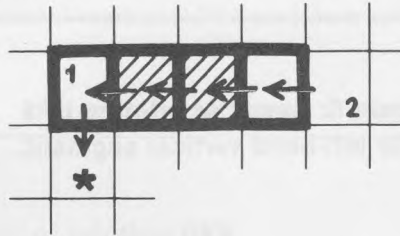
UP\$ = "[CR][CR][CU][CU]"

Figure 6 shows, segment by segment, the process of printing the proper sequence of strings to produce a seven-segment '2' on the screen. Actually this is done in lines 1510-1520 of C64 Clock. The values for the integer array elements (S%(T,1), S%(T,2), etc.) are always either 1 or 0. A blank segment is printed if this value is 0, and a filled segment is printed if it is 1. The process of coding and decoding the segment patterns is described later. In the figure, 1's and 0's are shown for the H\$() and V\$() subscripts as if they had been looked up in the array already.

Figure 6: Producing a seven-segment '2'.

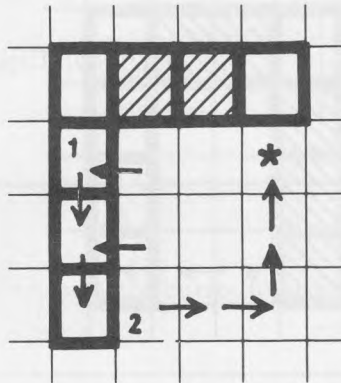
1 2

a. PRINT H\$(1)BK\$V\$(0)UP\$V\$(1)BK\$H\$(1)BK\$



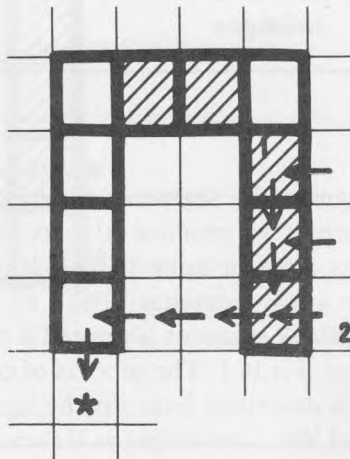
1 2

b. PRINT H\$(1)BK\$V\$(0)UP\$V\$(1)BK\$H\$(1)BK\$



1 2

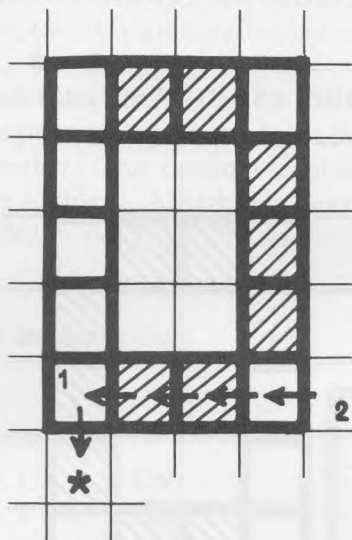
c. PRINT H\$(1)BK\$V\$(0)UP\$V\$(1)BK\$H\$(1)BK\$



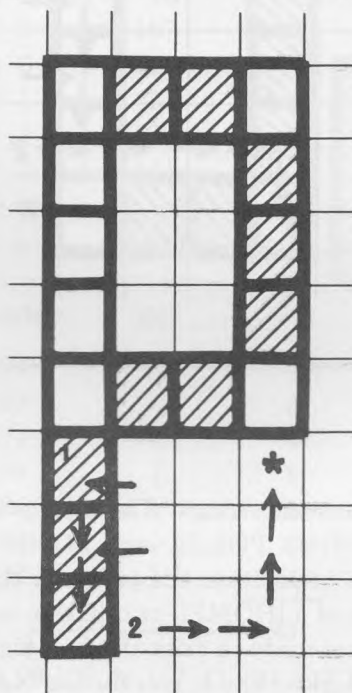
(continued)

Figure 6 (continued)

d. PRINT H\$(1)BK\$V\$(0)UP\$V\$(1)BK\$H\$(1)BK\$



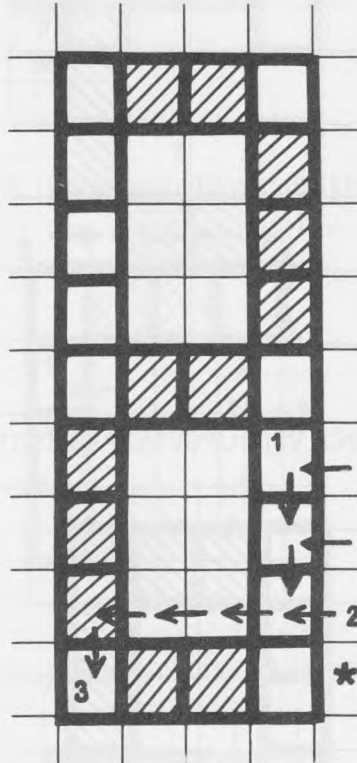
e. PRINT V\$(1)UP\$V\$(0)BK\$H\$(1)



(continued)

Figure 6 (continued)

1 2 3
f. PRINT V\$(1)UP\$V\$(0)BK\$H\$(1)



There are other unseen strings. The strings assigned to PO\$(1), PO\$(2), PO\$(3), PO\$(4), PO\$(5) and PO\$(6) consist of cursor movement characters to position the cursor at the beginning of each digit. PO\$(1) consists of a [HOME], two [CD]s, and a [CL]. After this string is printed, we are ready to print the first digit. The other strings include extra [CL]s to move the cursor for the beginnings of the other four digit positions.

Coding and Decoding the Digits

The seven-segment display lends itself well to control by digital circuitry. The whole pattern for a digit can be represented with a single binary number; 1's represent segments turned on and 0's represent segments turned off.

Figure 7 shows the numbering system for the segments of the C64 Clock display. Each segment is represented in a binary number by the corresponding bit number. The decimal equivalent of the binary number is produced by adding in the right powers of two. This decimal number then is included in the DATA statement in line 9000.

Figure 7: Coding of segment data.

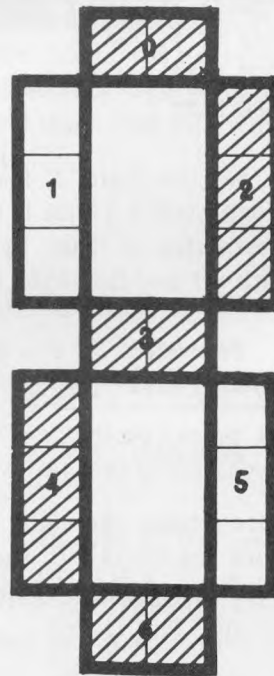
	64	32	16	8	4	2	1	
0	0	1	1	1	0	1	1	119
1	0	0	1	0	0	1	0	36
2	0	1	0	1	1	1	0	93
3	0	1	1	0	1	1	0	109
4	0	0	1	0	1	1	1	46
5	0	1	1	0	1	0	1	107
6	0	1	1	1	1	0	1	123
7	0	0	1	0	0	1	0	37
8	0	1	1	1	1	1	1	127
9	0	1	1	0	1	1	1	111

For digit '2'

Segment #	ON/OFF	BIT	
0	ON	1	1
1	OFF	0	0
2	ON	1	4
3	ON	1	8
4	ON	1	16
5	OFF	0	0
6	ON	1	64

93

9000 DATA 119,36,93,109,46,107,123,37,127,111



The decimal numbers allow very compact storage of the segment data in the BASIC program. However, working directly from these decimal numbers when a digit is printed is slow. As part of the initialization process, the short routine listed here converts the decimal numbers into patterns of 1's and 0's. This is accomplished with the BASIC AND function, using each successive power of two.

```

8500 PRINT "{CLEAR}SETTING UP DIGITS"
8510 DIM S%(9,6)
8520 FOR I=0 TO 9
8530 READ X
8540 FOR J=0 TO 6
8550 IF 2↑J AND X THEN S%(I,J)=1:GOTO 8570
8560 S%(I,J)=0
8570 NEXT J
8580 NEXT I

```

For the digit '2' (I=2), 93 is READ from the DATA statement. When J is 0, $2 \uparrow 0$ (or 1) is ANDed with the 93. The result is 1, which is interpreted as 'true' by the IF statement. $S\%(2,0)$ therefore is filled with a 1 and line 8560 is skipped.

```

0000 0001 (2 0 or 1)
AND 0101 1101 (93)
-----
0000 0001 (1)

```

Then J takes the value of 1. This time $2 \uparrow 1$ or 2 is ANDed with 93. Since the bit is a 0, the result of the operation is 0. This is read as 'false', so the rest of line 8550 is skipped and line 8560 is executed, setting $S\%(2,1)$ to 0.

```

0000 0010 (2 1 or 2)
AND 0101 1101 (93)
-----
0000 0000 (0)

```

Each bit is tested similarly, with 1 stored in the integer array if the result of the AND operation is non-zero and 0 stored if the result is zero. Then I increments to 3. Line 8530 READs 109 from the DATA statement, and the appropriate 1's and 0's are put into the integer array.

The final link in the decoding process is converting these 1's and 0's into segments. Figure 6 shows the subscripts for H\$() and V\$() as 1's and 0's. Where do these 1's and 0's come from? If you look at program lines 1510 and 1520 you will see that in place of each subscript an array element S%(T,) is shown. This is the very same array we just constructed. To determine whether to print a particular segment on or off, the program looks for a 1 or 0 in the integer array.

Keeping Time, the C64 Way

The C64 does most of the work when keeping time. The system maintains a special six-character string TI\$ that maintains the time in hours, minutes, and seconds. When the computer is turned on, TI\$ contains six '0' characters. The first two represent hours, the next two represent minutes, and the last two represent seconds. The value of TI\$ can be changed, though, to represent the real time. The following short program demonstrates C64's built-in clock.

```
10 INPUT "{CLEAR}TIME (HHMMSS)";TI$
20 PRINT "{CLEAR}"
30 PRINT "{HOME}"TI$
40 GOTO 30
```

RUN the program, inputting a string of six digits representing the current time. The time is maintained by the C64 in 24-hour format, so to get PM times add 12 to the hours portion. The time will be displayed continually at the top of the screen.

Press the STOP key, wait a few seconds, then type CONT and RETURN. The time shown is current because TI\$'s value is maintained even when no program is running. The only time the C64 fails to keep proper time in TI\$ is when the disk or cassette drive is used.

TI\$ is a special variable; its value changes independently of your program and it can be changed only to a string of six numeric characters. TI is another reserved variable. It increments for every sixtieth of a second (or 'jiffy'). Unlike TI\$, however, it cannot be assigned a value. It starts at 0 and continues to get bigger and bigger. A third reserved variable is ST, which is changed to reflect information on the status of data input and output operations. ST, like TI, cannot be assigned a value by a program.

C64 Clock Program Description

Initialization sequence (lines 10-30): The program starts with a call to the initialization subroutine (8000), where constants are defined and the segment coding for the digits is set up. Next, the time-setting routine 5500 is called (with T\$ set to "T"), followed by a screen-clear.

Program mainline (lines 100-240): Line 110 sets the array of position flags to all -1's. The elements of this array normally contain the last value for each digit in the time display. Minus 1 is an impossible digit value, so this ensures that all digits are printed on the first run. The MID\$ function is used to extract one character starting at the fifth position of TI\$. The RIGHT\$ function extracts one character from the right end of TI\$.

Lines 120-200 go through the six digits of TI\$, displaying them as seven-segment digits on the screen. Because the sixth digit changes the most often, the FOR...NEXT loop counts down from 6 to 1. This is accomplished by using the extension 'STEP -1' in line 120. Without this extension, the loop would execute only once. Line 130 extracts the proper digit from TI\$ using the MID\$ function. MID\$ returns a character, so we need to turn the character into a number in T using the VAL function. Line 140 drops through to line 150 unless a key has been pressed. Because this statement occurs in the middle of a FOR...NEXT loop, we can't jump out of it without a little tidying up. Before the GOTO 100, we must terminate the FOR...NEXT loop by setting I to the ending value and performing a NEXT statement.

Because the time maintained in TI\$ is in 24-hour format rather than AM/PM, the digits for the first two positions have to be adjusted for PM times. Subroutine 4000 performs this adjustment. The ON...GOSUB statement allows us to perform the full conversion (by starting at 4000) when the display reaches the second digit. However, when the display reaches the first digit, the conversion has already been performed and we need to extract only the first digit from TM\$ (by starting at line 4060). Remember that the FOR...NEXT loop (lines 120-200) counts down from 6 to 1. When I is 3, 4, 5, or 6, the ON...GOSUB statement is skipped.

Line 160 compares the current digit with the last digit printed in that position. If they are the same, there is no need to print the same thing over again and a branch is made to the NEXT statement (line 200). Line 170 prints the position string and line 180 prints the appropriate large digit in that position with a call to subroutine 1500. Finally, the 'previous digit' array element is set to the value of T. The NEXT statement goes back to 120 for the next digit.

In line 210, either 'AM' or 'PM' is printed in the upper left-hand corner, depending on the value of PM. PM can have a value only of -1 (for PM) or 0 (for AM). Adding 1 gives us the proper index for AP\$(), defined in line 8090. In line 220 the alarm-set flag is tested. If it is set (not equal to zero), then the alarm-check routine at line 1000 is executed. Line 230 prints either 'ALARM SET:' in red or nineteen spaces, depending on the status of AF, the alarm-set flag. If AF is set, the hours and minutes portion of TI\$ is printed next.

Finally, GOTO 110 starts the mainline over again, but without clearing the previous-digit array.

Alarm-check routine (lines 1000-1020): This routine consists of a simple comparison of the first four characters of TI\$ and TA\$ (the time for which the alarm is set). If the two strings are found to be equal, then the alarm routine is executed. As long as the alarm is set (AF does not equal zero) and the times are equal, the program flow will go from line 220 to the subroutine at line 10000 to check the alarm. If it is time for the alarm to go off then the program flow goes to the subroutine at line 2000 which makes the bell ring. Line 230 prints the alarm heading or blanks at the top of the clock. If the alarm flag is set then program flow goes to the subroutine at line 500 which prints the alarm time in AM/PM format. In line 240 program control returns to the main routine in line 120. If the alarm is turned off (by pressing 'A') or the time advances to the next minute, then the alarm will stop.

Print-digit routine (lines 1500-1530): This routine uses the array elements from S%(,) to print a seven-segment digit with the appropriate segments on or off. The first subscript, corresponding to the digit value, is received in T. Each element in the array is set to 1 or 0 by the set-up routine starting at line 8500. A 1 indicates that a segment is on, and a 0 indicates that a segment is off. Lines 1510-1520 print a series of strings, end to end. Each H\$() or V\$() receives a 1 or a 0 as its subscript from the value of the appropriate S%(,). H\$(1) and V\$(1) are 'on' segments, while H\$(0) and V\$(0) are 'off' segments. For a more detailed description, see the main text and the accompanying figures.

Alarm routine (lines 2000-2060): SC and LO were set in line 8110 to the addresses for the C64 registers that control the screen color and the beginning decimal address of the Sound Interface Device (SID). A value of 2 for SC turns the screen red (line 2010) and 5 turns it back to green (line 2050). The FOR ... NEXT loop in line 2020 clears the sound chip. The values in line 2030 are assigned for the attack, sustain, volume, high frequency, low frequency, and waveform, respectively. The statement POKE LO4,33 in line 2030 actually turns on the sound. By

changing the 33 to a 32 in line 2050, the sound is turned off. The FOR...NEXT loop in 2040 determines the duration of the sound.

Mode-select routine (lines 3000-3140): If the character in T\$ is an 'A', then line 3010 toggles the value of AF between -1, which is read by an IF...THEN statement as true, and 0, which is read as false. Minus 1 in binary is 1111 1111 and 0 is 0000 0000. The NOT instruction changes every 1 bit to 0 and every 0 bit to 1, so NOT(-1)=0 and NOT(0)=-1. If T\$ is not 'A', then the menu of options is printed. Line 3080 looks for a character from the keyboard. The five acceptable characters are tested in lines 3090-3130. If no acceptable character is received, then line 3140 branches back to 3020 for another character. A Carriage Return returns to the mainline without changing either the time or the alarm-set time. An 'A' causes the alarm-set subroutine (5000) to be executed and a 'T' causes the time-set (5500) subroutine to be executed. Each of these subroutine calls is followed by a clear-screen and RETURN to the mainline.

PM correction routine (lines 4000-4080): The left two digits of TI\$ (representing hours) are extracted with the LEFT\$ function. Next, this two-character string is converted to a number in TM using the VAL function. In line 4030, if TM is greater than 12, then 12 is subtracted from TM. This value is converted to a string consisting of a space and the one or two characters representing the hours. The space is removed by using the MID\$ function to take the remainder of the string starting with the second position. Line 4040 tests for 0 hours (midnight) and converts TM\$ to '12'. If TM\$ contains only one digit, then a '0' is placed on the front using the "" operator (concatenation).

Line 4060 extracts the proper digit from TM\$ to return to the mainline. This time MID\$ is used in its full form. I's value is set by the FOR...NEXT loop in the mainline. Here it can be only 1 or 2. One character, starting with the Ith position, is extracted. This is returned in T. Line 4070 sets the AM/PM flag PM to -1 if TM (the hours in 24-hour format) is greater than or equal to 12. If TM is less than 12, then PM is set to 0.

Set alarm (lines 5000-5050): Line 5010 sets the alarm-set flag to -1 and changes the screen color to red. The alarm/time set routine at line 6000 is called. Then TM\$, the temporary string holding the new time is copied into TA\$, presumably changed. Before returning, the screen color is changed back to green.

Set time (lines 5500-5550): This routine is very similar to the alarm set routine. The screen color is changed to purple. The set time/alarm routine is called in line 5520 and the current time is set to the new time held in TM\$. The screen color is returned to green before the return.

Alarm/time set (lines 6000-6520): This routine, used for both setting the time and for setting the alarm, starts with a clear-screen followed by the appropriate heading 'SET TIME' or 'SET ALARM'. The expression 'T\$="T"' evaluates to -1 if true and to 0 if false. Adding 1 produces a subscript of 0 or 1 for SA\$(), defined in line 8120.

Lines 6200-6230 input hours and minutes into TH and T(2). Space is provided at lines 6240-6260 to input seconds if you so desire. See the section on modifying the CLOCK program. Line 6270 asks whether it is AM or PM. In line 6290 the PM flag is set to -1 if it is PM. Note that T(1) represents the hours in 24-hour format, while TH is used to represent the hours in AM/PM format. If it is PM then line 6430 sets T(1) equal to TH plus twelve hours, converting it to 24-hour format. Therefore lines 6300-6410 use TH to display the clock in hour format, while lines 6500-6510 use the T(1) 24-hour format to change the internal clock.

Lines 6300 and 6310 form a new TM\$ from TH and T(2). ZR\$ is a string of six zeros defined in line 8010. It is used to pad the resulting string with zeros when TH or T(1) has only one digit. As explained above, using STR\$ on a positive number leaves a leading space. This is accounted for in operating on ZR\$. The maximum length of STR\$(TH) or STR\$(T(2)) is 3, including the space, so 3 minus this length determines the number of zeros to extract from the left end of ZR\$. The string of zeros is concatenated to the string of digits (with the space removed).

Line 6320 HOMEs the cursor and prints the appropriate AM/PM indication. MS\$ positions the cursor under the time display and prints HOURS MINUTES SECONDS. Line 6400 calls the static-display routine (line 7000) to display the time on the screen. Lines 6410-6420 print a message and wait for the user to strike a key.

TM\$ is put together in lines 6500 and 6510, as it was in lines 6300 and 6310. This time '00' is added for the seconds to make a full six-character string, as required for TI\$.

Static-time display (lines 7000-7040): This routine is much simpler than the dynamic display routine in the mainline. The FOR...NEXT loop calls subroutine 1500 to print the digits from left to right. T is the value of the appropriate digit in TM\$, while the appropriate PO\$() subscript is determined by the value of I.

Initialization (lines 8000-8590): Lines 8000-8120 define a number of constants, including the strings that form the on and off segments, positioning strings, messages, and C64 chip registers. These are discussed in more detail above where they are used.

Set up digits (lines 8500-8580): This routine takes the decimal numbers in the DATA statement (line 9000) and converts them into 1's and 0's in the integer array S%(,). See figure 7 for a graphic illustration of the process.

X is the current number read from the DATA statement. The expression '2 JANDX' yields the actual result of an AND operation on the bit level. Therefore, to test bit 3 we AND X with 2 raised to the 3rd power (or 8). The result is 8 if the bit is on, and 0 if it is off. The inside FOR...NEXT loop goes through the bit positions one by one starting with bit 0 and ending with bit 6. Bit 7 is not used. Remember that the IF statement accepts any non-zero value as true, so whenever a bit is on, a 1 will be put in the corresponding element of the array S%(,) and line 8560 is skipped. If the bit is off then line 8560 puts a zero into the array element.

Menu loader: This routine loads the disk menu program when the user selects Menu during the select mode.

Expanding the C64 Clock Program

Addition to set seconds

If you notice your clock listing you will see a space (lines 6240-6260) ready for you to insert the lines to allow seconds to be set on your clock. Carefully type in the lines listed below.

```
6240 INPUT "{WHITE,UP}SECOND (0-59)";T
6250 IF T<0 TO T>59 THEN 6240
6260 T(3)=T
6315 TM$=TM$+LEFT$(ZR$,3-LEN(STR$(T(3))))+MID$(STR$(T(3)),2)
6510 TM$=TM$+LEFT$(ZR$,3-LEN(STR$(T(2))))+MID$(STR$(T(2)),2)
6515 TM$=TM$+LEFT$(ZR$,3-LEN(STR$(T(3))))+MID$(STR$(T(3)),2)
```

Adding lines 6240-6260 adds an additional value input every time the clock or alarm is set. Lines 6315, 6510, and 6515 change TM\$, the temporary time string, so that TI\$, the C64 internal time string, will be adjusted properly.

More Attractive Digit Display

The appearance of the digits on the screen can be improved greatly by filling in the six segment intersections with solid or diagonal half characters. To accomplish this, we must define eleven different horizontal "segment" strings and change the coding to reflect these new possibilities. The analogy to a seven-segment display clock is no longer valid. Type in the following lines to improve the appearance of the digit display.

8030 H\$(0)=" {RVS} {RVSOFF}":	0	
H\$(1)="{RVS}{↑POUND} {C=*}{RVSOFF}":	1	
H\$(2)="{RVS}{↑POUND} {RVSOFF}{↑POUND}":	2	
H\$(3)="{RVS} {RVSOFF}"	3	
8032 H\$(4)="{RVS} {C=*}":	4	
H\$(5)="{RVS} {RVSOFF}{↑POUND}":	5	
H\$(6)="{RVS} {RVSOFF} {RVS} {RVSOFF}":	6	
H\$(7)="{C=*}{RVS} {C=*}{RVSOFF}"	7	
8034 H\$(8)="{C=*}{RVS} {RVSOFF}{↑POUND}":	8	
H\$(9)="{C=*}{RVS} {RVSOFF}":	9	
H\$(10)="{RVS} {RVSOFF}"	10	

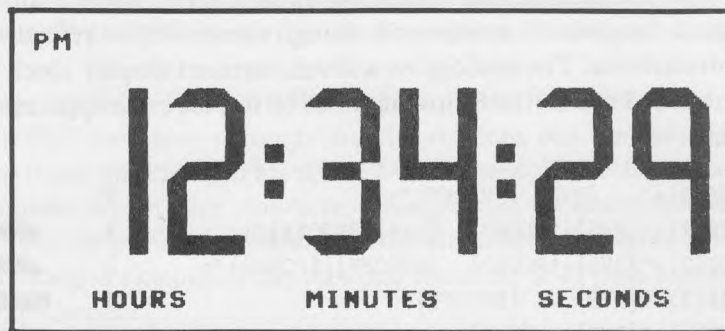
```

8530 READ X$
8540 FORT J=1 TO 7
8550 X=ASC(MID$(X$,J,1))-48
8560 S%(I,J-1)=X
9000 DATA "1116118","0010010","1012103",
          "101:018","6113010","3107015"
9010 DATA "1104118","3010010","111:118",
          "1119010"

```

Lines 8030-8034 define the eleven different horizontal segment strings. Note the use of the two diagonal fill characters and their reversed images. The digits are now coded with seven-character strings instead of single numbers. The colon is used to represent '10' with a single character. When S%(,) is constructed from these strings, the MID\$ function is used to extract each character. The ASC function converts the character into its numeric code. Subtracting 48 changes the code into a number ranging from 0 to 10. The colon was chosen to represent '10' because its code is 58.

The display routine (lines 1500 on) works exactly the same way. The PRINT statements still look up the proper subscript for H\$() in the array S%(,). Now there can be any number from 0 to 10 there, rather than just 0 or 1.



Adding a Chime

This change requires the addition of two program lines:

```
185 IF I=2 THEN CH=VAL(TM$)
225 IF CH THEN GOSUB 2000:CH=CH-1:GOTO 240
```

CH is used to count the proper number of chimes. In line 185 it is set to the current number of hours, as corrected in subroutine 4000 for AM/PM. This line is reached only when the digit in a particular position changes. If I is 2, then the hour has just changed. Line 225 tests CH; if it is not zero, then a chime is sounded by a call to the alarm routine. CH is decremented and the alarm test is skipped.

You may want to make improvements in the chime. Try programming a mechanism to avoid chiming when the time has just been set. Also, you might try using different frequencies for the chime and alarm.

Other changes to consider in the program are running it on 24-hour time, playing a melody on the hour, or introducing a ticking sound. The 24-hour modification is a matter of deletion rather than of addition. The others require more effort.

```

10 REM    C-64 CLOCK
20 GOSUB 8000
30 T$="T":GOSUB 5500
40 PRINT"{CLEAR}"
100 PRINT"{HOME}"AP$(PM+1)MS$ "HMS$
110 FOR I=1 TO 6:T(I)=-1:NEXT
120 FOR I=6 TO 1 STEP -1
130 T=VAL(MID$(TI$,I,1))
140 GET T$:IF T$<>" " THEN GOSUB 3000:I=1:NEXT:GOTO 100
150 ON I GOSUB 4060,4000
160 IF T=T(I) THEN 200
170 PRINT PO$(I);
180 GOSUB 1500
190 T(I)=T
200 NEXT
210 PRINT"{HOME}"AP$(PM+1)
220 IF AF THEN GOSUB 1000
230 PRINT AL$(AF+1);:IF AF THEN GOSUB 500
240 GOTO 120
500 REM    PRINT ALARM TIME
510 IF VAL(LEFT$(TA$,2))<12 THEN PRINT LEFT$(TA$,2)":
    "MID$(TA$,3,2)" AM{BLACK}":RETURN
520 PRINT VAL(LEFT$(TA$,2))-12 "{LEFT}":
    "MID$(TA$,3,2)" PM{BLACK}"
530 RETURN
1000 REM CHECK ALARM
1010 IF LEFT$(TI$,4)=LEFT$(TA$,4) THEN GOSUB 2000
1020 RETURN
1500 REM PRINT DIGIT
1510 PRINT H$(S$(T,0))BK$V$(S$(T,1))UP$V$(S$(T,2))BK$
    H$(S$(T,3))BK$V$(S$(T,4));
1520 PRINT UP$V$(S$(T,5))BK$H$(S$(T,6))
1530 RETURN
2000 REM ALARM
2010 POKE SC,2
2020 FOR L=LO TO LO+24:POKE L,0:NEXT:REM CLEAR SOUND CHIP
2030 POKE LO+5,9:POKE LO+6,0:POKE LO+24,15:POKE LO+1,28:
    POKE LO,214:POKE LO+4,33
2040 FOR I=1 TO 700:NEXT
2050 POKE LO+4,32:POKE SC,5
2060 RETURN

```

```

3000 REM MODE SELECT
3010 IF T$="A" THEN AF=NOT(AF):RETURN
3020 POKE SC,1:PRINT"{CLEAR,DOWN5,RIGHT14,RED}SELECT MODE:
      {BLUE}"
3030 PRINT"{DOWN4,RIGHT14}SET {RVS}A{RVSOFF}LARM"
3040 PRINT"{DOWN,RIGHT14}SET {RVS}T{RVSOFF}IME"
3050 PRINT"{DOWN,RIGHT14,RVS}RETURN{RVSOFF} TO CLOCK"
3060 PRINT"{DOWN,RIGHT14}EXIT TO {RVS}B{RVSOFF}ASIC"
3070 PRINT"{DOWN,RIGHT14}EXIT TO {RVS}M{RVSOFF}ENU{BLACK}"
3080 GOSUB 7500
3090 IF T$=CR$ THEN PRINT"{CLEAR}":RETURN
3100 IF T$="A" THEN GOSUB 5000:PRINT"{CLEAR}":RETURN
3110 IF T$="T" THEN GOSUB 5500:PRINT"{CLEAR}":RETURN
3120 IF T$="M" THEN GOSUB 20000
3130 IF T$="B" THEN PRINT"{CLEAR}":END
3140 GOTO 3020
4000 REM PM CORRECTION
4010 TM$=LEFT$(TI$,2)
4020 TM=VAL(TM$)
4030 IF TM> 12 THEN TM$=MID$(STR$(TM-12),2)
4040 IF TM=0 THEN TM$="12":GOTO 4060
4050 IF LEN(TM$)=1 THEN TM$="0"+TM$
4060 T=VAL(MID$(TM$,I,1))
4070 PM=TM>=12
4080 RETURN
5000 REM SET ALARM
5010 AF=-1:POKE SC,2
5020 GOSUB 6000
5030 TA$=TM$
5040 POKE SC,1
5050 RETURN
5500 REM SET TIME
5510 POKE SC,4
5520 GOSUB 6000
5530 TI$=TM$
5540 POKE SC,1
5550 RETURN

```

```

6000 REM ALARM/TIME SET
6010 PRINT "{CLEAR}      "SA$(T$="T")+1) "{DOWN3}"
6200 INPUT "{WHITE,UP}HOUR (1-12)";T:
      IF T< 1 OR T> 12 THEN 6200
6210 TH=T
6220 INPUT "{WHITE,UP} MINUTE (0-59)";T:
      IF T< 0 OR T> 59 THEN 6220
6230 T(2)=T
6240 REM
6250 REM  INSERT LINES TO SET SECONDS HERE
6260 REM
6270 INPUT "{WHITE,UP,SPC9,RVS}A{RVSOFF}M OR {RVS}P{RVSOFF}M";
      T$:IF T$<> "A" AND T$<> "P" THEN 6240
6280 PRINT "{BLACK,CLEAR}"
6290 PM=0:IF T$="P" THEN PM=-1
6300 TM$=LEFT$(ZR$,3-LEN(STR$(TH)))+MID$(STR$(TH),2)
6310 TM$=TM$+LEFT$(ZR$,3-LEN(STR$(T(2))))+MID$(STR$(T(2)),2)
6320 PRINT "{HOME}"AP$(PM+1)MS$ "    "HMS$"{DOWN6}"
6400 GOSUB 7000
6410 PRINT "{DOWN7,WHITE,SPC5,↑H}IT ANY KEY TO CONTINUE{BLACK}"
6420 GOSUB 7500
6430 T(1)=TH:IF PM=-1 THEN T(1)=TH+12
6500 TM$=LEFT$(ZR$,3-LEN(STR$(T(1))))+MID$(STR$(T(1)),2)
6510 TM$=TM$+LEFT$(ZR$,3-LEN(STR$(T(2))))+
      MID$(STR$(T(2)),2)+"00"
6520 RETURN
7000 REM STATIC TIME DISPLAY
7010 FOR I=1 TO 6
7020 PRINT PO$(I);:T=VAL(MID$(TM$,I,1)):GOSUB 1500
7030 NEXT
7040 RETURN
7500 GET T$:IF T$="" THEN 7500
7510 RETURN

```

```

8000 REM PRIMITIVE DEFINITIONS
8010 ZR$="000000"
8020 V$(0)=" {LEFT,DOWN} {LEFT,DOWN} ":V$(1)="{RVS} {LEFT,
      DOWN} {LEFT,DOWN} {RVSOFF}"
8030 H$(0)="      ":H$(1)=" {RVS} {RVSOFF} "
8040 BK$="{LEFT4,DOWN}":UP$="{RIGHT2,UP2}"
8050 PO$(1)="{HOME,DOWN3,RIGHT2}"
8060 FOR I=2 TO 6:PO$(I)=PO$(I-1)+ "{RIGHT5}":NEXT
8070 PO$(3)=PO$(3)+"{DOWN3,RVS} {RVSOFF,LEFT,DOWN2,
      RVS} {RVSOFF,RIGHT,UP5}":PO$(4)=PO$(4)+"{RIGHT2}"
8075 PO$(5)=PO$(5)+"{DOWN3,RIGHT2,RVS} {RVSOFF,LEFT,DOWN2,
      RVS} {RVSOFF,RIGHT,UP5}":PO$(6)=PO$(6)+"{RIGHT4}"
8080 MS$="{HOME,DOWN14}":HMS$="HOURS      MINUTES      SECONDS"
8090 AP$(0)="PM":AP$(1)="AM"
8100 AL$(0)="{HOME,RED,RIGHT6}ALARM SET: ":
      AL$(1)="{HOME,RIGHT6,SPC18}"
8110 LO=54272:SC=53281:CR$=CHR$(13)
8120 SA$(0)="SET TIME":SA$(1)="SET ALARM"
8130 POKE SC,1:POKE 36878,15
8500 PRINT"{CLEAR,BLACK}SETTING UP DIGITS"
8510 DIM S%(9,6)
8520 FOR I=0 TO 9
8530 READ X
8540 FOR J=0 TO 6
8550 IF 2{UP ARROW}J AND X THEN S%(I,J)=1:GOTO 8570
8560 S%(I,J)=0
8570 NEXT J
8580 NEXT I
8590 RETURN
9000 DATA 119,36,93,109,46,107,123,37,127,111
20000 REM MENU LOADER
20010 PRINT"{CLEAR}LOAD "+CHR$(34)+"0:MENU"+CHR$(34)+"",8":
      PRINT "{DOWN4}RUN"
20020 POKE 631,19: POKE 632,13: POKE 633,13: POKE 198,3: END

```


C64 Clock Variable Usage

Constants

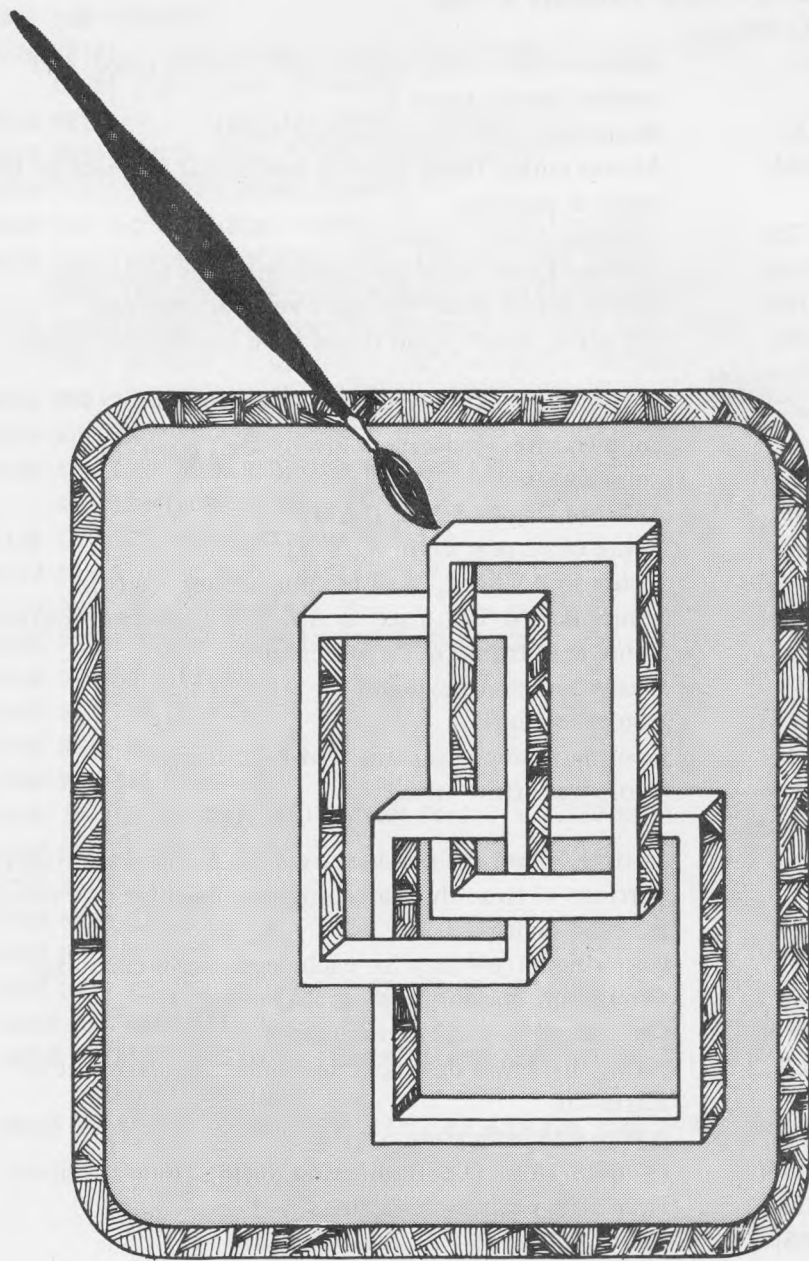
SC	Address controlling screen color, border color, and reverse mode status
LO	Beginning address controlling sound
BK\$	Moves cursor ready to print horizontal segment or left vertical segment
CR\$	Carriage return character
MS\$	Moves cursor ready for hours/minutes indication
UP\$	Moves cursor ready for right-vertical segment
ZR\$	Six zeros, used to pad times with insufficient digits

Variables

AF	Alarm flag (- 1 set, 0 off)
I	Loop counter, indicates current digit position
J	Loop counter
PM	AM/PM flag (- 1 PM, 0 AM)
T	Value of current digit
	Hours in AM/PM, used in time setting routine
TM	Hours in AM/PM, used in AM/PM correction routine
X	Value read from DATA statement
T\$	Character from keyboard
TA\$	Alarm-set time
TIS	C64-maintained real time, 24-hour format
TM\$	Temporary time string

Arrays

T()	In time/alarm-set routine, used for hours and minutes portions of time. In main program, used for last values for each digit position
S%(,)	Contains on/off data for each segment in each digit
H\$()	On and off horizontal segment strings
V\$()	On and off vertical segment strings
AL\$()	String displayed during time display on alarm-set or alarm-off
AP\$()	AM or PM indication
HM\$()	HOURS/MINUTES indication during time/alarm-set
PO\$()	Four digit-positioning strings
SA\$()	SET TIME or SET ALARM message



Screen Editor

Using Your Video Display

SCREEN EDIT AND DRAW INSTRUCTIONS	
KEY	FUNCTION
F1	DISPLAY THESE INSTRUCTIONS
F2	SELECT DISK OR TAPE DEVICE
F3	CHANGE SCREEN OR COLOR
F4	CHANGE BORDER OR COLOR
F5	LOAD SCREEN
F6	SAVE SCREEN
F7	ENTER/EXIT LINE DRAWING MODE

One of the most dramatic features of the Commodore 64 is its colorful graphics capability. Unfortunately, the software provided with the computer is very limited.

This Screen Editor program provides a full screen editor that permits you to experiment with the built-in characters and colors; to save and load complete display screens to tape or disk; to use specially designed graphic characters to make simple and complex line drawings; to change screen, border and character colors; and to explore the wonders of character graphics.

Running Instructions

1. Execute the SCREEN EDITOR program from the disk menu.
2. A screen of operating instructions will appear. Press any key to get into the Edit mode.
3. In the Edit mode, you have use of all the standard 'built-in' characters. These include all of the alphabet, numbers, punctuation and the graphics that are displayed on the C64 keyboard. Every time you press a key, you generate a character and the cursor moves one position right, just like a typewriter. Try making a simple picture using the standard keyboard characters.
4. Press F7 and notice that the cursor changes from a star to a small block. Now you are in the Draw mode which permits you to draw lines in eight directions, with the cursor moving in the direction of the line, plus supports automatic drawing of squares, rectangles, and other shapes. Try drawing some simple line drawings. You can go back to the Instructions at any time by pressing the F1 key, without losing your picture!
5. F7 toggles between the Edit mode and the Draw mode. All of the cursor control movements, including HOME, CLR, INSERT and DELETE are valid during the DRAW mode. Letters which are used to control the drawing are not available during DRAW. These are available by going back temporarily to the Edit mode. The DRAW mode will be explained in detail later.
6. If you want to load a picture from the disk, press F5 to call the loader, and type SCREENDEMO for the filename.

Operating Instructions

1. Display the instructions page by pressing the F1 key if you are in the editor. Notice the function keys. Each of these is used to perform an operation while in the screen editor mode.

F1 displays the instruction page.

F2 allows you to select tape or disk for storage.

F3 changes the screen color each time it is pressed.

F4 changes the border color each time it is pressed.

F5 loads a display screen from your storage device.

F6 save the current display on your storage device.

F7 toggles between the EDIT (star) and DRAW (small block) mode.

2. When you are in the DRAW mode (small block) you use the keys centered around H to print a line in that direction. The H toggles the pencil from write to erase. Now enter the screen editor by pressing E and clear the screen (shift-home). Press the F7 key to enter the DRAW mode and experiment.
3. Next return to the instructions by pressing F1. Notice that there are additional line drawing commands. You may use pre-defined shapes by pressing S for square, R for rectangle, Z for brick, or D for diamond. Additionally you may alter their sizes by using the W (width) and L (length) commands. If you wish to make a 3 by 9 rectangle you simply type L3W9R while in the draw mode. The side (length) of the square or the width of the diamond may be altered. The brick size cannot be changed.
4. When you are finished with your creation save it by hitting F6. You will be asked for the filename you wish to save it as. Please be patient while the computer saves your screen. Many conversions must be made and BASIC is slow. It takes about two minutes to save the screen to disk!
5. Remember if you get stuck just hit F1 to display the screen editor instructions.

Exploring Character Graphics

One of the great features of your Commodore 64 system is the use of a video display as the primary output device. Not many years ago, the major output device of even the most expensive computer systems was the Teletype®. This slow, noisy and expensive device could provide only a limited hard-copy output consisting of uppercase letters, numbers, and punctuation characters in black and white. What a difference from the capabilities of the modern microcomputer with its full color, high-speed video display. With all the spectacular things that computers can do using high-resolution graphics, sprites, players and missiles, it is easy to underestimate the power and usefulness of character graphics. Of course, you use character graphics all the time. All of your programming, word processing, and electronic spread sheets use the basic alpha-numeric characters of the computer. Have you ever seriously explored the other capabilities of these character graphics on your Commodore 64?

The Importance and Evolution of Character Graphics

What is a Character?

Webster's defines *character* as "a graphic symbol (as a hieroglyph or alphabet letter) used in writing or printing." There are many other definitions of *character*, but we all understand this basic concept. The most common characters we encounter are the letters that make up the alphabet. While these letter characters are primarily used for communication with other human beings, there are special computers, called *optical character readers*, that can also *read* our alphabet. Other types of characters which you might consider are:

- Magnetic characters on bank checks
- Bar Code characters on commercial products
- Symbols for maps, electronics, choreography, architecture, and others

The Evolution of Computer Characters

Computers do not use characters internally. All of the information in the computer is in the form of electronic bits — ones and zeros (which we will refer to here as internal codes, or just codes). Whether or not a particular code represents a character depends on the device that is interpreting these codes. The code whose bit pattern is

□ ■ ■ ■ ■ ■ □ ■ □ HEX 7A DEC 122

could be interpreted as the character 'z' on an ASCII terminal with upper and lower case; as 'Z' on an ASCII terminal with upper case only; as a 'graphic character' on a Commodore 64 display in one

particular mode or a "check mark character" in another mode; a 'diamond' if a PRINT CHR\$(122) is performed, and so on. The only time the computer is concerned with characters (as opposed to internal codes) is when it is dealing with input or output. And, it usually takes a lot of hardware and processing to turn an external character into an internal code within the computer, or to convert an internal code into a character pattern for output to some device.

In the earliest computers, there was no concept of *character* at all. Information was entered by changing external wires on *patch panels*. The next step was to control the input of information into the computer *via* a set of switches. It really was "All ones and zeros!" Information was output via a row of lights. You would really be stretching it a bit to call:

■ □ ■ □ □ □ ■ □

a character. A major conceptual step was achieved as computers were hooked up to Teletype machines, permitting the operator to enter information in standard characters for the first time. Granted, the Teletype was limited to upper case letters, numbers, punctuation and a few control codes — but it was a giant step forward. Even if the string of characters being input was meaningless to the operator — at least in a language sense — it was certainly much easier to enter:

A9 14

than to set switches to read:

■ □ ■ □ ■ □ □ ■
□ □ □ ■ □ ■ □ □

The IBM punch card is another example of the relationship between the human oriented character and the computer's requirement for binary data. The correspondence between the alphabetic characters and their numeric codes is very apparent on the IBM card. On the top line of the card the keypunch machine types the familiar typewriter characters, but underneath, the little rectangular holes are arranged to form a unique binary code for each different character. The operator enters information into the keypunch machine in alphanumeric characters; the computer card reader reads the binary holes.

A Broader Definition

Besides the alphabetic, numeric, and punctuation characters, the computer understands and uses other codes that have no character associated with them. These include the typewriter-like functions — carriage return, line feed, and tab. Other functions such as cursor movement, inserting and deleting characters and lines, home, clear screen and clear-to-end-of-line and -page make sense on a video

terminal. Additional functions such as stop list, break, changing colors, or reversing the image depend on the particular brand of microcomputer.

The typical personal computer's microprocessor handles numbers in eight-bit bytes. Eight bits are enough to handle 256 different codes — more than adequate for most microcomputer applications. Commodore systems, including PET, CBM, VIC, and Commodore 64, have 128 different characters (plus the 128 reverse images) that can be displayed on the screen. There are more than 128 codes (not all are displayable characters) that can be generated from the keyboard and handled by BASIC. The extra codes include cursor movements, color controls, and special functions. There are two complete character sets available (one at a time). One includes an extensive array of graphic characters, and the other includes lower case, at the expense of 26 graphic characters.

Hardware Characteristics Of Screen Display

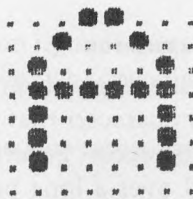
The method that the Commodore 64 uses to display characters and other graphic images on the CRT monitor or TV set is a fairly complicated procedure. Since the screen must be rewritten at the rate of 60 times a second to prevent screen flicker, the whole process of converting from computer codes to screen display character patterns must be done often and quickly..

The smallest screen display point is called a pixel (short for picture element). These pixels can be individually seen on a high quality monitor. The number of horizontal and vertical pixels is called the screen resolution. The Commodore 64 has a resolution of 320 horizontal by 200 vertical.

Most microcomputers use memory mapped video display. This means that the current screen information is stored in random access memory. The RAM memory locations used for storing the current screen information are called the screen memory. Each position on the screen has a corresponding location in memory which holds the internal code for the character to be displayed at that position. (High-resolution graphics usually use some sort of encoding scheme so that several pixels can be stored in one byte. This is done to prevent the screen display from using all available RAM.) Screen memory can be almost anywhere in the Commodore 64 and is pointed to with a screen address vector. The standard default area for screen memory storage is \$0400 - 07FF.

The system input routine normally handles the keyboard input and

The microcomputer has special hardware, the display processor, to handle the generation of characters from internal codes, the refreshing of the video display, and other display oriented activities. The system microprocessor is normally *not* heavily involved in the continual maintenance and refreshing of the display. During each raster scan (60 each second), the screen memory must be scanned by the display processor. Each memory location, including those containing blanks or spaces, is read and the code is converted into a bit-pattern that will generate the character for printing on the screen. A sample bit-pattern for a Commodore "A" is shown below:



The character memory has the bit-patterns for all of the characters that the computer can print on the screen. These patterns are normally stored in ROM. In the Commodore 64, the character memory is pointed to by an address in RAM, which means that this pointer can be modified to point to a start address in RAM where you can define your own character set.

This process must be repeated 60 times a second. All screen updating is done during the vertical blanking period when the beam is moving back to the top left of the CRT in preparation for the next scan. This prevents the jibberish that would appear on the screen if a character should be changed during the screen refresh process.

The Screen Editor allows you to draw line images, character graphics images and character screen formats directly on the screen interactively with immediate feedback and correctability for fast, efficient generation of screen displays. Compared to programming PRINT statements with the accompanying running, editing and rerunning to work everything out, a screen editor allows more time to be spent on the actual looks of the screen and less time on how to generate it.

The program allows the 'cursor' to move around the screen to be able to place characters anywhere on the screen independently of other characters already in place. These characters range from normal alphabetic characters to individually programmed characters which can be any shape possible in the matrix format of the computer. The screen may be saved to disk or tape and later loaded. When the display screen is saved, the program also must save the color memory information.

Programmable Character Generator

Someone, somewhere, during the design of your Commodore 64, decided what each and every character that your system could generate would look like. The character generation capabilities of the microcomputer has evolved over a long period of time. The current generation of personal computers — including the Commodore 64 — has made significant additions to the character graphics. These additions include larger character sets (ROM and RAM were becoming cheaper); extensive use of color; multiple display modes; programmable characters and more. The result is that you are no longer dependent on a fixed set of graphics.

With the Commodore 64 you can program new characters. Characters are normally stored in ROM in an 8 X 8 bit matrix of 8 bytes per character. The Commodore 64 has 256 different characters plus the inverse images. You can redefine characters by placing the character table in RAM and changing the pointer locations to reflect the new area to be utilized. Then, by copying the old set and changing a few of the characters or creating a completely new set, you can print any type of character that you want on the screen, as long as it has an 8 X 8 format.

Why should you need additional characters? You have probably lived this long without them — in fact, you have probably not made use of all the characters your computer gives you!

The DRAW Program

The Screen Editor gave us the capability to put any character in any location of the display. This allows us to make pretty pictures using the built-in character graphics of our computer. However, it is

awkward to draw line-oriented pictures and diagrams since the cursor is always moving one position to the right, just as if we were typing text, even when we would rather not move the cursor in that direction. The DRAW routines solve this problem in two steps:

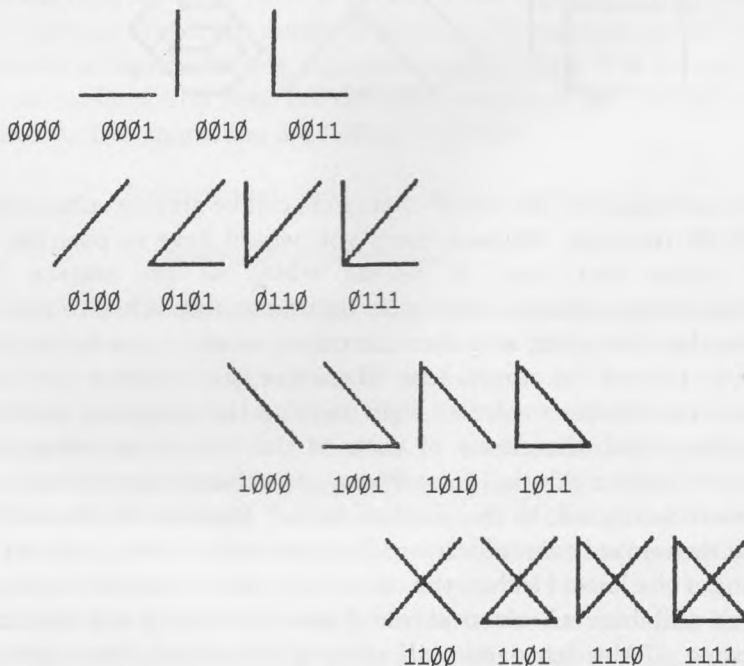
1. It defines a set of line graphics which can be used to draw charts, pictures and diagrams.
2. It provides special cursor controls that move the cursor in the direction of the line being drawn.

These two features working together make drawing very easy.

This special drawing character set is made up of all possible combinations of four basic lines:

line	code
bottom horizontal	0001
left vertical	0010
left top to bottom right diagonal	0100
bottom left to top right diagonal	1000

Combinations of these lines generate sixteen characters:

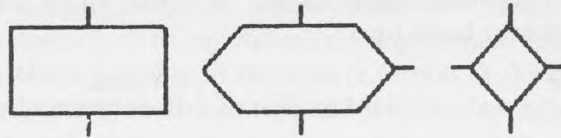


This is a useful character set as it stands, since it can be used to make line drawings such as these:

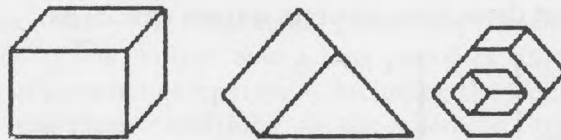
Two Dimensional Shapes:



Chart Symbols:

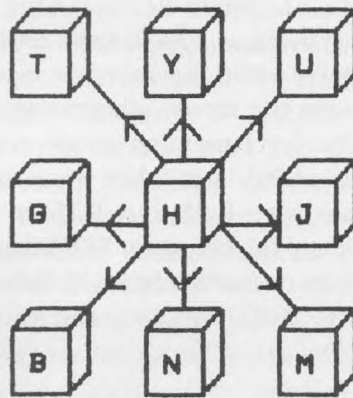


Three Dimensional Shapes:



The usefulness of this set of characters can be greatly enhanced by the DRAW routines. Without them you would have to position the cursor where you want it, decide which of the sixteen line combinations you want at that spot, figure out which key to press to generate that character, and then the cursor would move to the right when you pressed the correct key. There is a better way.

The Draw routines redefine eight keys on the computer keyboard to represent both directions of each of the four independent line elements: North-South, East-West, Northeast-Southwest and Northwest-Southeast. In the diagram below, think of the letter H as being at the current cursor location. If you press the letter J, which is to the right of the letter H, then the cursor will move one position to the right and will draw a horizontal line. Press the letter G and the cursor will move to the left, and will draw a horizontal line segment. Likewise, the other letters surrounding the letter H will move the cursor in the same relative direction as that key is to the letter H, and draw the corresponding line segment.



As you press the key relative to the letter H, the cursor moves and draws a line segment in that direction. You can draw with lines in any direction — quickly and effortlessly!

What happens when there is already a line segment in the character space through which the cursor is moving? For example, what if there is already a horizontal line segment and the letter Y is pressed for a vertical up line? You then get the combination of the two individual segments. The characters are 'ORed' together.

A horizontal — plus a vertical |
generates a corner L

That is the reason that all sixteen possible combinations of the four line segments are defined. A corner is the combination of the horizontal and vertical, meeting in the corner.

While it is handy to be able to draw line segments to form a picture or diagram, it would also be nice to be able to erase individual segments to make alterations and corrections. The DRAW routines also provide this feature. The letter H is used to change the "color" of the pen from the current character color to the background color. Drawing with the background color effectively erases line segments. Note that the entire character position is not erased, only the line segment corresponding to the movement of the cursor. This makes it easy to erase a line and correct any mistakes. The normal cursor movement keys are left intact, so that you can move the cursor without drawing any lines.

Program Description

Initialization [lines 10-40, 20000-20400]: Line 10 calls the main initialization routine starting at 20000. Line 20010 moves the BASIC memory pointer to reserve additional memory space to save the special character set and to save the screen when swapping. Lines 20020 to 20090 move a machine level memory swap program into memory. This special routine is called later when pages of memory are to be swapped to copy the character ROM into RAM or to swap the display. Lines 20100 to 20160 copy the character ROM into RAM. Lines 20170 to 20215 redefine sixteen characters in RAM that will be used for line drawing. Lines 20300 to 20400 complete the initialization by setting up some character strings and defining initial values.

Main Loop [lines 50 to 670]: The call to subroutine 3000 reads the current character and color. Line 60 reads the current color and then plots character CH and color CL by calling subroutine 1200. Line 100 tests variable FT. If FT is 0 then it is the first time through the loop and FT is set to 1 for subsequent tests. Subroutine 17000 is called to display the instructions. Line 110 waits for a character from the keyboard. When a character is pressed, line 120 calls the subroutine at 14000 that tests for the Draw or Edit modes. On return, variable A is tested. If it is empty, then Draw has serviced the key and the main loop is restarted at line 50. If A is not empty, then Draw did not service the character and the main loop must. Lines 210 to 380 perform tests on the character and, if equal, perform some specific task to service the special character. If it is not a special character, then the cursor is set by a call to subroutine 2000 and the character is displayed. Lines 520 to 670 test for the position of the cursor relative to the display, making adjustments at the edges of the display. The cursor is reset using subroutine 2000 and the main loop begins again at line 50.

Cursor/Character Subroutines [Lines 1000 to 3030]: The subroutine at line 1000 plots a character and color by POKEing them into the appropriate locations in memory. The subroutine at line 2000 positions the cursor within the screen by POKEing values into special Commodore 64 locations and then calling a resident (Kernal) routine to convert the information into a cursor address. The subroutine at line 3000 reads the current character and color from memory using PEEKs.

Drawing Routines [Lines 11400 to 11755]: The Dispatch Routine in lines 11400 to 11420 tests for the Draw/Edit mode key, F7. If it is pressed, then the mode is switched. If the mode is now Edit, then an immediate return is made to the Editing routines. If it is now in Draw

mode, then the fifteen special Draw keys (the eight directions T Y U G J B N M, the draw/erase H, and the six special functions S Z L W R D) are tested. If one of the eight Draw Line keys is encountered, then calls are made to the Draw Line Segment subroutines. Each of the eight Draw line keys cause both movement of the cursor and writing or erasing a line segment. The letter H controls pen up/pen down by changing the sign of the variable PEN and then exits. The special functions are serviced by branching to routines in lines 11700 to 11750. If not one of the Draw mode keys, then a return is made to the Editor for normal operation. This permits the cursor controls and all keys other than the special fifteen to be used normally. All of the digits, for example, are available.

The Draw Line Segment routines consist of eight sets of calls to the Move and Display subroutines, one for each direction key. Each of the eight calls controls the generation of one of the eight line segment/direction combinations. For example, the line segment generated for the T key, that is the diagonal going Northwest from the current cursor location, is the series:

Move the cursor left one space
Display the Northwest diagonal character
Move the cursor up one space

The Draw Line Segment subroutines in lines 11480 to 11488 provide this support service.

The Move subroutines in lines 11500 to 11520 handle the cursor movement. There are four basic directions: UP, DOWN, LEFT and RIGHT. Diagonal movements are made by moving both horizontally and vertically. The order of the movement and the displaying of the line segment are critical. For example, the letter T first moves the cursor left one position, draws the diagonal, then moves the cursor up one position. The letter U first prints the diagonal, then moves the cursor one position right and one position up.

The Display subroutines in lines 11600 to 11660 control the displaying of the correct graphic character. If there is a character in the space to be written that is not one of the line segment characters or combinations, then it is erased. If the current location to be written in contains line segments, then the new segment is combined with the existing segments by a logical OR operation if the pen is down (writing), or is removed from the existing segments by a logical AND operation if the pen is up (erasing). A flag is set so that upon return to the normal Editing routines, a test will show that the character was a Draw command and that it has been serviced. The Display subroutines handle the actual output of the graphic character.

The Special Functions are serviced by routines in lines 11700 to 11755. The Square starting at line 11700, for example, uses four consecutive loops to draw first up, then right, then down and finally left to make a square that is LN units on each side. The brick shape starts at line 11710; the length of the rectangle starts at line 11720; the width of the rectangle starts at line 11730; the rectangle shape starts at line 11740; and, the decision box starts at line 11750.

Disk and Tape Subroutines [lines 14000 to 16100]: These subroutines are used to save screens to or load screens from disk or tape. A discussion of this particular code is beyond the scope of this book.

Screen Saving Routines [lines 16900 to 16980]: These routines are used to preserve the screen display when the screen is used for instructions. Lines 16910 and 16920 call the special routine that was loaded at initialization time to move the graphic and color information from regular memory to the display memory locations. Lines 16960 and 16970 save the graphic and color information from the display memory to the regular memory.

Display Instructions [lines 17000 to 19090]: This begins by calling subroutine 16950 to save the current display. It then calls subroutine 18000 which PRINTs the display instructions via lines 18010 to 19070. The subroutine waits in line 19080 for a keyboard input and then returns. Line 17030 calls the subroutine at 16900 to restore the original display. Lines 17040 to 17080 service a few special characters or return to the main loop at line 50.

Using Templates in the Drawing Program

Once you start using the Line Drawing Program, you will discover that you often want to use the same basic shape again and again. Before long it may occur to you that there must be a better way than to re-type all of these keystrokes. And there is. The term 'template' will be used to generically describe a number of techniques in which a set of characters can be automatically joined together to form a new entity which can be invoked with a single keystroke. The concept is not as difficult as it sounds. Suppose I draw a building made of bricks. This entails many keystrokes if I draw every single line for every single brick. Since the basic brick shape is repeated over and over, it would be easier and faster if an entire brick could be generated with a single stroke. The set of strokes to make a brick that is one unit tall and two

units wide would be:

Y J J N G G

This would generate the brick shape:



If the program had a special key which called the various subroutines required to make these six line drawing segments, plus two additional cursor right movements, this would produce the brick shape. Since the brick drawing starts in the lower left corner and leaves the cursor in the lower right corner, you can draw a whole line of bricks simply by pressing the special key the desired number of times. If the key was Z, for example, pressing the letter Z six times would produce six bricks in a row.

This simple concept of a pre-programmed shape can be expanded to include any other character graphics you desire. This is an example of a fixed template.

Variable Templates

Suppose that you wanted to use a lot of squares and rectangles in your drawing. You could define each possible size of square, rectangle or other shape that you plan to use, but that is tedious. Let the computer do the dull stuff! This drawing program lets you set the length of a side by typing L and a length value, or the width of a side by typing W and a width value. The value must be between 1 and 9. To draw a square with the length you previously set, type S. To draw a rectangle with the height (length) and width that you set, type R. A decision box that uses the set width is called by typing the letter D. You can now quickly and efficiently draw a square or rectangle anywhere on the display that you set your cursor. It sure beats doing it line character by line character. You can see how the basic concept can be extended to other shapes which have variable sizes within them.

Operating Notes

1. The line segment is drawn with the current color of the cursor. If the cursor color is changed, then the line colors will be changed.
2. All of the cursor control movements, including HOME, CLR, INSERT and DELETE are valid during the DRAW mode. The only characters which are NOT available during DRAW mode are the letters that are used to control the Drawing:

T Y U G H J B N M S Z L W R D

These are available by going back temporarily to the Edit mode.

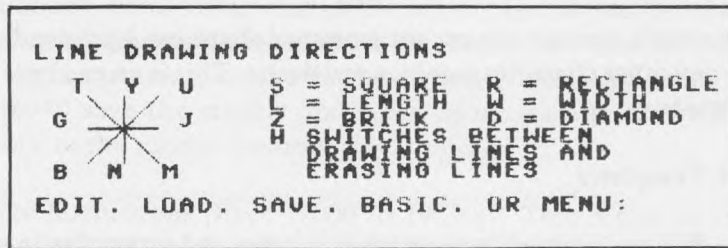
3. The cursor is changed in the DRAW mode to a small square. This represents the lower left corner of a character area and is the location where the horizontal, vertical and one diagonal meet.

4. The F7 key switches between the Draw and Edit modes.

5. All of the normal graphic characters of the computer are available. The special line graphics have used the reverse forms of the characters:

@ A B C D E F G H I J K L M N O.

Typing Reverse A, for example, will generate the horizontal line segment.



```
10 REM SCREEN EDITOR WITH DRAWING PACKAGE
20 GOSUB 20000: REM INITIALIZE
30 PRINT "{CLEAR}";:REM CLEAR SCREEN
40 PRINT BK$:H=0:V=0
50 GOSUB 3000:X=XX
60 C=PEEK(646):GOSUB 1200
100 IF FT=0 THEN FT=1:GOTO 17000
110 GET T$:IF T$="" THEN 110
120 GOSUB 11400:IF A$="" THEN 50
210 IF T$="{F1}" THEN GOTO 17000
220 IF T$="{F2}" THEN GOSUB 1000:GOSUB 14000:GOTO 100
230 IF T$="{F3}" THEN POKE SA,(PEEK(SA)+1) AND 15
240 IF T$="{F4}" THEN POKE SB,(PEEK(SB)+1) AND 15
250 IF T$="{F5}" THEN GOSUB 1000:GOSUB 16950:GOTO 16000
260 IF T$="{F6}" THEN GOSUB 1000:GOSUB 16950:GOTO 15000
300 IF T$=DL$ THEN GOSUB 1000:H=H-1:GOTO 600
310 IF T$=IN$ THEN GOSUB 1000:GOTO 600
320 IF T$=CR$ THEN GOSUB 1000:V=V+1:H=0:GOTO 600
```



```

330 IF T$="{HOME}" THEN GOSUB 1000:GOTO 40
340 IF T$="{CLEAR}" THEN 30
350 IF T$="{UP}" THEN GOSUB 1000:V=V-1:GOTO 600
360 IF T$="{DOWN}" THEN GOSUB 1000:V=V+1:GOTO 600
370 IF T$="{RIGHT}" THEN GOSUB 1000:H=H+1:GOTO 600
380 IF T$="{LEFT}" THEN GOSUB 1000:H=H-1:GOTO 600
400 T=ASC(T$):IF T<32 OR (T>127 AND T<160) THEN PRINT T$,:
      GOTO 60
500 GOSUB 2000
510 PRINT T$;
520 H=H+1
600 IF H=MX AND V=MY THEN H=MX-1:GOTO 660
610 IF H<0 THEN H=MX:V=V-1
620 IF H>MX THEN H=0:V=V+1
630 IF V<0 THEN V=MY:GOTO 600
640 IF V>MY THEN V=0
645 IF FG=1 THEN FG=0:RETURN
650 IF T$=DL$ OR T$=IN$ THEN PRINT T$,:POKE 216,0
660 GOSUB 2000
670 GOTO 50
1000 REM PLOT CHARACTER AND COLOR
1010 X=CH:C=CL
1200 PO=V*RW+H
1210 POKE OG+PO,X
1220 POKE CM+PO,C
1230 RETURN
2000 REM POSITION CURSOR
2010 POKE 781,V:POKE 782,H:POKE 783,0:SYS 65520:RETURN
3000 REM STORE CURRENT CHAR AND COLOR
3010 PO=V*RW+H
3020 CH=PEEK(OG+PO):CL=PEEK(CM+PO)
3030 RETURN
11400 REM PROCESS DRAW COMMANDS
11404 A$=T$
11405 IF A$="{F7}" AND MODE=1 THEN MODE=0:XX=42:X=XX:
      GOSUB 1200:RETURN
11410 IF A$="{F7}" AND MODE=0 THEN MODE=1:XX=123:X=XX:
      GOSUB 1200:RETURN
11420 IF MODE=0 THEN RETURN
11450 REM DECODE KEYS
11451 IF A$="Y" THEN 11481:REM NORTH
11452 IF A$="U" THEN 11482:REM NE

```

```

11453 IF A$="J" THEN 11483:REM EAST
11454 IF A$="M" THEN 11484:REM SE
11455 IF A$="N" THEN 11485:REM SOUTH
11456 IF A$="B" THEN 11486:REM SW
11457 IF A$="G" THEN 11487:REM WEST
11458 IF A$="T" THEN 11488:REM NW
11459 IF A$="H" THEN PEN = -PEN:GOTO 11660
11460 IF A$="S" THEN 11700: REM SQUARE
11461 IF A$="Z" THEN 11710: REM BRICK
11462 IF A$="L" THEN 11720: REM LENGTH
11463 IF A$="W" THEN 11730: REM WIDTH
11464 IF A$="R" THEN 11740: REM RECTANGLE
11465 IF A$="D" THEN 11750: REM DECISION
11470 RETURN
11480 REM ROUTINES TO WRITE 8 DIRECTIONS
11481 GOSUB 11602:GOTO 11501
11482 GOSUB 11608:GOSUB 11502:GOTO 11501
11483 GOSUB 11601:GOTO 11502
11484 GOSUB 11503:GOSUB 11604:GOTO 11502
11485 GOSUB 11503:GOTO 11602
11486 GOSUB 11504:GOSUB 11503:GOTO 11608
11487 GOSUB 11504:GOTO 11601
11488 GOSUB 11504:GOSUB 11604:GOTO 11501
11500 REM MOVE CURSOR
11501 GOSUB 1000:V=V-1:GOTO 11510
11502 GOSUB 1000:H=H+1:GOTO 11510
11503 GOSUB 1000:V=V+1:GOTO 11510
11504 GOSUB 1000:H=H-1
11510 FG=1:GOSUB 600
11520 GOSUB 3000 :RETURN
11600 REM CALCULATE DRAW PATTERNS
11601 BP=1: GOTO 11610
11602 BP=2: GOTO 11610
11604 BP=4: GOTO 11610
11608 BP=8
11610 SX=128
11620 IF (CH> 127 AND CH< 144) THEN SX=CH
11630 IF PEN = 1 THEN CH=(SX OR BP)
11640 IF PEN = -1 THEN CH=(SX AND(-BP-1))
11650 CL=PEEK(646) : GOSUB 1000
11660 A$="": RETURN
11700 REM S = SQUARE

```

```

11701 FOR I=1 TO LN:GOSUB 11481:NEXT
11702 FOR I=1 TO LN:GOSUB 11483:NEXT
11703 FOR I=1 TO LN:GOSUB 11485:NEXT
11704 FOR I=1 TO LN:GOSUB 11487:NEXT
11705 GOTO 11650
11710 REM Z = BRICK 1 X 2
11711 GOSUB 11481:GOSUB 11483:GOSUB 11483:GOSUB 11485:
      GOSUB 11487:GOSUB 11487
11712 GOSUB 11502:GOTO 11502
11720 REM GET LENGTH OF RECTANGLE
11722 GET A$:IF A$="" THEN 11722
11723 LN=VAL(A$):IF LN< 1 AND LN> 9 THEN 11722
11725 GOTO 11650
11730 REM INPUT WIDTH FOR RECTANGLE
11732 GET A$:IF A$="" THEN 11732
11733 WD=VAL(A$):IF WD< 1 AND WD> 9 THEN 11732
11735 GOTO 11650
11740 REM RECTANGLE
11741 FOR I=1 TO LN:GOSUB 11481:NEXT
11742 FOR I=1 TO WD:GOSUB 11483:NEXT
11743 FOR I=1 TO LN:GOSUB 11485:NEXT
11744 FOR I=1 TO WD:GOSUB 11487:NEXT
11745 GOTO 11650
11750 REM DECISION BOX
11751 GOSUB 11488:GOSUB 11488:GOSUB 11482:GOSUB 11482
11752 FOR I=1 TO WD-2:GOSUB 11483:NEXT
11753 GOSUB 11484:GOSUB 11484:GOSUB 11486:GOSUB 11486
11754 FOR I=1 TO WD-2:GOSUB 11487:NEXT
11755 GOTO 11650
14000 REM SELECT DISK/TAPE FOR STORAGE
14010 GOSUB 16950:REM SAVE DISPLAY
14020 PRINT"{CLEAR,BLACK}CURRENT DEVICE = {BLUE}"DV$
14030 PRINT"{BLACK}SELECT D{BLUE}ISK OR {BLACK}T{BLUE}APE "
14040 GET T$:IF T$="" THEN 14040
14050 IF T$="T" THEN DV=1:DV$="TAPE":GOTO 16900
14060 IF T$="D" THEN DV=8:DV$="DISK":GOTO 16900
14070 IF T$=CR$ THEN GOTO 16900
14080 GOTO 14030
15000 REM DUMP TO TAPE OR DISK
15005 GOSUB 16950: REM SAVE SCREEN
15010 INPUT"{CLEAR,BLACK}SAVE FILE NAME";NA$:
      IF LEN(NA$)=0 THEN GOSUB 16900:GOTO 50

```

```

15015 PRINT "{RED}PLEASE BE PATIENT - {BLUE}
      THIS TAKES TIME{BLACK}"
15016 PRINT "          ABOUT {RED}2 1/2 MINUTES{BLACK}!!!"
15020 PR$="@0:"EX$=",S,W"
15030 IF DV=1 THEN GOSUB 15970:PR$="":EX$=""
15040 OPEN 3,(DV),(DV),(PR$+NA$+EX$)
15060 X%=((PEEK(SA) AND 15)*16) OR (PEEK(SB) AND 15):
      PRINT# 3,X%
15070 GOSUB 15810
15080 C%=PEEK(SC+PO) AND 15:S%=PEEK(SG+PO):PRINT# 3,C%:
      PRINT# 3,S%
15100 CLOSE 3:GOSUB 16900:GOTO 50
15800 REM OUTPUT ROUTINE
15810 FOR Y=0 TO MY:A$="":RF=0:CF=-1
15820 FOR X=0 TO MX
15830 PO=RW*Y+X:C%=PEEK(SC+PO) AND 15:S%=PEEK(SG+PO)
15840 IF C%<>CF THEN A$=A$+C$(C%):CF=C%
15850 RV=-(S% AND 128)/128:IF RV AND (NOT(RF)) THEN
      A$=A$+"{RVS}":RF=-1:GOTO 15870
15860 IF (NOT(RV)) AND RF THEN A$=A$+"{RVSOFF}":RF=0
15870 S%=S% AND 127
15880 IF S%<32 THEN S%=S%+64:GOTO 15920
15890 IF S%>63 AND S%<96 THEN S%=S%+128:GOTO 15920
15900 IF S%>95 THEN S%=S%+64
15910 IF S%=32 THEN S%=160
15920 A$=A$+CHR$(S%)
15930 NEXT X
15940 IF (Y=MY) THEN A$=LEFT$(A$,LEN(A$)-1)
15950 PRINT# 3,A$:NEXT Y:RETURN
15960 REM TURN OFF CASSETTE UNTIL KEY IS PRESSED ON KEYBOARD
15970 POKE 0,63:POKE 1,(PEEK(1) OR 32)
15980 GET T$:IF T$="" THEN 15980
15990 POKE 0,47:POKE 1,PEEK(1) AND 223:RETURN
16000 REM LOAD SCREEN FROM TAPE OR DISK
16005 GOSUB 16950: EX$="":IF DV=8 THEN EX$=",S,R"
16010 INPUT"{CLEAR,BLACK}LOAD FILE NAME";NA$:
      IF LEN(NA$)=0 THEN GOSUB 16900:GOTO 50
16020 OPEN 3,(DV),(DV-1),(NA$+EX$)
16040 INPUT# 3,X%:POKE SA,(X% AND 240)/16:POKE SB,X% AND 15
16050 FOR Y=0 TO MY
16060 INPUT# 3,A$
16070 PRINT A$;

```

```

16080 NEXT Y
16085 PRINT "{HOME}";
16090 INPUT# 3,C%,S%:PO=MY*RW+MX:POKE OG+PO,S%:POKE CM+PO,C%
16100 CLOSE 3:PRINT "{BLACK}";: GOTO 50
16900 REM RESTORE SCREEN AND COLOR MEMORY
16910 POKE NP,4:POKE FP,SG/256:POKE TP,OG/256:SYS MV:
      REM RESTORE GRAPHIC MEMORY
16920 POKE NP,4:POKE FP,SC/256:POKE TP,CM/256:SYS MV:
      REM RESTORE COLOR MEMORY
16930 RETURN
16950 REM SAVE SCREEN AND COLOR MEMORY
16960 POKE NP,4:POKE FP,OG/256:POKE TP,SG/256:SYS MV:
      REM SAVE DISPLAY
16970 POKE NP,4:POKE FP,CM/256:POKE TP,SC/256:SYS MV:
      REM SAVE COLOR MEMORY
16980 RETURN
17000 REM DISPLAY INSTRUCTIONS
17010 GOSUB 16950 : REM SAVE DISPLAY
17020 GOSUB 18000 : REM DISPLAY INSTRUCTIONS
17030 GOSUB 16900 : REM RESTORE DISPLAY
17040 IF A$="L" THEN T$="{F5}":GOTO 250
17050 IF A$="S" THEN T$="{F6}":GOTO 260
17060 IF A$="B" THEN PRINT "{CLEAR}":END
17070 IF A$="M" THEN 17200
17080 GOTO 60
17200 REM MENU LOADER
17210 PRINT "{CLEAR}LOAD "+CHR$(34)+"0:MENU"+CHR$(34)+"",8":
      PRINT "{DOWN4}RUN"
17220 POKE 631,19: POKE 632,13: POKE 633,13: POKE 198,3: END
18000 REM DISPLAY INSTRUCTIONS
18010 PRINT "{CLEAR,BLACK}SCREEN EDIT AND DRAW INSTRUCTIONS"
18020 PRINT "{DOWN,RED}KEY FUNCTION"
18030 PRINT "{DOWN,BLACK}F1 {BLUE}DISPLAY THESE INSTRUCTIONS"
18040 PRINT "{BLACK}F2 {BLUE}SELECT DISK OR TAPE DEVICE"
18050 PRINT "{BLACK}F3 {BLUE}CHANGE SCREEN COLOR"
18060 PRINT "{BLACK}F4 {BLUE}CHANGE BORDER COLOR"
18070 PRINT "{BLACK}F5 {BLUE}LOAD DISPLAY"
18080 PRINT "{BLACK}F6 {BLUE}SAVE DISPLAY"
18090 PRINT "{BLACK}F7 {BLUE}ENTER/EXIT LINE DRAWING MODE"
19000 PRINT "{DOWN,BLACK}LINE DRAWING DIRECTIONS"
19010 PRINT "{DOWN,BLACK} T Y U S{BLUE} = SQUARE
      {BLACK}R{BLUE} = RECTANGLE"

```

```

19020 PRINT "{BLACK} {RVS}D{RVSOFF} {RVS}BH{RVSOFF,SPC6}
      L{BLUE} = LENGTH {BLACK}W{BLUE} = WIDTH"
19030 PRINT "{BLACK} G {RVS}AEKA{RVSOFF} J      Z{BLUE} =
      BRICK {BLACK}D{BLUE} = DIAMOND"
19040 PRINT "{BLACK} {RVS}HF{RVSOFF,SPC7}
      H {BLUE}SWITCHES BETWEEN"
19050 PRINT "{BLACK} {RVS}H{RVSOFF} {RVS}BD{RVSOFF,SPC8,
      BLUE}DRAWING LINE AND"
19060 PRINT "{BLACK} B N M {BLUE}ERASING LINES"
19070 PRINT "{DOWN,BLACK}E{BLUE}DIT, {BLACK}L{BLUE}OAD,
      {BLACK}S{BLUE}AVE, {BLACK}B{BLUE}ASIC, OR {BLACK}
      M{BLUE}ENU:"
19080 GET A$:IF A$="" THEN 19080
19090 PRINT "{HOME}": RETURN
20000 REM MAIN INITIALIZATION
20010 POKE 51,0:POKE 52,48:POKE 55,0:POKE 56,48:
      REM RESERVE MEMORY
20020 PRINT "{CLEAR,BLACK} GENERATING GRAPHIC CHARACTERS"
20030 NP=251:FP=252:TP=254:RC=53248:PC=12288:
      SC=16384:SG=20480
20040 READ MS:MV=MS :REM MEMORY START
20050 READ X:IF X<> 256 THEN POKE MS,X:MS=MS+1:GOTO 20050
20060 REM PROGRAM TO COPY ROM TO RAM
20070 DATA 49664
20080 DATA 166,251,160,0,132,251,132,253,177,
      251,145,253,200,208,249,230
20090 DATA 252,230,254,202,208,242,96,256
20100 POKE 56334,PEEK(56334) AND 254
20110 POKE 1,PEEK(1) AND 251
20120 REM MOVE ROM CHARS TO RAM
20130 POKE NP,8:POKE FP,RC/256:POKE TP,PC/256:SYS MV
20140 POKE 1,PEEK(1) OR 4
20150 POKE 56334,PEEK(56334) OR 1
20160 POKE 53272,(PEEK(53272)AND 240)+12
20170 PC=12288+(128*8)
20180 FOR I=PC TO PC+(16*8)-1:READ A: POKE I,A:NEXT
20190 REM CHARACTER PATTERNS FOR DRAW
20200 DATA 0,0,0,0,0,0,0,0
20201 DATA 0,0,0,0,0,0,0,255
20202 DATA 128,128,128,128,128,128,128,128
20203 DATA 128,128,128,128,128,128,128,255
20204 DATA 128,64,32,16,8,4,2,1

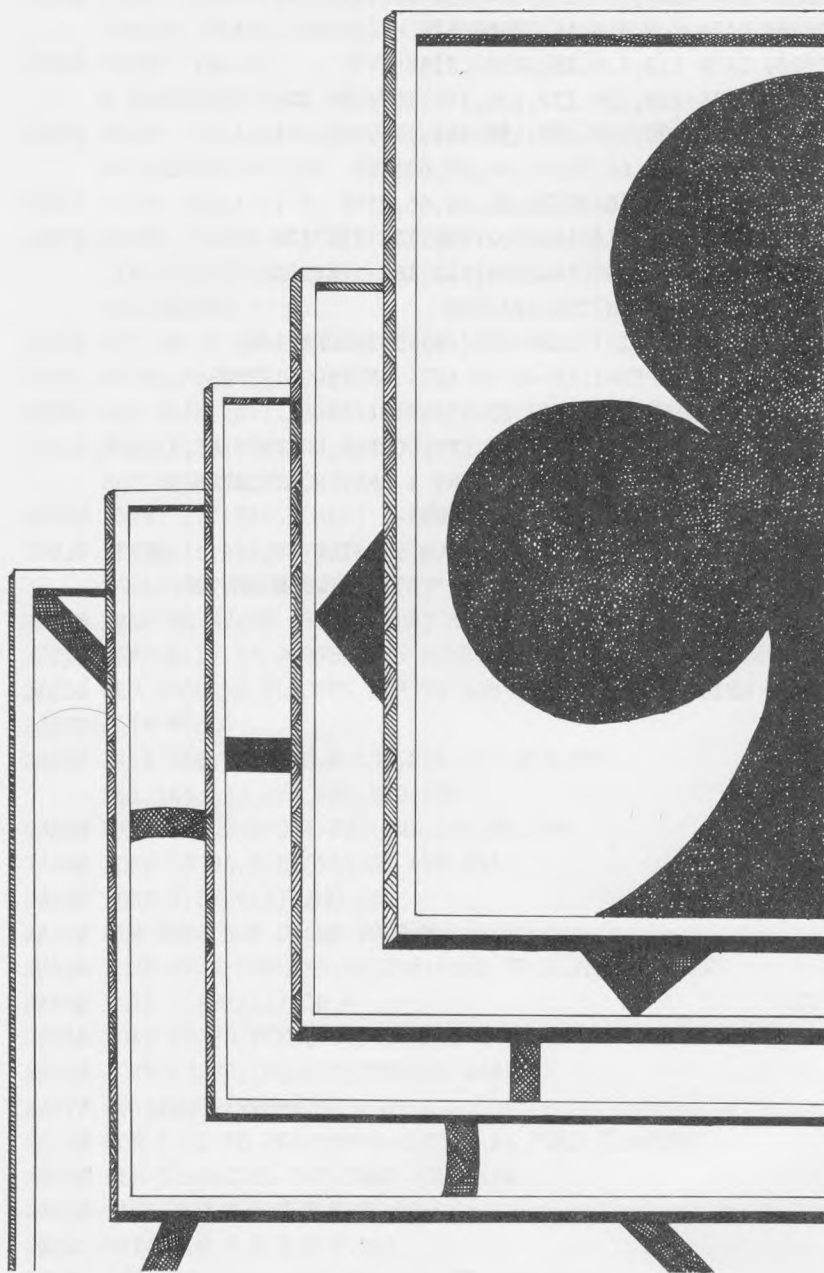
```



```

20205 DATA 128,64,32,16,8,4,2,255
20206 DATA 128,192,160,144,136,132,130,129
20207 DATA 128,192,160,144,136,132,130,255
20208 DATA 1,2,4,8,16,32,64,128
20209 DATA 1,2,4,8,16,32,64,255
20210 DATA 129,130,132,136,144,160,192,128
20211 DATA 129,130,132,136,144,160,192,255
20212 DATA 129,66,36,24,24,36,66,129
20213 DATA 129,66,36,24,24,36,66,255
20214 DATA 129,194,164,152,152,164,194,129
20215 DATA 129,194,164,152,152,164,194,255
20300 REM MISC INITIALIZATION
20310 CR$=CHR$(13):DL$=CHR$(20):IN$=CHR$(148)
20320 MODE=0:PEN=1:XX=42:OG=1024:CM=55296:FT=0
20340 RW=40:MX=39:MY=24:SB=53280:SA=53281
20350 C$="{BLACK,WHITE,RED,CYN,PURPLE,GREEN,BLUE,YELLOW,
      ORANGE,BROWN,LT RED,GREY 1,GREY 2,LT GREEN,
      LT BLUE,GREY 3}":L=LEN(C$)-1
20360 DIM C$(L):FOR I=0 TO L:C$(I)=MID$(C$,I+1,1):NEXT
20370 R$(0)="{RVSOFF}":R$(1)="{RVS}":X=160:CH=32
20380 POKE SB,2:POKE SA,1
20390 DV=8:DV$="DISK"
20400 RETURN

```



Sorting

Information on Five Sorting Methods

```

      BAR SORTER
WHICH SORT?      CMP SWP
 1 BUBBLE SORT
 2 INSERT SORT
 3 SELECT SORT
 4 SHELL  SORT
 5 QUICK  SORT
 6 EXIT TO BASIC
 7 EXIT TO DISK MENU
CHOOSE

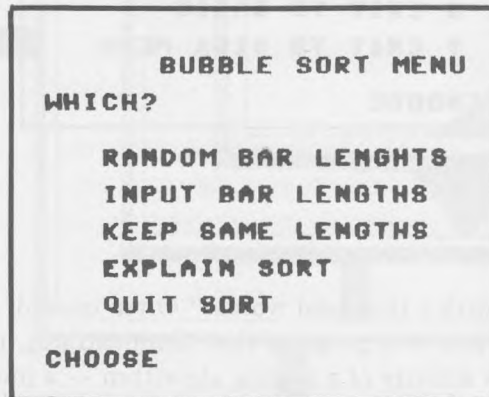
```

"A picture is worth a thousand words." Often quoted, but how often implemented? Here is a program that demonstrates, on the screen, behind-the-lines activity of a sorting algorithm — a much taught and well-documented topic.

Have you ever wondered how a computer keeps all of the information stored on it in proper order? Have you ever wanted to sort a long list into alphabetic or numerical order? Here are the answers to everything you need to know about computerized sorting, and then some.

Operating Instructions

1. Execute BAR SORT from the disk menu.
2. A page explaining BAR SORT will appear. After reading it, hit any key to continue. When the Bar Sort menu appears, press 1 for the bubble sort.
3. When the Bubble Sort menu appears, press E for a short explanation. Hit any key to get back to the Bar Sort menu.
4. Choose R for random numbers and sit back and watch as your C64 puts the bars into descending order.
5. To stop the action, press the spacebar; to continue, press it again. Hit X to eXecute one step-at-a-time. Hit S to slow down action. Hit F to increase the sort speed. Hit Q to quit the sort and return to the sort menu.
6. To become acquainted with the program, select each of the other sorts by number from the menu, then read on.



Background Information

To begin, we present some background of sorting theory, including information on five sorting methods: bubble sort, insertion sort, selection sort, Shell sort, and quick sort. A complete program is included for each method.

Apart from specific details for each of the algorithms, the theory connected with sorting has to do with efficiency. The sorts listed above are more or less in ascending order of efficiency, bubble sort being the slowest. Bubble sort, however, is the easiest to understand, so we will spend the most time on it.

When discussing sorting, it is necessary to define a few mathematical terms; please bear with us and we will make this as painless as possible. The number of items to be sorted (the total number of objects in a group with which the program deals) is called "N". Number of elements to sort = N.

To talk about the efficiency of any algorithm, you need to have some measurable quantities. For sorting algorithms, concentrate on the number of comparisons and the number of interchanges. A comparison occurs whenever one element of the group to be sorted is compared to another element. The value of that second element can be another member of the group, or it can be a very large or very small number chosen specifically to sort the group above or below. Thus, a statement such as

```
IF A(I)=A(I+1) THEN ...
```

is an example of a comparison of one element in the array A (in this case the *I*th element) to the next higher one (*I*). Also, a statement like

```
IF A(I)=MAX THEN ...
```

is a comparison of the element A(*I*) to a large number, MAX.

An interchange occurs whenever a member of the group is moved from one place in the computer's memory to another and, possibly, some other element takes its place. The classic interchange can be described by the sequence of statements,

```
TEMP=A(I)  
A(I)=A(J)  
A(J)=TEMP
```

assuming that *I* is not equal to *J*. Not all sorting algorithms use this form, but usually there is an easily identifiable interchange step whose repetition you can count.

Counting the number of comparisons and interchanges that take place during program execution helps you measure the efficiency of that algorithm. In addition, there is computer time spent in loop control and subroutine calls that is difficult to estimate except by empirical observation.

Now that you are armed with a few terminological weapons, we shall present some of the sorting buzz phrases. Do not be alarmed if they sound Greek. They are background information only and not necessary to the operation of the program itself.

If you count the number of comparisons made during the sorting of a group by a particular algorithm, it may turn out that N^2 (N squared) comparisons are needed. This means that "on the order of" N times N comparisons are needed to sort an array of size N using that particular algorithm.

Another type of algorithm may use a different amount of comparisons called $N * (\log_2 N)$ — N log N. This is N multiplied by the log base 2 of N, a smaller number than the previous example, take our word for it. Why should you be interested in these numbers, and what is the significance of the difference between them?

Look at table 1. It shows values for N, N^2 , $(\log_2 N)$ and $N * (\log_2 N)$. Assuming that the overhead from one algorithm to the next is relatively constant, you can see that there is an ever increasing difference between N^2 and N log N. To make the comparison more concrete, assume that a comparison costs .001¢, and that you need to sort an array containing 1,048,576 numbers. Using an N^2 sort will cost \$10,995,116.27, while using an N log N sort will cost \$209.72. Of course, a single comparison on today's big computers costs considerably less than .001¢. But even at .0000001¢ per comparison (a rate of 1¢ per 10,000,000 comparisons), the cost will be 2¢ for an N log N sort and \$1,099.51 for an N^2 sort! With that kind of differential, you can see why a commercial sorting program would not use an N^2 approach.

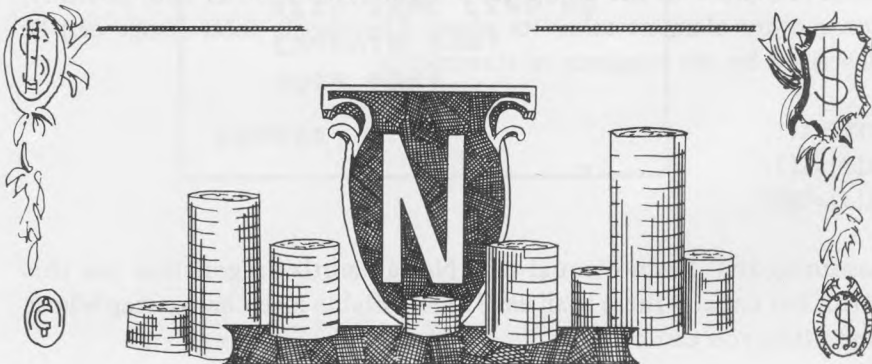


Table 1
Powers of 2 and Logs of Powers of 2

N	N ²	Log ₂ N	N Log ₂ N
64	4,096	6	384
128	16,384	7	896
256	65,536	8	2,048
512	262,144	9	4,608
1,024	1,048,576	10	10,240
2,048	4,194,304	11	22,528
4,096	16,777,216	12	49,152
8,192	67,108,864	13	106,496
16,384	268,435,456	14	229,376
32,768	1,073,741,824	15	491,520
65,536	4,294,967,296	16	1,048,576
131,072	17,179,869,184	17	2,228,224
262,144	68,719,476,736	18	4,718,592
524,288	274,877,906,944	19	9,961,472
1,048,576	1,099,511,627,776	20	20,971,520

=====> 1 BUBBLE SORT <=====

METHOD BUBBLE SORT COMPARES ADJACENT
BARS AND INTERCHANGES THEM IF
THEY ARE OUT OF ORDER.
THUS BARS ARE 'BUBBLED' UP TO
TO THEIR PROPER POSITION.

=====> 2 INSERT SORT <=====

METHOD INSERT SORT HOLDS THE NEXT
BAR IN TEMP WHILE COMPARISONS
ARE MADE FROM THE FIRST BAR
TO THE CURRENT BAR TO FIND THE
PROPER POSITION IN WHICH TO
INSERT TEMP.

=====> 3 SELECT SORT <=====

METHOD SELECT SORT GOES THROUGH THE
BARS PLACING THE LARGEST BAR
FOUND AT THE END OF THE LIST
AND STARTS AT THE TOP AGAIN
UNTIL ALL BARS ARE IN ORDER.

=====> 4 SHELL SORT <=====

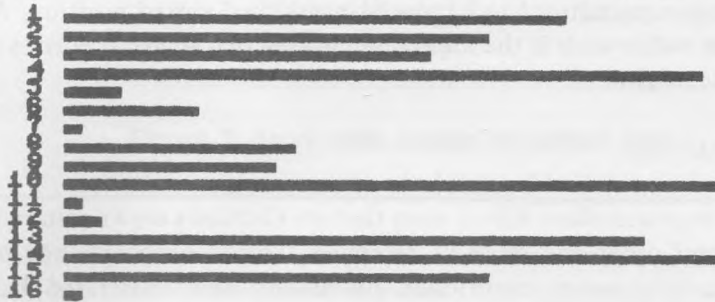
METHOD SHELL SORT SORTS BY THE SHELL-
METZNER METHOD WHICH SEARCHES
THE BARS BY DECREASING INTER-
VALS, FINE TUNING THE SEARCH
UNTIL ALL BARS ARE IN ORDER.

=====> 5 QUICK SORT <=====

METHOD QUICK SORT INTERCHANGES
NON-ADJACENT BARS BY
PARTITIONING THE BARS INTO
SMALLER GROUPS, SORTING
EACH GROUP INDEPENDENTLY.

RANDOM BARS

BUBBLE SORT CMP 3 SWP 3 TEMP = 3



HIT SPACE FOR TO STOP/START
 FOR SLOW OR STEP FOR FAST MOTION
 Q TO QUIT TO BUBBLE SORT MENU

SORTED BARS

BUBBLE SORT CMP122 SWP 67 TEMP = 3



HIT ANY KEY

Bubble Sort

This algorithm is probably the most widely known and easiest to understand of all sorting algorithms. Assume that N elements, which are denoted by $A(1)$, $A(2)$, $A(3)$, ... $A(N)$, are to be arranged in ascending order. This sort operates by sweeps through the array, causing various elements to "bubble up" in the process. For each pass, at least one element is guaranteed to be placed in its final sorted position. At the heart of each sweep is the idea of comparing two adjacent entries in the array:

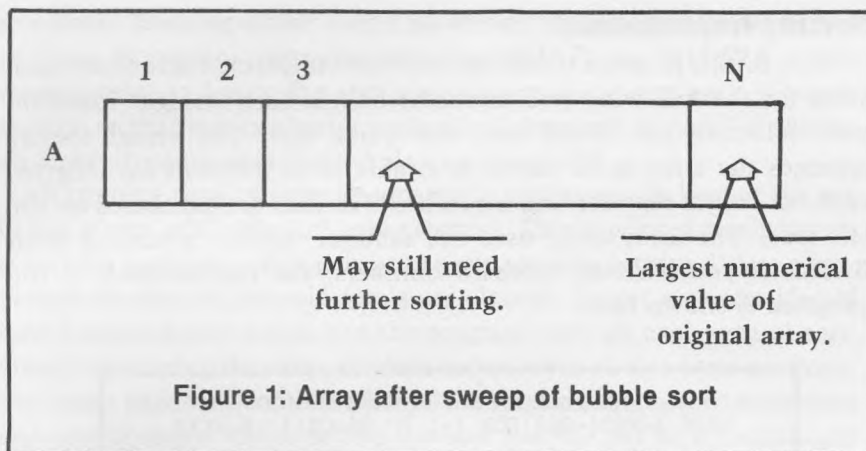
$$A(I) > A(I+1)$$

If $A(I)$ is greater than $A(I+1)$, then the two elements are known to be out of correct order and must be swapped. This is accomplished by the classic interchange mentioned previously and illustrated here in BASIC. If $A(I)$ is less than or equal to $A(I+1)$, then no swap is required. Otherwise, $A(I)$ is put into a TEMPorary location, $A(I)$ is set to equal $A(I+1)$, and finally $A(I+1)$ is set to equal TEMP which is the old $A(I)$.

```
100 IF A(I) <= A(I+1) THEN 140
110 TERM=A(I)
120 A(I)=A(I+1)
130 A(I+1)=TERM
140 ...
```

Now consider the number of comparisons necessary to arrange the array into sorted order. First, you can make no assumptions about the original array with respect to the relative positions of the members. You must successively compare each element to the next higher one until the end is reached. That is, you compare $A(1)$ to $A(2)$ and $A(2)$ to $A(3)$, until you compare $A(N-1)$ to $A(N)$. Positions of various elements will be swapped (using the code in figure 1); in particular, the largest element in the original array will end up in the $A(N)$ position after the sweep is completed. To be convinced that this is true, ask yourself, "If the largest value is originally in $A(I)$, then with what other array elements will it be swapped?"

After one sweep of the array, you have a picture such as that shown in figure 1. You have partially sorted the original array. In fact, the



only sorted element is the last. On each subsequent sweep through the array, one additional element is sorted. The last sweep is when $A(1)$ and $A(2)$ are interchanged, if necessary, to complete the sort. Since two elements are arranged by this swap, and one by each previous sweep, it requires $N - 1$ sweeps through the array to ensure that all elements are sorted, as shown in the listing below.

```

10 FOR I=1 TO N-1
20 FOR J=1 TO N-1
30 IF A(J) < A(J+1) THEN 70
40 TEMP=A(J)
50 A(J)=A(J+1)
60 A(J+1)=TEMP
70 NEXT J
80 NEXT I

```

Since each sweep makes $N - 1$ comparisons, the total number of passes necessary to completely sort the array is $(N - 1)^2$. In the case of large numbers, this is relatively close to N^2 . This inefficiency of the bubble sort is compensated for by its simplicity, especially in understanding how it works. Consequently, it is quite acceptable for sorting tasks involving only small values of N .

Sorting Implemented

The BASIC program in listing 1 provides implementations of visual sorts for the following five methods: bubble sort, straight insertion sort, selection sort, Shell sort, and quick sort. The visual display arranges the array to be sorted as a table of 16 different bar lengths, each bar length representing a number. The bars are numbered on the left side. The array table uses the random number generator from BASIC. If you wish no repeated numbers, the modification to the program is shown here:

```
6000 A=RND(-RS):FOR I=1 TO 36:C%(I)=0:NEXT
6010 FOR I=1 TO LL
6020 A=RND(1)*36+1:IF C%(A)<>0 THEN 6020
6030 C%(A)=1:B%(I)=A:NEXT
6040 RETURN
12005 DIM C%(36)
```

All values generated are positive and less than 36. This is due to horizontal space constraints of the display and does not reflect any inherent limitations in the algorithms themselves.

Each of the sorting algorithms are carried out in the program. As the array is sorted, the values displayed on the screen reflect the changes taking place internally. Various colors are used to highlight this. Each time a comparison is made, the bar being examined is displayed in purple. If it becomes a temporary storage number, it is switched to red. If it is swapped with another number, it changes to green. In any case, when the program is through with that number, the bar returns to black. Notice that some sorts hold temporary values longer than others and a bar may remain red for some time. You can get a good idea of how each algorithm does its job just by watching the pattern of colors on the screen. In addition, each sort prints on the top and bottom borders of the display some additional information about what is happening.

The program begins with a menu and asks for your choice of sort. After you have entered your selection, you may choose to have a new random pattern, enter the pattern of your choice, or repeat the previous random pattern. This is so that you can see the difference in the various sorts using the same pattern. You cannot repeat a pattern until you have sorted a new pattern.

To stop the program at any point, press the spacebar. To restart the sort, again press the spacebar. Pressing X will eXecute one step-

at-a-time. Pressing either F or S to restart the program or while the program is running puts the sort into 'F(AST)' or 'S(LOW)' mode, respectively, to hurry the sort along, or make it sort each element slowly, so that you can better understand the steps involved in sorting an array. To quit from the running sort press Q.

A running total of comparisons and interchanges is kept at the top of the screen while the sort is in progress. This is printed on the menu screen for each sort while using the same random pattern. When a new pattern is selected, the old totals are cleared. To get a clearer idea of how long each sort takes, run the program through each type of sort, always specifying the same random order. After all five sorts are done, the menu page will contain a list of the running totals of comparisons and interchanges. Examine this list and you will get an accounting of how long it took each sort to complete its work. An interchange is worth approximately two comparisons, timewise. Note that if you quit in the middle of a sort the number of comparisons and interchanges for that particular sort will not be saved.

Telephone Book

There is an additional sorting program on the C64 disk. Phone Book is an example of a bubble sort program that sorts something useful — in this case a telephone log. The sort alphabetizes your personal telephone book. When run, the program prompts you to choose an option:

1. Create a new file (you can enter up to 100 names and telephone numbers).
2. Add, change, or delete names (you can alter a file that you have already created).
3. Load a file already saved on tape or disk (you can load in your telephone book to look up numbers or make changes).
4. Save file to tape or disk (you can put your tape recorder in record mode and save your telephone book on tape).
5. Sort file (you can sort the file into alphabetical or numerical order).
If you want the sorted file saved alphabetically, you must resave it at this time.
6. Select the current device as disk or tape (default is disk)
7. Exit to basic.
8. Exit to the disk menu.

A sample program to demonstrate how sorting could be used in a useful manner is also included. Execute PHONE BOOK from the disk.

Since the *Mastering Your Commodore 64* disk is permanently write protected, you must have another formatted disk to use if you wish to save the telephone files to disk.

Since we don't have any files saved on the disk, type '1' to create a new file. When asked for the name of the file, press return to accept the default name of 'TELEPHONEFILE'. At this time you could enter up to 100 different people and their telephone numbers.

For this trial run, put in your name, last name first as prompted. Do not put a comma after your last name, as that would indicate to the C64 that you have two inputs on that line. After your first name, press return. Then enter your telephone number in the form 000-000-0000. Press return after that entry also. Enter one other person and number at this time. When prompted to enter name again, type 'DONE' in upper or lower case and press return. This will return you to the main menu. You have now created a short telephone file of two names. It is in the computer's memory but not yet saved.

There are several options listed on the menu that you can now perform. Choosing the second menu selection will allow you to thumb through the file, one name at a time. After each name and number is displayed you may DELETE the entry by pressing 'D', CHANGE all or part of the entry by pressing 'C', or CONTINUE through the names by pressing return. During the CHANGE operation a return signifies that the entry is acceptable, and typing new information replaces the old entry with the new.

The LOAD and SAVE options work similarly to the PLAYER load and save. If you have a question about that operation, read about tape or disk files in that chapter.

The SORT selection will perform a bubble sort on either the name field or the phone number field. You also have the option of watching the sort or having the sort done more quickly and returning you to the menu when finished. When in the visual sort mode, you must press the space bar for each cycle through the sorting algorithm. This is to slow the sort down enough so that you can see the interchanges.

Exit returns you to BASIC.

The Bar Sorter Program

Line 20 POKes the border color light blue and the screen color white.

Lines 100-170 print the explanation page when the program is first executed.

The subroutine at line 1000 controls all the screen printing during the sorting operation. CD\$ is a string of cursor downs and controls the

vertical displacement of the print line. Each line is printed A lines down from the top of the screen. B\$ is a series of C64 character-graphic half blocks that are used to make up the bar lines. The amount of line printed (B%(A)) corresponds to the random number assigned to that position in the array. C\$ is a line of spaces and is used to fill out the rest of the line, which erases any previous lines that might have been longer in length. Line 1070 prints the CoMPares, SWaPs, and TEMP bar number at the top of the screen.

The input subroutine at line 2000 looks for certain key characters that tell the program how to proceed. A Q is the quit character and causes the program to stop the sort and GOTO the sort menu without saving the current statistics. A space is the indicator to pause in the sort until another key is pressed. The X eXecutes one step-at-a-time. The F character (for FAST) sets the delay timer to 1, the smallest possible delay. S (for SLOW) sets the delay timer to 800. The subroutine at line 2060 is also called to read a key from the keyboard during menu choices.

The subroutine at line 3000 prints the compare and swap totals next to the menu names. It uses the subroutine at line 5000 to make sure that the numbers are all three characters long. The subroutine at line 4000 clears the compare and swap totals when changing random patterns. Line 6000 initializes the array with random numbers between 1 and 36.

Bubble sort is at line 7000. The compare is made at line 7020 and swap, if necessary, at line 7050. The flag is set every time there is a swap. If the sort goes through the entire loop without a swap, then it is finished and the flag returns unset and, in line 7030, sets the loop counter to MAX, which finishes the sort.

The straight insertion sort subroutine begins at line 8000. The compare is made in line 8020 and elements are swapped in lines 8040-8050 when necessary. T is used to hold each succeeding element while the compares are made back to the beginning of the array to see where the current T line is to be inserted.

The selection sort saves the largest line currently found in A. The compare is at line 9040 and swapping is done at line 9080. A is used to store the largest element currently found, which is moved to the bottom position at the end of each loop, with the bottom position progressing toward the beginning of the array.

The subroutine at line 10000 sorts by the Shell-Metzner method, which searches the array by decreasingly smaller intervals. The SP%() array contains the intervals that are used in the loop counters. The STEP increments of SP are 10, 6, 4, 2, and 1, which increasingly fine-tune the sort until completed. The compare is done in line 10040 and

swapped, when necessary, at 10080.

Quick sort is located at line 11000 and is the most complicated sort (see shaded box). Compares are made not only of the elements, but also of the loop counters. In addition to swaps of the elements, one or more of the elements may be stored on a temporary stack until a suitable place in the array is found to put them. It is also the quickest of the sorts.

Initialization is done at line 12000. Lines 12080-12110 print the main sort menu, while lines 12150-12200 print the individual sort menu.

The directions for controlling the sort are printed starting at 12300, and the sort title and bars are printed in line 12340. Line 12350 calls the appropriate sort routine. When the sort is completed lines 12360-12380 clear the bottom of the screen and wait for the user to hit a key. The program then returns to the main sort menu.

Lines 13000-13030 input bar lengths from 1-36.

The explanation of the sorts is done in the subroutine starting at line 14000. The information for the explanations starts at line 15000. The disk menu loader begins at line 20000.

```
10 REM      BAR SORTER
20 POKE 53281,1:POKE 53280,14
100 PRINT"{CLEAR,BLUE,DOWN2}          BAR SORTER
110 PRINT"{DOWN4}BAR SORT IS INTENDED FOR AMUSEMENT
    AND      EDUCATION.
120 PRINT"{DOWN2}BAR SORT PUTS BARS INTO ASCENDING ORDER
130 PRINT"  USING DIFFERENT SORTING ALGORITHMS.
140 PRINT"{DOWN2}FIRST, CHOOSE FROM FIVE DIFFERENT SORTS.
150 PRINT"{DOWN}LATER, INCORPORATE USEFUL SORTS INTO
160 PRINT"  YOUR PERSONAL APPLICATIONS.
170 PRINT"{DOWN2,SPACE9,RED}HIT ANY KEY TO CONTINUE{BLUE}":
    GOSUB 2060:GOTO 12000
1000 REM    PRINT ROUTINE
1010 PRINT"{HOME}":IF A> LL THEN RETURN
1020 CL=PEEK(646):PRINT LEFT$(CD$,A-1);
1030 M$=" ":IF A> 9 THEN M$=""
1040 PRINT"{DOWN}";M$;A;LEFT$(B$,B%(A));LEFT$(C$,36-B%(A))
1050 A$=STR$(CO):GOSUB 5000:CO$=A$:A$=STR$(SW):
    GOSUB 5000:SW$=A$
1060 A$=STR$(TT):GOSUB 5000
1070 PRINT"{HOME,RIGHT14,PURPLE}CMP"CO$" {GREEN}SWP"
    SW$"{BLACK}"" {RED}TEMP = A$"{BLACK}";
```

```

1080 PRINT "{HOME}" CD$ "{UP2} ";
1100 IF CL < > 4 THEN 1200
1110 PRINT "{PURPLE}COMPARING";
1120 GOTO 1220
1200 IF CL < > 5 THEN 1240
1210 PRINT "{GREEN}SWAPPING ";
1220 A$=STR$(I)
1225 GOSUB 5000:PRINT A$ " WITH";
1230 A$=STR$(J):GOSUB 5000:PRINT A$;
1240 GOSUB 2000:RETURN
2000 REM SCREEN INPUT
2010 GET A$:IF A$=" " THEN 2060
2020 IF A$="Q" THEN 12150
2030 IF A$="F" THEN QT=1
2040 IF A$="S" THEN QT=500
2042 IF A$="X" THEN GOTO 2070
2044 IF X$="X" THEN GOTO 2010
2050 FOR WW=1 TO QT:NEXT:GOTO 2070
2060 GET A$:IF A$="" THEN 2060
2070 X$=A$:RETURN
3000 IF CO(I)=0 THEN 3040
3010 PRINT "{HOME}" LEFT$(CD$,2*I+4):PRINT "{UP,RIGHT16,RVS}";
3020 A$=STR$(CO(I)):GOSUB 5000:PRINT A$ " ";:A$=STR$(SW(I))
3030 GOSUB 5000:PRINTA$ "{RVSOFF}";
3040 PRINT: RETURN
4000 FOR I=1 TO 5:CO(I)=0:NEXT:RETURN
5000 IF LEN(A$)> 3 THEN A$=RIGHT$(A$,3)
5010 IF LEN(A$)< 3 THEN A$=" "+A$
5020 RETURN
6000 A=RND(-RS):FOR I=1 TO LL:B%(I)=RND(1)*36+1:NEXT:RETURN
7000 FOR L=1 TO LL:F=0:FOR J=1 TO LL-L:CO=CO+1
7010 A=J+1:I=J+1:PRINT "{PURPLE}";:GOSUB 1000:
PRINT "{BLACK}";:GOSUB 1000
7020 IF B%(J)> B%(J+1) THEN GOSUB 7050:A=J:
GOSUB 1000:A=J+1:I=A:GOSUB 1000
7030 NEXT J:IF F=0 THEN L=LL
7040 NEXT L:RETURN
7050 A=J:TT=A:PRINT "{RED}";:GOSUB 1000:T=B%(J):
B%(J)=B%(J+1):B%(J+1)=T
7060 SW=SW+1:A=J+1:I=A:PRINT "{GREEN}";:GOSUB 1000:
PRINT "{BLACK}";:F=1:RETURN
8000 FOR K=2 TO LL:T=B%(K):A=K:TT=A:PRINT "{RED}";:GOSUB 1000

```



```

8010 FOR J=K-1 TO 1 STEP -1:A=J:I=K:CO=CO+1:
      PRINT:PRINT"{PURPLE}";:GOSUB1000
11040 A=J:CO=CO+1:GOSUB 1000:PRINT"{BLACK}";:GOSUB 1000
11045 A=I:GOSUB1000:GOTO11030
11050 I=I+1:IF I> LL THEN 11070
11055 A=J:PRINT"{PURPLE}";:GOSUB1000
11060 A=I:CO=CO+1:PRINT"{PURPLE}";:GOSUB 1000:
      PRINT"{BLACK}";:GOSUB 1000
11070 IF B%(I)>=V THEN 11090
11080 GOTO 11050
11090 IF J<=I THEN 11130
11100 A=I:SW=SW+1:PRINT"{GREEN}";:GOSUB 1000:A=J:
      PRINT"{GREEN}";:GOSUB 1000:PRINT"{BLACK}";
11110 T=B%(I):B%(I)=B%(J):B%(J)=T
11120 A=I:GOSUB 1000:A=J:GOSUB 1000:GOTO 11030
11130 B%(P)=B%(J):B%(J)=V:SW=SW+1
11135 II=I:I=P
11140 A=P:GOSUB 1000:A=J:PRINT"{GREEN}";:GOSUB 1000:
      PRINT"{BLACK}";:GOSUB 1000
11145 I=II
11150 IF J-P<QQ-J THEN 11170
11160 S%(TP+1)=P:S%(TP+2)=J-1:P=J+1:GOTO 11180
11170 S%(TP+1)=J+1:S%(TP+2)=QQ:QQ=J-1
11180 TP=TP+2:GOTO 11010
11190 IF TP=0 THEN 11210
11200 QQ=S%(TP):P=S%(TP-1):TP=TP-2:GOTO 11010
11210 RETURN
12000 DIM B%(21),SP%(5),CO(5),SW(5):LS=16
12010 SP%(1)=10:SP%(2)=6:SP%(3)=4:SP%(4)=2:SP%(5)=1
12020 B%(21)=35:Q$(1)="BUBBLE SORT":
      Q$(2)="INSERT SORT":Q$(3)="SELECT SORT"
12030 Q$(4)="SHELL SORT":Q$(5)="QUICK SORT":RS=TI
12040 B$="{I40}"
12050 CD$="{DOWN21}":BL$="{SPACE8}":
      MD$="{DOWN5,RVS}METHOD{RVSOFF}":
12060 C$="{SPACE40}"
12070 PRINT"{CLEAR,BLUE}":LL=16
12080 PRINT"{CLEAR,DOWN,RIGHT8,RVS}BAR SORTER{RVSOFF}":
      PRINT"{DOWN2}WHICH SORT? {RVS}CMP SWP{RVSOFF}"
12090 FOR I=1 TO 5:PRINT"{DOWN,RVS}"I"{RVSOFF}"Q$(I);:
      GOSUB 3000:NEXT
12100 PRINT"{DOWN,RVS} 6{RVSOFF} EXIT TO BASIC":
      PRINT "{DOWN,RVS} 7{RVSOFF} EXIT TO DISK MENU"

```



```

12110 PRINT"{DOWN2,RED}CHOOSE{BLUE}":CO=0:SW=0:GOSUB 2060
12120 Q=VAL(A$):IF Q<10RQ>7 THEN 12070
12130 PRINT"{CLEAR}":IF Q=6 THEN END
12140 IF Q = 7 THEN 20010
12150 PRINT"{CLEAR,DOWN}          {RVS}"Q$(Q)" MENU"
12160 PRINT"{HOME,DOWN3}WHICH?":
      PRINT"{DOWN3,RIGHT3,RVS}R{RVSOFF}ANDOM BAR LENGTHS"
12170 PRINT"{DOWN,RIGHT3,RVS}I{RVSOFF}NPUT BAR LENGTHS"
12180 PRINT"{DOWN,RIGHT3,RVS}K{RVSOFF}EEP SAME LENGTHS"
12190 PRINT"{DOWN,RIGHT3,RVS}E{RVSOFF}XPLAIN SORT"
12200 PRINT"{DOWN,RIGHT3,RVS}Q{RVSOFF}UIT TO SORT MENU":
      PRINT"{DOWN2,RED}CHOOSE{BLUE}":GOSUB 2060
12210 IF A$="R" THEN RS=TI:GOSUB 6000:GOSUB 4000:GOTO 12300
12220 IF A$="K" AND B%(1)>0 THEN GOSUB 6000:GOTO 12300
12230 IF A$="I" THEN GOSUB 13000:GOTO 12300
12240 IF A$="E" THEN GOSUB 14000:GOTO 12150
12250 IF A$="Q" THEN GOTO 12080
12260 GOTO 12150
12300 PRINT"{CLEAR}":FOR I=1 TO LL:PRINT"":NEXT
12310 PRINT"{CLEAR,DOWN21}HIT {RVS}SPACE BAR{RVSOFF}
      TO STOP/START"
12320 PRINT"      {RVS}S{RVSOFF} FOR SLOW OR {RVS}F{RVSOFF}
      FOR FAST MOTION"
12325 PRINT"      {RVS}X{RVSOFF} TO XECUTE STEP BY STEP"
12330 PRINT"      {RVS}Q{RVSOFF} TO QUIT TO "Q$(Q)" MENU"
12340 PRINT"{HOME,BLUE,RVS}"Q$(Q)"{BLACK,RVSOFF}";:
      FOR I=1 TO LL:A=I:GOSUB 1000:NEXT
12350 ON Q GOSUB 7000,8000,9000,10000,11000
12360 PRINT"{HOME}"CD$"{UP5}":
      FOR I=1 TO 3:PRINT"{SPACE30}":NEXT
12370 PRINT"{RIGHT4}      {RED}HIT ANY KEY{BLUE,SPACE18}"
12380 PRINT"{SPACE33}"
12385 PRINT"{SPACE33}"
12390 GOSUB 2060:CO(Q)=CO:SW(Q)=SW:TT=0:GOTO 12210
13000 FOR I=1 TO LL:
      PRINT"{CLEAR,DOWN3}LINE #"I" LENGTH (1-36)"
13010 INPUT B%(I)
13020 IF B%(I)>36 OR B%(I)<1 THEN PRINT"NOT ALLOWED":
      GOTO 13010
13030 NEXT:RETURN
14000 REM SORT EXPLAIN
14010 PRINT"{CLEAR}":
      PRINT"{DOWN,SPACE15,RVS}SORT TYPE{RVSOFF,DOWN}"

```

```

14020 FOR I=1 TO 5:PRINT BL$ "I;:PRINT Q$(I):NEXT
14030 PRINT "{DOWN12,SPACE9,RED}HIT ANY KEY TO CONTINUE{BLUE}"
14040 PRINT "{HOME,DOWN2}"
14050 FOR I=1 TO Q:PRINT "":NEXT
14060 PRINT "=====> {RIGHT17}< ====="
14100 ON Q GOSUB 15100,15200,15300,15400,15500
14110 GOSUB 2060:RETURN
14900 GOSUB 2060:RETURN
15100 PRINT "{DOWN2}"MD$Q$(1)" COMPARES ADJACENT"
15110 PRINT BL$"BARS AND INTERCHANGES THEM IF"
15120 PRINT BL$"THEY ARE OUT OF ORDER."
15130 PRINT BL$"THUS BARS ARE 'BUBBLED' UP TO"
15140 PRINT BL$"TO THEIR PROPER POSITION."
15150 RETURN
15200 PRINT "{DOWN}"MD$Q$(2)" HOLDS THE NEXT"
15210 PRINT BL$"BAR IN TEMP WHILE COMPARISONS"
15220 PRINT BL$"ARE MADE FROM THE CURRENT BAR"
15230 PRINT BL$"TO THE FIRST BAR TO FIND THE"
15240 PRINT BL$"PROPER POSITION IN WHICH TO"
15250 PRINT BL$"INSERT TEMP."
15260 RETURN
15300 PRINT "{DOWN}"MD$Q$(3)" GOES THROUGH THE"
15310 PRINT BL$"BARS PLACING THE LARGEST BAR"
15320 PRINT BL$"FOUND AT THE END OF THE LIST"
15330 PRINT BL$"AND STARTS AT THE TOP AGAIN"
15340 PRINT BL$"UNTIL ALL BARS ARE IN ORDER."
15350 RETURN
15400 PRINT MD$Q$(4)" SORTS BY THE SHELL-"
15410 PRINT BL$"METZNER METHOD WHICH SEARCHES"
15420 PRINT BL$"THE BARS BY DECREASING INTER-"
15430 PRINT BL$"VALS, FINE TUNING THE SEARCH"
15440 PRINT BL$"UNTIL ALL BARS ARE IN ORDER."
15450 RETURN
15500 PRINT "{UP}"MD$Q$(5)" INTERCHANGES"
15510 PRINT BL$"NON-ADJACENT BARS BY"
15520 PRINT BL$"PARTITIONING THE BARS INTO"
15530 PRINT BL$"SMALLER GROUPS, SORTING"
15540 PRINT BL$"EACH GROUP INDEPENDENTLY."
15550 RETURN
20000 REM MENU LOADER
20010 PRINT "{CLEAR}LOAD "+CHR$(34)+"0:
      MENU"+CHR$(34)+"8":PRINT "{DOWN4}RUN"

```

```
20020 POKE 631,19: POKE 632,13: POKE 633,13: POKE 198,3: END
28030 A=J+1:SW=SW+1:PRINT"{GREEN}";:GOSUB 1000:PRINT"{BLACK}";
```

Variable Usage for Bar Sort

B%()	Array for bar lengths
SP%()	Array for span lengths in Shell sort
CO()	Table of compares
CO\$	Formatted CO() for printing
SW()	Table of swaps
SW\$	Formatted SW() for printing
Q\$()	Names of sorts
S%()	Stack storage for quicksort
I,J,K,-	
L,WW	Loop counters
V,T,TP,	
M and SP	Temporary storage
F	Flag for bubble sort done
Q	Menu selection
QT	Delay counter
P,QQ	Partitions in quicksort
RS	Random number save
A	Vertical printing coordinate

The Telephone Program

Line 20: SIZE may be set as large as memory allows to accommodate a larger number of names and phone numbers. If the number is too large, a problem may occur in that "garbage collection" will be done to clean up various unused strings to create more string storage. This can cause the computer to appear "dead" for up to five minutes while the process continues. There is no simple cure for this problem.

Line 1000 is the get character subroutine.

Line 2000: 53272 is the memory location controlling character memory. It is POKEd to 'OR 6' at the beginning of the program to allow for lower-case input on the names. It is POKEd back to 'OR 4' at the end of the program to return the C64 to normal upper-case input. If you stop the program with the RUN/STOP key, the C64 will still be in

lower case. Press the COMMODE and the SHIFT keys simultaneously to revert to upper-case input. This is the menu routine and chooses the subroutine based on your input selection.

Line 3000 is the routine to add names to the file. It prints the input prompts and keeps track of the number of entries.

Line 4000: The add, change, and delete routine lists the file one entry at a time and checks to see if you want to change or delete. When the end of the file is reached, it checks to see if you want to add any more names, in which case it jumps to the add routine, or else it returns to the menu.

Line 5000: Pressing return assigns the default value of 'TELEPHONEFILE' in both the LOAD routine and the CREATE routine (line 3000). The number of entries in the file is stored in N\$(0).

Line 6000 starts the SAVE routine.

Line 7000: A bubble sort is performed on either the names, N\$() field, or the phone numbers, P\$() field. AA=0 for name sort and AA=1 for number sort. The compares in lines 7070 and 7080 are executed according to AA. The subroutine at line 7130 does the interchanges and the one at line 7170 prints the display during sort, reversing the one to be switched.

Line 8000 is a short routine to determine whether the OPEN and CLOSE routines should use the number appropriate for disk or tape by assigning D\$ to either 'd' or 't'.

Line 9000 returns to upper case, clears the screen, and ends the program.

There are several points of interest in this program that do not occur in the other sort programs. The POKE in line 2000 sets the keyboard to upper/lower-case mode. This is so you can make a file with capitalized names. The POKE in line 9000 restores the keyboard to normal operation. Line 4020 makes sure that no entry is too large to fit on the screen. The change entry routine in lines 4080-4090 allows for a return key to indicate that the current entry is OK and does not need to be retyped. The entry is changed only if something new is typed in. Lines 7040-7050 set an uppermost boundary to sort against so that there is sure to be an entry larger than any typed in. Lines 7130 and 7140 exchange both the name and the telephone number.

```
10 REM PHONE BOOK
20 SIZE=100: PRINT"{CLEAR}": POKE 53280,14:
POKE 53281,1: GOTO 2000
1000 GET A$: IF A$="" THEN 1000
1010 RETURN
```

```

2000 POKE 53272, (PEEK(53272) AND 240) OR 6
2010 DIM N$(SIZE), P$(SIZE)
2020 B$="{SPACE20}": D$="D"
2030 P$(0)="{↑N}OT {↑U}SED"
2050 PRINT "{CLEAR, DOWN2, RIGHT4, ↑T, ↑E, ↑L, ↑E,
      ↑P, ↑H, ↑O, ↑N, ↑E, SPACE, ↑L, ↑O, ↑G}
2060 PRINT "{DOWN}          {BLUE, ↑M, ↑E, ↑N, ↑U}"
2070 PRINT "{DOWN2}1. {↑C}REATE NEW FILE
2080 PRINT "2. {↑A}DD, {↑C}HANGE, {↑D}ELETE NAMES
2090 PRINT "3. {↑L}OAD EXISTING FILE
2100 PRINT "4. {↑S}AVE EXISTING FILE
2110 PRINT "5. {↑S}ORT FILE
2115 IF D$="D" THEN PRINT "6. {↑T}APE/{RVS, ↑D}ISK{RVSOFF}":
      GOTO 2120
2116 PRINT "6. {RVS, ↑T}APE{RVSOFF}/{↑D}ISK"
2120 PRINT "7. {↑E}XIT TO BASIC"
2125 PRINT "8. {↑E}XIT TO DISK MENU{DOWN3}"
2130 PRINT "{RED, ↑P}LEASE CHOOSE BY NUMBER{BLUE}"
2140 GOSUB 1000: A=0: AA=0
2145 IF VAL(A$)<1 OR VAL(A$)>8 THEN GOTO 2130
2150 ON VAL(A$) GOSUB 3000, 4000, 5000, 6000,
      7000, 8000, 9000, 20005: GOTO 2050
3000 PRINT "{CLEAR, DOWN5, RED, ↑E}NTER NAME OF FILE{BLUE}"
3010 PRINT "{↑R}ETURN IS DEFAULT WHICH IS
      '{↑T, ↑E, ↑L, ↑E, ↑P, ↑H, ↑O, ↑N, ↑E}'
3020 INPUT F$: IF LEN(F$)=0 THEN F$="TELEPHONEFILE"
3030 I=1
3040 PRINT "{RED, CLEAR, ↑E}NTER {↑L}AST {↑N}AME({↑S}PACE)";
3050 PRINT "{↑F}IRST NAME< {↑R}ETURN> ";
3060 PRINT "AND {↑P}HONE {↑N}UMBER ";
3070 PRINT "000-000-0000
3080 PRINT "{↑E}NTER {↑D, ↑O, ↑N, ↑E} WHEN FINISHED{BLUE}": PRINT
3090 INPUT "{↑N}AME"; N$(I)
3095 IF N$(I)="DONE" OR N$(I)="{↑D, ↑O, ↑N, ↑E}" THEN N$(I)="":
      I=I-1: N$(0)=STR$(I) RETURN
3100 INPUT "{↑P}HONE "; P$(I)
3110 IF I>SIZE-2 THEN RETURN
3120 I=I+1: GOTO 3090
4000 J=0
4010 PRINT "{CLEAR}": J=J+1: PRINT "{UP}";
4020 PRINT "{DOWN5, ↑N}AME: "N$(J); LEFT$(B$, 39-LEN(N$(J)))
4030 PRINT "{↑P}HONE: "P$(J)
4040 PRINT "{DOWN9, RED, ↑P}LEASE CHOOSE: {BLUE}

```

```

4041 PRINT "{DOWN,RVS,↑D,RVSOFF}ELETE AN ENTRY"
4042 PRINT "{RVS,↑C,RVSOFF}HANGE AN ENTRY"
4043 PRINT "{RVS}< {↑R}ET> {RVSOFF} TO GO TO NEXT ENTRY"
4044 PRINT "{RVS,↑Q,RVSOFF}UIT TO MENU"
4050 GOSUB 1000
4060 IF LEN(A$)=0 THEN 4100
4070 IF A$="D" OR A$="{↑D}" THEN FOR K=J TO I: N$(K)=N$(K+1):
    P$(K)=P$(K+1): NEXT: I= -1
4080 IF A$="C" OR A$="{↑C}" THEN INPUT "{HOME,DOWN7,↑N}EW
    NAME{SPACE}";N$: IF LEN(N$ <> 0 THEN N$(J)=N$
4090 IF A$="C" OR A$="{↑C}" THEN INPUT "{↑N}EW PHONE ";P$:
    IF LEN(P$)<> 0 THEN P$( )=P$
4100 IF J< I THEN 4010
4105 IF A$="Q" OR A$="{↑Q}" THEN GOTO 4130
4110 PRINT "{RED,DOWN,↑A}DD MORE NAMES? {↑Y}/{↑N,BLUE}"
4120 GOSUB 1000: IF A$="Y" OR A$="{↑Y}" THEN I=I+1: GOTO 3040
4130 PRINT "{CLEAR}": RETURN
5000 PRINT "{CLEAR,DOWN4,RED,↑E}NTER NAME OF EXISTING FILE{BLUE}"
5020 PRINT "< {↑R}ETURN> {SPACE}ENTERS
    '{↑T,↑E,↑L,↑E,↑P,↑H,↑O,↑N,↑E,↑F,↑I,↑L,↑E}'
5030 F$="": INPUT F$: IF LEN(F$)=0 THEN F$="TELEPHONEFILE"
5035 D=1: S=0: IF D$="D" THEN D=8: S=8: F$=F$+",S,R"
5040 OPEN 1,D,S,F$
5050 INPUT# 1,N$(0): I=VAL(N$(0))
5060 INPUT# 1,P$(0)
5070 FORJ=1 TO I
5080 INPUT# 1,N$(J)
5090 INPUT# 1,P$(J)
5100 NEXT: CLOSE 1: PRINT "{CLEAR}": RETURN
6000 PRINT "{CLEAR,DOWN2,↑S}AVING FILE
6010 D=1: S=2: IF D$<> "D" THEN 6070
6020 D=8: S=8
6030 OPEN 1,D,S,F$
6040 PRINT# 1,"S0: "F$
6050 CLOSE 1
6060 F$=F$+",S,W"
6070 OPEN 1,D,S,F$
6080 FOR J=0 TO I
6090 PRINT# 1,N$(J)
6100 PRINT# 1,P$(J)
6110 NEXT: CLOSE 1: PRINT "{CLEAR}": RETURN
7000 IF N$(1)=" THEN PRINT "{CLEAR,DOWN3,↑N}O {↑F}ILE
    TO {↑S}ORT": GOSUB 1000: PR NT "{CLEAR}": RETURN

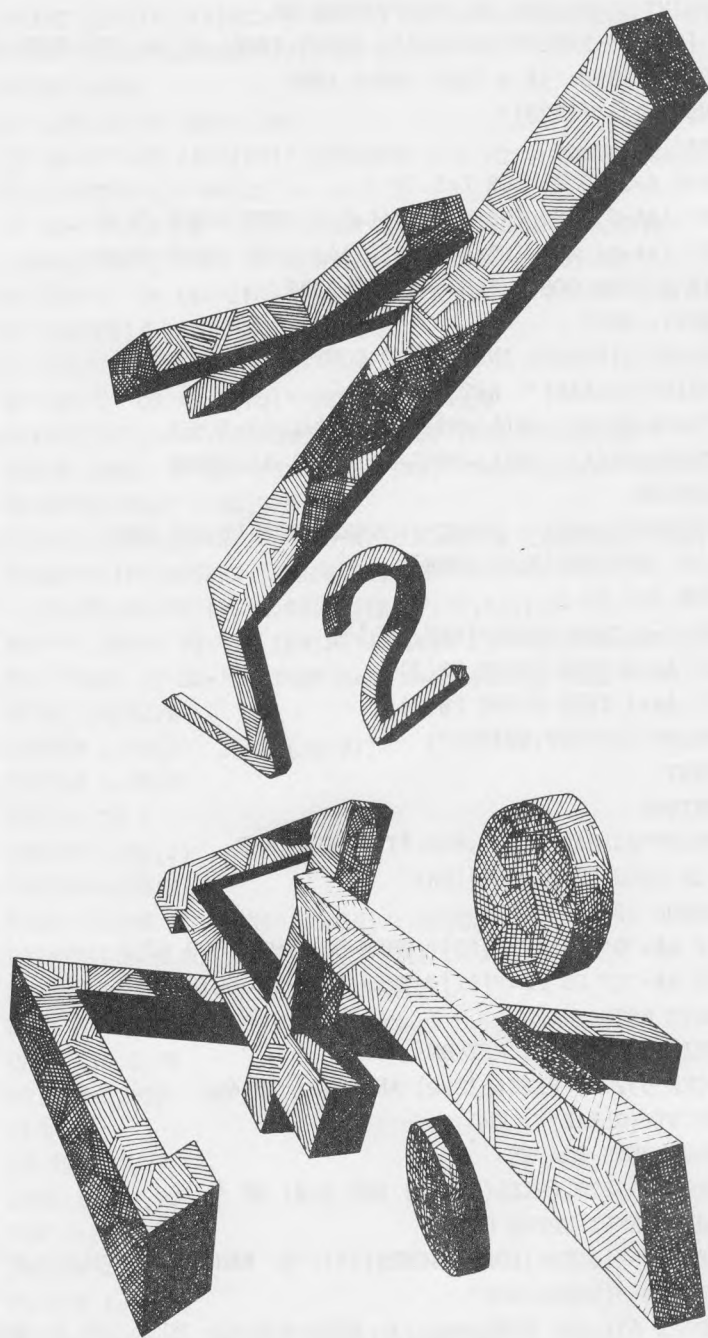
```



```

7010 PRINT "{CLEAR,DW3,↑V}ISUAL {↑S,↑O,↑R,↑T} {↑Y}/{↑N}?":
      GOSUB 1000: IF A$="Y" THEN A=1
7020 PRINT "{DOWN,RVS,↑N,RVSOFF}AMES OR
      {RVS,↑P,RVSOFF}HONE#S?": GOSUB 1000: IF A$ "P" THEN AA=1
7030 GOSUB 7160: IF A THEN GOSUB 1000
7040 N$(I+1)="{↑Z20}"
7050 P$(I+1)=": : : -: : : -: : : : "
7060 FOR K=1 TO I: FOR L=1 TO I
7070 IF (AA=0) AND (N$(L)>N$(L+1)) THEN GOSUB 7130
7080 IF (AA=1) AND (P$(L)>P$(L+1)) THEN GOSUB 7130
7090 IF A THEN GOSUB 7160: GOSUB 1000
7100 NEXT: NEXT
7110 PRINT "{↑P}RESS ANY KEY TO CONT.": GOSUB 1000
7120 PRINT "{CLEAR}": RETURN
7130 TEMP$=N$(L): N$(L)=N$(L+1): N$(L+1)=TEMP$
7140 TEMP$=P$(L): P$(L)=P$(L+1): P$(L+1)=TEMP$
7150 RETURN
7160 PRINT "{CLEAR}": PRINT "{DOWN22,RED}HIT ANY KEY
      TO CONTINUE{BLUE,HOME}"
7170 FOR J=1 TO I
7180 IF J=L THEN PRINT "{RVS,RED}";
7190 IF AA=0 THEN PRINT N$(J)
7200 IF AA=1 THEN PRINT P$(J)
7210 PRINT "{RVSOFF,WHITE}";
7220 NEXT
7230 RETURN
8000 PRINT "{CLEAR,DW3,RVS,↑T,RVSOFF}APE
      OR {RVS,↑D,RVSOFF}ISK?"
8010 GOSUB 1000
8020 IF A$="D" OR A$="{↑D}" THEN D$="D": GOTO 8050
8030 IF A$="T" OR A$="{↑T}" THEN D$="T": GOTO 8050
8040 GOTO 8000
8050 PRINT "{CLEAR}": RETURN
9000 POKE 53272,(PEEK(53272) AND 240) OR 4:
      PRINT "{CLEAR}": END
20000 REM MENU LOADER
20005 POKE 53272,(PEEK(53272) AND 240) OR 4:
      REM RESET UPPER CASE
20010 PRINT "{CLEAR}LOAD "+CHR$(34)+"0: MENU"+CHR$(34)+"",8":
      PRINT "{DOWN4}RUN"
20020 POKE 631,19: POKE 632,13: POKE 633,13: POKE 198,3: END

```



MICRO Calc

A Miniature Spread Sheet

```
* INTEREST CALCULATION DEMONSTRATION*  
* AMOUNT OF LOAN  
A=8000  
* MONTHS  
M=48  
* INTEREST RATE  
I=11.9  
* FORMULAS  
I=I/1200  
D=(1-(1+I)-M)/I  
P=A/D  
P=INT(P*100+.5)/100  
* MONTHLY PAYMENT  
P? 210.28
```

The "electronic spread-sheet" is one of the most popular types of business programs available. The first program of this kind, *VisiCalc*, made quite an impact on the business world and many off-shoots of that original program are now on the market. A spread-sheet program lets you perform mathematical computations on the video display, and applications range from accounting to inventory to printing bar graphs. MICRO Calc is a miniaturized version of a spread-sheet program. With the touch of a key you can perform even very elaborate calculations. Once your sheet is defined for a particular application, you can save it and use it again and again for that application.

Operating Instructions

1. Execute MICRO Calc from the disk menu. The screen will fill with a sample sheet, as shown above.
2. Press the '@' key and after a few seconds the number 210.28 appears on the eighth line. You have just calculated the monthly payment on an \$8000 car, with a 48-month term and 11.9 annual percentage rate.
3. Press the INST/DEL key four times (the diamond moves left, blanking out the 8000 on the first line) and type 6000 to change the amount.
4. Press the '@' key again, and the new payment appears on the eighth line.
5. This time press the return twice to get to the interest line and INST/DEL four times.
6. Type 9.9 to change the interest rate, press '@', and you will soon see how much difference a couple of percentage points makes.

```
A=
M=
I=
I=I/1200
D=(1-(1+I)↑-M)/I
P=A/D
P=INT(P*100+.5)/100
P?
```

Using MICRO Calc

If you don't already have MICRO Calc in memory, Execute MICRO CALC from the *Master Your C64* disk menu. After a brief delay, the screen will fill with the sample screen. Press shifted CLR/HOME and the empty screen will appear with twenty yellow lines and a yellow diamond will appear in the upper left-hand corner. The diamond, called the "cursor," indicates where characters you type will appear on the screen.

A Simple Example

For this first example we want to add two numbers and print the result. Our two numbers are labeled A and B and the sum defined as C. Type the left-hand portion of each of the following lines, pressing RETURN after each.

```
A=5
B=7
C=A+B
C?
```

Now press '@'. The cursor disappears for a second, the answer '12' (the sum of $5 + 7$) is printed after the 'C?' and the cursor reappears after the '5' on the first line. To change the value of A to equal 2, press the INST/DEL key, type '2', and then '@'. The new answer (9) appears after the 'C?'. The formula may be changed. For example, to make it 'C = A - B', use the CRSR down key (unshifted) to move the cursor to the end of the third line. Remove the 'B' by pressing INST/DEL twice, then type the new portion of the formula '- B'. When you press '@', the new answer, '- 5', appears on the fourth line.

Now let's try a more practical example — calculating averages. Use shifted CLR/HOME to clear the screen, and type the following, pressing RETURN after each line.

```
A=165
B=148
C=173
S=A+B+C
S?
V=S/3
V?
```

Press '@' and, after a brief delay, the sum of the three scores, '486', will appear after the S? on the fifth line and the average of the scores, '162', after the V? on the seventh. This procedure could be used to calculate bowling scores. Any three numbers you want to average could be used.

To show how similar "programming" MICRO Calc is to writing BASIC, here is a BASIC program that does the same thing as the averages sheet above.

```
10 INPUT "FIRST SCORE";A
20 INPUT "SECOND SCORE";B
30 INPUT "THIRD SCORE";C
40 S=A+B+C
50 PRINT S
60 V=S/3
70 PRINT V
80 GOTO 10
```

BASIC Program to Calculate Averages

The Rules

MICRO Calc, as you can see, is easy to use. However, it has some rules that must be followed to avoid errors.

1. Each line must begin with a letter, followed either by a '?' or an '='.
2. Nothing should be typed after a '?'. That is where the program fills in its result.
3. After an '=', you must type either a number or a BASIC expression that evaluates to a number. Any BASIC function that yields a numerical result, such as TAN, SQR, RND, LOG, or ABS, may be used. BASIC string functions, such as LEFT\$, MID\$, ASC, or VAL, and integer variables may not be used.

Consult your *C64 User's Guide* to learn about BASIC's built-in functions. Then use MICRO Calc to help you understand how they work.

MICRO Calc allows you to use 26 storage locations called "variables." Each variable is uniquely identified by a letter of the alphabet. In our first example, when we typed 'A = 5', we assigned the value of 5 to the variable A, we typed 'B = 6' to assign the value 6 to B, and typed 'C = A + B' to assign the sum of A and B to C. Until it is changed, 5 is substituted for A whenever it is used in an expression. Likewise, B and C retain their values until they are changed.

As in a BASIC program, MICRO Calc looks at the lines on the screen from the top down. Therefore, a 'C?' on a line before C is defined will give the wrong answer. Also, if you assign a variable a new value, the new value will be used in all subsequent calculations.

Editing

MICRO Calc has limited editing capabilities. You can move the cursor to the end of the line below by pressing either RETURN or CRSR DOWN. From the last line, the cursor will move to the top line. Similarly, CRSR UP will move the cursor to the end of the line above, and from the top line to the bottom line. The only way to change an existing line is to delete from the right end and retype.

Avoiding Errors

To minimize typographical errors, MICRO Calc does not allow characters that could cause trouble. However, if you are not careful it is possible to make a fatal error. The program will stop and an error message, such as ?SYNTAX ERROR or ?DIVISION BY ZERO ERROR, will be printed. Should one of these occur, type RUN, and press RETURN. You will have lost everything that was on the screen, but you should not have to reload the program.

If you use parentheses to assign a variable, for example ' $C = (AB)/(A - B)$ ', be sure there are as many right parens as left parens. Also, remember that, unlike in algebra, multiplication here must be explicitly indicated with an asterisk. Use ' $10*B$ ' and ' $5*(A*B)$ ', not ' $10B$ ' and ' $5(AB)$ '. If you use a variable that has not been assigned in a previous line, its value is assumed to be zero. Therefore, division by an unassigned variable will result in a ?DIVISION BY ZERO ERROR.

Saving the Screen

To save a format for re-use, type in just the assignments ($A =$, $B =$, ...) and the formulas ($C = A + B$), and then save to disk without running the program. For example, clear the MICRO Calc screen using shift CLR/HOME, and type in the second example (bowling average) without any values for A, B, and C. Don't press '@'! The program would crash, but this is the most convenient form for saving the screen. When you are satisfied you have typed the screen correctly, save the screen to disk. Press the F3 key and then press 'S' for SAVE. Type in AVERAGE and press RETURN. Your screen is now stored on your disk under the name AVERAGE.

Loading the Screen

Clear the MICRO Calc screen using shift CLR/HOME, and press the F3 key. This time press 'L' for LOAD, enter AVERAGE and press RETURN. If all goes well, in a few seconds the screen will return with the lines you saved before.

KEY	OPERATION
-----	-----------

```

CRSR    UP AND DOWN CURSOR MOVEMENT
RETURN  DOWN TO NEXT LINE
DEL     DELETE LAST CHARACTER
CLR     CLEAR SCREEN AND ALL VARIABLES
HOME    HOME CURSOR TO TOP LINE
@       PERFORM CALCULATIONS
F1      DISPLAY THIS HELP INFORMATION
F3      LOAD OR SAVE TO DISK OR TAPE
        - RETURN WILL RESTORE DISPLAY
F7      CLEAR VARIABLES

```

WRITE EQUATIONS IN NORMAL BASIC FORM
USE A SEMI-COLON TO SEPARATE MULTIPLE
STATEMENTS ON A LINE (NOT A COLON)
USE SINGLE LETTER VARIABLES
TO PRINT RESULTS USE A QUESTION MARK
AS IN: C?
A COMMENT LINE BEGINS WITH AN *
BASIC, MENU, OR SPACE TO CONTINUE

Pythagorean Theorem

According to the Pythagorean Theorem, "The hypotenuse of a right triangle is equal to the square root of the sum of the squares of the other two sides." You can use MICRO Calc to solve right triangles quickly. Enter the following lines, pressing RETURN after each line.

A=3	assign one side equal to 3
B=4	assign another side equal to 4
C=SQR(A ² +B ²)	the equation
C?	the answer

1. ☐ **Yes**
 2. ☐ **No**
 3. ☐ **Don't know**

Monthly Payments

You can calculate monthly payments of installment loans using the following formula:

$$D = \frac{1 - \frac{1}{(1+i)^m}}{i}$$

i is the interest rate per month and m is the number of months. The principal (the amount you are borrowing) is divided by D to get the monthly payment. Below is the screen you enter to perform these calculations with MICRO Calc.

```
* INTEREST CALCULATION title
* AMOUNT OF LOAN amount of loan
A=8000
* MONTHS number of months of loan
M=48
* INTEREST RATE monthly interest rate
I=11.9
* FORMULAS formulas for calculations
I=I/1200 monthly interest rate
D=(1-(1+I)1-M)/I interest divisor
P=A/D monthly payment rate
P=INT(P*100+.5)/100 round rate to nearest cent
* MONTHLY PAYMENT
P? display monthly payment
```

Before you try it out, SAVE the screen as described above. When it is SAVED the program will return with the cursor at the end of what you typed on the first line. Enter values for A, M, and I, press '@' and the result will appear on the eighth line. Now you can make a change, as you did in the example above; hit '@' and instantly see the change in the monthly payment.

Once you have the hang of MICRO Calc, you can use it to perform a wide variety of repetitive calculations. Be sure to save the more elaborate screens to disk, and you will develop a library of useful screens.

Convert Kilometers to Miles, Feet, and Inches

```
K=? input kilometer value
F=3280.8336*K kilometer to foot conversion factor
F? display kilometer to foot conversion
M=INT(F/5280) convert feet to miles
G=F-M*5280 remaining feet
F=INT(G) whole feet
I=INT((G-F)*12) whole inches
M? display miles
F? display feet
I? display inches
```

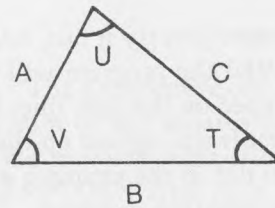
K=8	input kilometer value
F=3280.8336*K	kilometer to foot conversion factor
F? 26246.6688	display kilometer to foot conversion
M=INT(F/5280)	convert feet to miles
G=F-M*5280	remaining feet
F=INT(G)	whole feet
I=INT((G-F)*12)	whole inches
M? 4	display miles
F? 5126	display feet
I? 8	display inches

Calculate Triangle

```

T=
U=
C=
V=180-T-U
PI=3.14159265
V=PI*V/180
U=PI*U/180
B=SIN(U)*C/SIN(V)
B?

```



Calculate a second side of triangle, given two angles and included side.

T=30	input angle T in degrees
U=60	input angle U in degrees
C=2	input side C
V=180-T-U	calculate second side
PI=3.14159265	
V=PI*V/180	convert degrees to radians
U=PI*U/180	
B=SIN(U)*C/SIN(V)	apply law of sines
B? 1.73205081	display side B

Calculate third side of triangle,
given two sides and included angle.

```
A=◆
B=
V=
PI=3.14159265
V=PI*V/180
D=A^2+B^2
E=2*A*B*COS(V)
C=SQR(D-E)
C?
```

```
A=6◆
B=7
V=40
PI=3.14159265
V=PI*V/180
D=A^2+B^2
E=2*A*B*COS(V)
C=SQR(D-E)
C? 4.54447652
```

input side A
input side B
input angle V

convert degrees to radians
apply law of cosines

display side C

Random Number Demonstration

```
T=◆
T=RND(-T)
T=RND(1)
T?
T=RND(1)
T?
T=RND(1)
T?
T=RND(1)
T?
```

input seed value
set new seed value
find first random number
display first random number
find second random number
display second random number
find third random number
display third random number
find fourth random number
display fourth random number

T=7.204	input constant seed value
T=RND(-T)	set constant seed value
T=RND(1)	find and display random numbers
T? .203287635	same sequence every time
T=RND(1)	
T? .241506909	
T=RND(1)	
T? .734391969	
T=RND(1)	
T? .172166796	
T=TI	input value of internal timer
T=RND(-T)	set timer value as new seed
T=RND(1)	find and display random numbers
T? .864552711	different sequence each time
T=RND(1)	
T? .699672199	
T=RND(1)	
T? .147837161	
T=RND(1)	
T? .876745393	

How the Program Works

Initialization (10-60): The screen is cleared by printing the CLR character. Then the screen color is set to white by POKEing 53281 with 1 and the border color is set to blue by POKEing 53280 with 6. Subroutine 8000 READs the machine-language routine from the DATA statements 9828-9868 and POKEs them into memory. Next, seven frequently used constants are defined. Q\$ contains CHR\$(34), the quote. The RETURN and DEL keys do not cause characters to appear on the screen or in a listing, but their numeric codes, 13 and 20, can be used with the CHR\$ function. RB\$ is used to wipe out the diamond cursor by printing a RVS ON character, an underline, and the RVS OFF character. BL\$ is a string of 38 underlines used to pad out the remainder of screen lines. DI\$ is the diamond cursor. The cursor left character sets the next print position on top of the diamond. CC\$ is a solid dot, set character color black, and cursor left used in comment editing. NL is the number of lines and is set to 20. Line 50 also DIMensions the arrays S\$ and S which contain the strings and numbers for each line. Finally, line 60 sets the line pointer LL to 1, and calls a routine at line 8400 that loads the demonstration screen from DATA statements 9900-9910 into the arrays S\$ and S. The program branches to line 7000 where the operating instructions are printed, and then waits for an input.

Program mainline (100-280): Line 100 sets the line pointer LL to 1, clears the screen content arrays using subroutine at line 8500, clears the variables using subroutine at line 9000, and prints the blank screen. If we hadn't loaded the demonstration screen (line 60), this would be the first instruction in the mainline. The clear key [CLR] causes the program to return to this line. Line 105 resets the pointer to 1. Line 110 is encountered every time the cursor is moved to a new line. The contents of that screen line (S\$(LL)) are moved into the temporary string S\$. If the last character is not a '?', then the screen line is printed (115), followed by the cursor. If that character is a '?', then the whole line is wiped out with a reverse on {RVS} and the 38 underline characters of BL\$. A carriage return (CR\$) moves the cursor to the beginning of the next line and the cursor up {UP} character moves the cursor back up to the beginning of the current line, ready for line 115. This sequence causes any value left on the line by a previous calculation to be wiped out, leaving only the value request statement.

Line 120 calls the editor subroutine (2000). Unless the cursor is moved to a new line or a load, save, or calculate command is given, the program flow returns here after each character is accepted. See the discussion of the editor subroutine 2000 for details of its operation. T\$ is the character just accepted by the editor. All accepted characters, except the command characters, are dealt with in line 200. (These characters fail the IF tests in lines 130-180.) They are simply tacked on the end of the temporary string S\$. The " operation, when used with strings, causes the characters to be put together end to end in a process called 'concatenation.' If the length of S\$ is less than the maximum 37, then the program loops back to 120 to get another character. If the line has reached its maximum length, then the program falls through to the next line sequence, described below.

The command characters are dealt with one by one in lines 125-180. '@', clear {CLEAR}, carriage return, cursor down {DOWN}, cursor up {UP}, and delete are dealt with by branching elsewhere in the program. Pressing F1 causes a branch to line 7000 which displays the instructions. The '@' character causes a branch to line 300 which starts the calculation process.

Line 135 tests for F7 which clears the variables.

The clear screen command is dealt with in line 140 simply by branching to line 100, where the screen arrays are emptied before printing the blank screen.

Line 150 causes a branch on either a carriage return or cursor down {DOWN} to the 'next line' sequence at 210. Before moving to the next

screen line, line 210 moves the temporary string S\$ into the proper array element (indicated by the line pointer LL). Then LL is incremented. If $LL = NL + 1$ (NL is a constant set to 20 in line 50), then LL is reset to 1 and the cursor adjusted for the first line; otherwise, the cursor is advanced to the next line. Note that, in either case, before the cursor position is moved the old diamond is wiped out by printing RB\$. Also, in both cases, the program branches to 110, the 'new line' point.

Line 160 branches on a cursor up {UP} to 270 where the 'previous line' sequence is executed. The logic here is similar to the 'next line' sequence starting at 210. The line pointer is decremented and the cursor moved up to the last line. The exception is when LL reaches zero. Then the last line is reached with the cursor home and 19 cursor downs {HOME, DOWN19} sequence. Finally, a branch is made back to line 110, the 'new line' point.

Line 170 branches on a delete character to 240. If the temporary string S\$ is already empty then the character is ignored; otherwise, S\$ is redefined to include all but the right-most character. This is accomplished with the LEFT\$ function. Within the parentheses, two items are separated by commas. The first is the name of the string being operated upon, and the second is an expression that evaluates to a number. The string resulting from the operation (in this case, the redefined S\$) consists of that number of characters starting at the left end. The number is calculated by taking the number of characters in S\$ (using the LEN function) and subtracting 1. Next the result of this operation is duplicated on the screen by wiping out the cursor with RB\$, moving left twice, and reprinting the cursor on top of the deleted character.

Editor subroutine (2000-2070): A call is made to subroutine 4500 to get a character, which is returned in T\$. Next T\$ is tested against the command characters. The command characters are dealt with individually upon return to the program mainline. Most of the other acceptable characters fall into one of the three ranges tested in 2030-2050. Each character has a unique numeric code, which makes it testable with less than and greater than statements. For instance, the code for a comma is 34, while the code for a semicolon is 59. All the characters with codes between, but not including, these two numbers are acceptable. Complete tables of these codes are available in the *Commodore 64 Users Guide*. If T\$ fails all of the IF tests, the program drops through to line 2070, where a branch is made to line 2000 to get another character.

Line processing (3000-3050): Line 3010 checks the screen line by line. If the right character on a line is a '?', then the line is assumed to

be a value request and subroutine 6500 is called. Subroutine 6500 places the proper user variable in the appropriate element of the screen's value array.

Line 3020 tests if the first character is an asterick which denotes a comment line which is to be skipped. Line 3030 tests for lines of two or fewer characters and skips them; otherwise, the line is assumed to be a valid assignment statement. A\$ is set to the contents of the line and the assignment subroutine 4000 is called. All lines on the screen are processed. The top line message is cleared and a RETURN is made to the main line code.

Assignment (4000-4100): Line 4020 uses the MID\$ function to extract each character, in sequence, from the screen line. The ASC function converts the character to its numeric code. Then the numeric code is POKEd into a memory area called the INPUT buffer. After all characters in the line have been transferred, a zero is POKEd to indicate the end. SYS828 calls the machine-language routine that tokenizes the statement, evaluates the expression, and assigns the value to the user variable. 828 is the address of the start of the machine-language program.

GET character (4500-4510): The BASIC GET function accepts a single character from the keyboard. This character is placed in the variable T\$. If no key has been pressed, then BASIC makes T\$ a null, or empty string. This is tested with the IF statement, so if no key is pressed, line 4500 will repeat over and over. When a key is pressed, control is passed to the RETURN statement in 4510.

LOAD and SAVE routines (5000-5120): This routine is reached with the F3 key. For storing and retrieving data and programs a file is set up using an OPEN statement, which usually consists of four items separated by commas. The first item is the file's number. After the file is set up, only its number is required in subsequent operations. The second item is the device number. The disk is device number 8 and the cassette is device number 1. The third item in the OPEN statement is the secondary address. For the cassette unit, 0 means we are reading from the tape and 1 means we are writing to it. Finally, a string may be included to identify the file.

Lines 5000-5040 determine whether the operation is a LOAD or SAVE and set SA, which will be the secondary address, accordingly. The name of the file NA\$ is accepted in line 5090. Line 5100 tests if T\$ is 'D'. If it is, then DV is set to 8 for disk, SA is increased by 8, and NA\$ is set to '0:', the prefix for disk 0. Next channel 15 is opened up. Line 5120 opens logical file number 1 for the cassette unit or the disk, using the name and secondary address established above. 'IF SA' is the same

as 'IF SA=0', so 5090 is the SAVE routine and 5110 is the LOAD routine. After returning from the SAVE or LOAD operation, the file is CLOSED, the machine code re-established (GOSUB 8000), and the screen reprinted.

The SAVE routine (5150-5160) goes through the screen line array line by line. Each line is put into the temporary string S\$. If S\$ is null, then it is set to a dummy character. When the loop is through, A\$ will have all the characters of the screen line, with a carriage return character after each line. This is accomplished with the " or concatenation operation. Finally the PRINT# command is used to write the single string A\$ to the file. This records it on the tape or on disk.

The LOAD routine (5170-5940) uses the INPUT# command to get one line at a time from tape or disk. The carriage return characters included in the SAVE operation indicate the end of each string to the INPUT# function. The dummy characters are discarded and replaced with null strings.

Lines 6500-6810 retrieve the value of a variable. Subroutine 3000 comes here when a '?' is detected at the end of a line. First, the letter identifying the variable is determined. This is accomplished in line 6500 by using the LEFT\$ function to get the first character in the line. The ASC function converts the character to its numeric code, and subtracting 64 converts the codes, which start at 65 for the letter A, into numbers 1 to 26. Numbers 1 to 13 are handled in 6510, while numbers 14 to 26 are converted to 1 to 13 and handled in line 6530. When BB is 1, subroutine 6560 is called; when it is 2, subroutine 6570 is called. When BB starts out as 14, the call is to 6690. Each one of these mini-subroutines sets XX to the contents of a different user variable. Before returning, the appropriate screen line value array element is set equal to XX.

Display operating instructions (7000-7200): Instructions for MICRO Calc operation appear in lines 7000-7190. Lines 7195-7200 handle the selection by the user.

Print screen with values (8200-8220): The tricky part of this routine is filling out the rest of the line with reverse blanks after a value is printed. SS is used to temporarily hold the contents of the value array element for the current line. If there is no value for the line, X\$ will remain null, and the line will be reprinted as it is.

A value is first converted to a string using the STR\$ function. The length of that string is subtracted from 16. This determines how many spaces to extract from BL\$ with the LEFT\$ function. The resulting X\$ includes the value (converted to a string), the [RVS] character, and the

right number of spaces to fill out the line.

Set machine-language routine (8000): The RESTORE statement sets the DATA pointer to the beginning of the DATA elements. This is not required the first time the DATA is read, but later, after a LOAD or SAVE operation, the DATA pointer will be in the wrong place. The READ statement gets the DATA element indicated by the pointer, places it in AA, and advances the pointer to the next DATA element. As each number is READ it is POKed in sequence into the addresses beginning at 828. These numbers make up the short machine-language program, explained earlier.

Line 8400 READs in the lines of the demonstration screen from DATA statements 9900 and 9910. Line 8500 clears the screen arrays, and lines 8510-8530 print the screen, leaving the cursor on the first line.

The subroutine at line 9000 sets all the user variables to zero. Setting A to zero and then using A to set all the other variables increases the speed slightly.

Expanding MICRO Calc

MICRO Calc was kept simple intentionally so that it would be easy to use. The next section describes how you can make changes in a few lines to improve the power of MICRO Calc.

Flashing cursor

With the MICRO Calc program in memory, type the following program lines:

```
4500 POKE 204,0:POKE 207,0
4510 GET T$:IF T$="" THEN 4510
4520 POKE 204,1:RETURN
```

Normally, when the GET function is used, the cursor is turned off. POKEing a 0 into this location turns the flashing cursor on, and POKEing a 1 turns it off. Incidentally, you can change the cursor character from the diamond to anything you want. Just change the diamond near the end of line 20 to the desired character. If you want to save this, or any other enhanced version, be sure you don't record over the original version of the program.

Suggestions for other changes

With just a little work, MICRO Calc can be made to handle strings as well as numbers. First, all string names within the program must be changed to two characters. Second, a separate series of subroutines

with an appropriate calling routine, similar to 6500-6810, must be added to extract the proper values for printing. Third, the quote and dollar characters must be accepted by the editor. Finally, the print routine at 7000 must be made to recognize and print strings. Probably other minor problems would have to be solved as well. The editor can be improved considerably by allowing insert, delete, and cursor movements within a line. You can also define constants within the program, such as $25 \text{ PI} = 3.14159265$. Then you can use PI in a variable definition on any screen. Consult the variable usage table below to be sure you don't re-use a name. You might also find it convenient to define some functions, such as for converting degrees to radians, within the program.

```

10 REM MICRO CALC
20 PRINT "{CLEAR}":POKE 53281,1:POKE 53280,6:GOSUB 8000
30 Q$=CHR$(34):CR$=CHR$(13):DL$=CHR$(20):
   RB$="{RVS,=@,RVSOFF}"
40 BL$="{=@38}":DI$="{↑Z,RVSOFF,LEFT}":CC$="{↑Q,BLACK,LEFT}"
50 NL=20:DIM S$(NL),S(NL)
60 LL=1:GOSUB 8400:PRINT "{HOME,DOWN}";:GOTO 7000
100 LL=1:GOSUB 8500:PRINT "{HOME,DOWN}";:GOSUB 9000
105 LL=1
110 S$=S$(LL):IF RIGHT$(S$,1)="?" THEN
   PRINT "{RVS}"BL$CR$"{UP}";
115 PRINT "{BLUE}"$SDI$;
120 GOSUB 2000
125 IF T$="{F1}" THEN 7000
130 IF T$="@" THEN 300
135 IF T$="{F7}" THEN GOSUB 9000:LL=1:PRINT "{HOME,DOWN}";:
   GOTO 110
140 IF T$="{CLEAR}" THEN 100
150 IF T$=CR$ OR T$="{DOWN}" THEN 210
155 IF T$="{HOME}" THEN LL=NL+1:GOTO 220
160 IF T$="{UP}" THEN 270
170 IF T$=DL$ THEN 240
180 IF T$="{F3}" THEN S$(LL)=S$:GOSUB 5000:GOSUB 9000:LL=1:
   GOTO 110
200 IF LEN(S$)<37 THEN S$=S$+T$:PRINT T$DI$;:GOTO 120
210 S$(LL)=S$:LL=LL+1
220 IF LL=NL+1 THEN LL=1:PRINT RB$;:PRINT "{HOME,DOWN}";:
   GOTO 110

```



```

230 PRINT RB$CR$;:GOTO 110
240 IF S$="" THEN 120
250 PRINT RB$"{LEFT2}"DI$;
260 S$=LEFT$(S$,LEN(S$)-1):GOTO 120
270 S$(LL)=S$:LL=LL-1
275 IF LL=0 THEN LL=NL:PRINT RB$"{HOME,RIGHT10,DOWN20}";:
    GOTO 110
280 PRINT RB$CR$"{UP2}";:GOTO 110
300 PRINT RB$:S$(LL)=S$:GOSUB 3000:GOSUB 8200:
    PRINT"{HOME,DOWN}";:LL=1:GOTO 110
2000 GOSUB 4500
2010 IF T$="{HOME}" OR T$=CR$ OR T$="{DOWN}" OR T$="{UP}"
    OR T$=DL$ OR T$="{CLEAR}" THEN RETURN
2020 IF T$="{F1}" OR T$="{F3}" OR T$="{F7}" OR T$="@ "
    OR T$=" " THEN RETURN
2030 IF T$>" ," AND T$<": THEN RETURN
2040 IF T$>": " AND T$< "[" THEN RETURN
2050 IF T$>" '" AND T$< ", " THEN RETURN
2060 IF T$="{UP ARROW}" THEN RETURN
2070 GOTO 2000
3000 PRINT"{HOME,RED}CALCULATING{BLUE}"
3010 FOR JJ=1 TO NL:IF RIGHT$(S$(JJ),1)="?" THEN GOSUB 6500:
    GOTO 3050
3020 IF LEFT$(S$(JJ),1)="*" THEN 3050
3030 IF LEN(S$(JJ))<3 THEN 3050
3040 A$=S$(JJ):GOSUB 4000
3050 NEXT:PRINT"{HOME}" "":RETURN
4000 II=0:KK=II
4010 II=II+1:KK=KK+1:IF II>LEN(A$) THEN GOSUB 4100:RETURN
4020 XX=ASC(MID$(A$,II,1)):IF XX=59 THEN GOSUB 4100:GOTO 4010
4030 IF XX=33 THEN GOSUB 4100:RETURN
4040 POKE 511+KK,XX:GOTO 4010
4100 POKE 511+KK,0:KK=0:SYS 828:RETURN
4500 GET T$:IF T$="" THEN 4500
4510 RETURN
5000 PRINT"{CLEAR,BLACK}L{BLUE}OAD OR {BLACK}S{BLUE}AVE":
    LS=0:GOSUB 5030
5010 IF LS=1 THEN GOSUB 8510:GOSUB 9030:GOTO 105
5020 GOSUB 9000:GOTO 105
5030 GOSUB 4500:IF T$=CR$ THEN LS=1:RETURN
5040 IF T$="L" THEN SA=0:FD$=",S,R":GOTO5070
5050 IF T$="S" THEN SA=1:FD$=",S,W":GOTO5070
5060 GOTO 5030

```

```

5070 PRINT "{DOWN,BLACK}D{BLUE}ISK OR {BLACK}T{BLUE}APE":
      GOSUB 4500:IF T$=CR$ THEN LS=1:RETURN
5080 IF NOT((T$="D") OR (T$="T")) THEN 5070
5090 INPUT "{DOWN2}NAME";NA$:IF LEN(NA$)=0 THEN LS=1:RETURN
5100 IF T$="D" THEN SA=SA+8:DV=8:NA$="@0:" +NA$+FD$:
      OPEN 15,8,15:GOTO 5120
5110 DV=1:NA$=""
5120 OPEN 1,DV,SA,NA$:IF SA AND 1 THEN GOSUB 5150:GOTO 5140
5130 GOSUB 5170
5140 CLOSE 1:CLOSE 15:GOSUB 8000:GOSUB 8510:
      PRINT "{HOME,DOWN}";:RETURN
5150 A$="":FOR II=1 TO NL:S$=S$(II):IF S$="" THEN S$="{=B}"
5160 A$=A$+S$+CR$:NEXT:PRINT# 1,A$:DE=0:GOTO 5900
5170 FOR II=1 TO NL:INPUT# 1,A$:DE=0:GOSUB 5900:
      IF DE THEN I=NL:NEXT:RETURN
5180 IF A$="{=B}" THEN A$=""
5190 S$(II)=A$:NEXT:RETURN
5900 IF DV=1 THEN RETURN
5910 INPUT# 15,D1$,D2$,D3$,D4$:IF VAL(D1$)=0 THEN RETURN
5920 PRINT "{CLEAR}"D1$" "D2$" "D3$" "D4$
5930 FOR JJ=1 TO 2000:NEXT
5940 DE=-1:RETURN
6500 BB=ASC(LEFT$(S$(JJ),1))-64:IF BB> 13 THEN BB=BB-13:
      GOTO 6530
6510 ON BB GOSUB 6560,6570,6580,6590,6600,6610,6620,
      6630,6640,6650,6660,6670,6680
6520 GOTO 6540
6530 ON BB GOSUB 6690,6700,6710,6720,6730,6740,6750,
      6760,6770,6780,6790,6800,6810
6540 S(JJ)=XX
6550 RETURN
6560 XX=A:RETURN
6570 XX=B:RETURN
6580 XX=C:RETURN
6590 XX=D:RETURN
6600 XX=E:RETURN
6610 XX=F:RETURN
6620 XX=G:RETURN
6630 XX=H:RETURN
6640 XX=I:RETURN
6650 XX=J:RETURN
6660 XX=K:RETURN

```

```

6670 XX=L:RETURN
6680 XX=M:RETURN
6690 XX=N:RETURN
6700 XX=O:RETURN
6710 XX=P:RETURN
6720 XX=Q:RETURN
6730 XX=R:RETURN
6740 XX=S:RETURN
6750 XX=T:RETURN
6760 XX=U:RETURN
6770 XX=V:RETURN
6780 XX=W:RETURN
6790 XX=X:RETURN
6800 XX=Y:RETURN
6810 XX=Z:RETURN
7000 PRINT "{CLEAR,BLACK} {SPACE}MICRO CALC
      OPERATING INSTRUCTIONS"
7010 PRINT "{DOWN}KEY  OPERATION"
7020 PRINT "{DOWN,BLACK}CRSR{BLUE}  UP AND DOWN
      CURSOR MOVEMENT"
7030 PRINT "{BLACK}RETURN{BLUE} DOWN TO NEXT LINE"
7040 PRINT "{BLACK}DEL{BLUE}  DELETE LAST CHARACTER
7050 PRINT "{BLACK}CLR{BLUE}  CLEAR SCREEN AND ALL VARIABLES"
7060 PRINT "{BLACK}HOME{BLUE}  HOME CURSOR TO TOP LINE"
7070 PRINT "{BLACK}@{BLUE}    PERFORM CALCULATIONS"
7080 PRINT "{BLACK}F1{BLUE}    DISPLAY THIS HELP INFORMATION"
7090 PRINT "{BLACK}F3{BLUE}    LOAD OR SAVE TO DISK OR TAPE"
7100 PRINT "          RETURN WILL RESTORE DISPLAY"
7110 PRINT "{BLACK}F7{BLUE}    CLEAR VARIABLES"
7120 PRINT "{DOWN,BLACK}WRITE EQUATIONS IN NORMAL BASIC FORM"
7130 PRINT "USE A SEMI-COLON TO SEPARATE MULTIPLE"
7140 PRINT "  STATEMENTS ON A LINE (NOT A COLON)"
7150 PRINT "USE SINGLE LETTER VARIABLES"
7160 PRINT "TO PRINT RESULTS USE A QUESTION MARK"
7170 PRINT "  AS IN: {BLUE}C?{BLACK}"
7180 PRINT "A COMMENT LINE BEGINS WITH AN *"
7190 PRINT "{DOWN,BLACK}B{BLUE}ASIC, {BLACK}M{BLUE}ENU,
      OR {BLACK}SPACE{BLUE} TO CONTINUE"
7195 GOSUB 4500:IF T$="M" THEN 20000
7196 IF T$="B" THEN PRINT "{CLEAR,BLACK}":POKE 53281,1: END
7200 GOSUB 8510:GOSUB 9030:GOTO 105
8000 RESTORE:FOR II=0 TO 42:READ AA:POKE 828+II,AA:
      NEXT:RETURN

```

```

8200 PRINT "{HOME,DOWN}";:FOR II=1 TO NL:S$=S$(II):SS=S(II)
8210 X$="":IF RIGHT$(S$,1)="?" THEN X$=STR$(SS)
      +"{RVS}"+"LEFT$(BL$,34-LEN(STR$(SS)))
8220 PRINT S$X$:NEXT:RETURN
8400 FOR II=1 TO NL:READ S$(II):S(II)=0:NEXT:RETURN
8500 FOR II=1 TO NL:S$(II)="" :S(II)=0:NEXT
8510 PRINT "{CLEAR,DOWN}";:FOR II=1 TO NL:S$=S$(II)
8520 PRINT "{BLUE}"S$"{RVS}"LEFT$(BL$,38-LEN(S$))
8530 NEXT:PRINT "{DOWN,BLACK}"MID$(NA$,4)"{LEFT4}      ":RETURN
9000 PRINT "{HOME,RED}CLEARING VARIABLES"
9010 A=0:B=A:C=A:D=A:E=A:F=A:G=A:H=A:I=A:J=A:K=A:L=A:M=A
9020 N=A:O=A:P=A:Q=A:R=A:S=A:T=A:U=A:V=A:W=A:X=A:Y=A:Z=A
9030 FOR JJ=1 TO 1000:NEXT
9040 PRINT "{HOME}                                {BLUE}":RETURN
9828 DATA 165,122,141,112,3,165,123,141,113,3,
      169,0,133,122,169,2,133,123,32,121
9848 DATA 165,169,0,133,122,169,2,133,123,32,
      165,169,173,112,3,133,122,173,113,3
9868 DATA 133,123,96
9900 DATA * INTEREST CALCULATION DEMONSTRATION,
      *, * AMOUNT OF LOAN
9910 DATA A=8000,*, MONTHS,M=48,*, INTEREST RATE,I=11.9
9920 DATA * FORMULAS,I=I/1200,D=(1-(1+I){UP ARROW}-M)/I,
      P=A/D,P=INT(P*100+.5)/100
9930 DATA * MONTHLY PAYMENT,P?,,,,,,
20000 REM MENU LOADER
20010 PRINT "{CLEAR}LOAD "+CHR$(34)+"0:MENU"+CHR$(34)+"8":
      PRINT "{DOWN4}RUN"
20020 POKE 631,19: POKE 632,13: POKE 633,13: POKE 198,3: END

```

Machine-Language Program for MICRO Calc

The machine-language program listed here is included in the BASIC program in DATA statements 9828-9868. It is POKED byte by byte into the C64's memory. The routine is called for each assignment line on the screen during the calculate process.

What the routine does is very simple, yet an equivalent BASIC program would be considerably longer and slower. Most of the program is involved with manipulating the character pointer (\$7A,\$7B). When the routine begins, the pointer marks the place in the BASIC program.

Therefore, the pointer contents must be saved in a temporary location (\$0370,\$0371) so we can direct BASIC back to the right place. Next the pointer is changed to point to the input buffer, where the BASIC program has stored the current line. The BASIC ROM routine converts the line into tokenized form. The pointer must again be directed to the beginning of the input buffer. Then the LET routine evaluates the line and assigns the calculated value to the specified variable. Before returning to BASIC, the original value for the pointer is retrieved from the temporary locations. With the RTS instruction, BASIC now can resume where it left off.

Assembly Program at 828 in MicroCalc

033C A5 7A	ENTER	LDA	TXTPTR	SAVE CURRENT BYTE PNTR
033E 8D 70 03		STA	TEMPLO	OF MICROCALC BASIC PROGRAM
0341 A5 7B		LDA	TXTPTR+1	TWO BYTE POINTER TO SAVE
0343 8D 71 03		STA	TEMPHI	
0346 A9 00		LDA	#BUFLO	REPLACE WITH POINTER TO THE
0348 85 7A		STA	TXTPTR	SYSTEM INPUT BUFFER
034A A9 02		LDA	#BUFHI	TWO BYTE POINTER TO SET
034C 85 7B		STA	TXTPTR+1	
034E 20 79 A5		JSR	TOKEN	TOKENIZE THE BASIC LINE
0351 A9 00		LDA	#BUFLO	RESET THE POINTER TO THE
0353 85 7A		STA	TXTPTR	START OF THE INPUT BUFFER
0355 A9 02		LDA	#BUFHI	
0357 85 7B		STA	TXTPTR+1	
0359 20 A5 A9		JSR	LET	EXECUTE THE ASSIGNMENT
035C AD 70 03		LDA	TEMPLO	RESTORE THE REAL
035F 85 7A		STA	TXTPTR	PROGRAM POINTER
0361 AD 71 03		LDA	TEMPHI	
0364 85 7B		STA	TXTPTR+1	
0366 60		RTS		AND RETURN

MICRO Calc Variable Usage

Constants:

NL number of screen lines
CR\$ carriage return
DL\$ delete character
RB\$ reverse blank
BL\$ spaces
DI\$ diamond cursor

Variables

A...Z reserved for user numbers
A\$...Z\$ reserved for user strings
II loop counter
JJ loop counter
LL screen line pointer
XX temporary, used in value passing
BB letter of variable converted to index 1...26
SS current element of value array
S\$ working copy of screen line
T\$ last character from GET (4500)
A\$ working string for tape and disk
X\$ used in formatting screen lines
S\$() array of screen line contents
S() array of screen line values

Disk Menu

Automatic Disk File Service

The Disk Menu program that you have been using to load and/or execute programs from this book can be used on any Commodore 64 diskette. To move MENU to your own diskette, simply LOAD "MENU",8 from the *Mastering Your Commodore 64* diskette and SAVE "MENU",8 to your own diskette. That's all there is to it. There is nothing specific to the particular diskette. The program reads the directory of the diskette and tests to see if the file type for an entry is PRG. If not, then the entry is ignored since only PRG files can be directly loaded and run. Other file types, such as SEQ must normally be read by a particular program, and may not be directly accessed. Examples are the files SONG.SEQ used by the PLAYER program and SCREENDEMO.SEQ used by the SCREEN EDITOR program. These are not listed by MENU, but may be listed using the standard C64 directory command (LOAD "\$",8 followed by LIST) which shows ALL files regardless of type.

If you want to return to the Menu after running some program of your own, simply add the lines shown below to your program, plus some method of getting to these lines — for example a particular keyboard input. Adding a test such as:

```
IF T$="x" THEN 200000
```

where T\$ contains a character input from the keyboard and "x" is the character to cause your program to call the menu, will do the trick.

```
200000 MENU LOADER
200010 PRINT "{CLEAR}LOAD "+CHR$(34)+"0.MENU"+CHR$(34)+"",8":
      PRINT "{DOWN4}RUN"
200020 POKE 631,19: POKE 632,13: POKE 633,13: POKE 198,3: END
```

Since the line numbers for the Menu Loader are not used internally, they may be changed to any convenient addresses.

```

10 REM DISK MENU PROGRAM
20 POKE 53280,6:POKE 53281,1
30 PRINT"{CLEAR,BLACK}";
100 DIM F$(224)
110 REM -- POKE ML ROUTINE --
120 FOR X=880 TO 950: READ Z: POKE X,Z: CS=CS+Z: NEXT
130 IF CS<>10432 THEN PRINT "{CLEAR,RVS}
    ERROR - CHECK DATA STATEMENTS...{RVSOFF}": END
150 REM -- READ DIRECTORY --
160 PRINT "{CLEAR}"TAB(8)"{RVS}
    DISK PROGRAM MENU {RVSOFF,DOWN}"
180 D=0:D$=""
190 OPEN 15,8,15
200 PRINT# 15,"I"+D$: GOSUB 290
210 OPEN 1,8,0,"$"+D$: GOSUB 290
220 H$=" "+"": SYS 880:
    REM 17 SPACES - READ HEADING
230 IF ST THEN CLOSE1: CLOSE15: GOTO 340
240 F$=" "+"": SYS 880:
    REM 17 SPACES - READ FILENAME
250 IF RIGHT$(F$,1)="P" THEN C=C+1: F$(C)=LEFT$(F$,16)
260 GOTO 230
270 :
280 REM -- DISK ERROR ROUTINE --
290 INPUT# 15,EN,EM$,ET,ES: IF EN=0 THEN RETURN
300 PRINT EN;EM$;ET;ES
310 CLOSE 1: CLOSE 15: END
320 :
330 REM -- DISPLAY MENU --
340 IF F=0 OR F=C THEN S=1: F=C: IF C>40 THEN F=40:
    GOTO 360
350 IF C>F THEN S=S+20: F=C: IF C>S+20 THEN F=S+19
360 L=0
370 PRINT "{RVS}DRIVE #:{RVSOFF}";D;"{RVS}DISK NAME:
    {RVSOFF} ";H$"{DOWN}"
380 FOR X=S TO S+19
390 L=L+1
395 IF LEN(F$(X))=0 THEN 420
400 IF ASC(F$(X))=32 THEN 420
410 PRINT "{RVS}"RIGHT$(STR$(L),2)"{RVSOFF}";CHR$(95);" ";
    F$(X)
420 NEXT
430 PRINT

```

```

440 PRINT "ENTER: # TO LOAD, - # TO EXECUTE"
450 PRINT " {UP ARROW} NEXT PAGE, {POUND}
      NEW DISK, @ EXIT"
460 PRINT "{RVS} SELECT PROGRAM {RVSOFF}";:INPUTG$
470 IF G$="{POUND}" THEN 160
490 IF G$="{UP ARROW}" THEN 340
500 IF G$="@" THEN PRINT "{CLEAR}":END
510 EX=2 : REM ASSUME LOAD
520 G=VAL(G$):IF G<0 THEN EX=3:G=ABS(G)
530 IF (S+G-1)>F THEN 450
550 REM -- CHAIN --
560 F$=F$(S+G-1)
570 IF RIGHT$(F$,1)=" " THEN F$=LEFT$(F$,LEN(F$)-1):
      GOTO 570
580 PRINT "{CLEAR}LOAD"+CHR$(34)+D$+": "+F$+CHR$(34)+"",8"
590 PRINT "{DOWN4}?"+CHR$(34)+"{CLEAR}"+CHR$(34)+"":RUN"
600 POKE 631,19: POKE 632,13: POKE 633,13: POKE 198,EX
610 END
870 REM -- ML ROUTINE DATA --
880 DATA 162,1,32,198,255,160,1,177,71,133
890 DATA 99,200,177,71,133,100,162,32,32,207
900 DATA 255,202,224,27,208,248,32,207,255,201
910 DATA 34,240,5,202,208,246,240,29,160,0
920 DATA 32,207,255,201,34,240,5,145,99,200
930 DATA 208,244,32,207,255,240,10,201,80,208
940 DATA 247,160,16,145,99,208,241,32,204,255
950 DATA 96

```

References

1. *Mastering Your VIC-20*, Phil Daley, Robert M. Tripp and Loren Wright, Micro Ink, P.O. Box 6502, Chelmsford, MA 01824, 192 pages.
2. *BASIC Line Delete for PET/CBM and VIC*, Thomas Henry, MICRO (No. 55), pp. 47-51.
3. *VIC/PET GOMOKU*, David Malmberg, MICRO (No. 54), pp. 59-61.
4. *VIC RS-232 Printer*, Michael V. Tulloch, MICRO (No. 57), pp. 17-20.
5. *Sorting Revealed*, Richard C. Vile, Jr., MICRO (No. 26), pp. 13-29.
6. *VIC Hi-Res Graphics Explained*, Nicholas J. Vrtis, MICRO (No. 56), pp. 11-13.
7. *Hi-Res Plotting with the VIC*, Fred Wallace, MICRO (No. 58), pp. 19-21.
8. *Commodore 64 Programmer's Reference Guide*, Commodore Business Machines, Inc. and Howard W. Sams & Co., Inc., 4300 W. 62nd Street, Indianapolis, IN 46268, 486 pages.
VIC Revealed, Nick Hampshire, Hayden Book Co., Inc., 50 Essex St., Rochelle Park, NJ 07662, 267 pages.
10. *Exploring Character Graphics*, Phil Daley, Robert M. Tripp and Loren Wright, MICRO (No. 66), several articles.

P-H Software



HAMVELOPE PAT. PEND.

**Mastering Your Commodore 64®
Through Eight BASIC Projects**

By the Staff of *MICRO Magazine*

© 1984, *The Computerist, Inc.*



Written and produced by the editors and programmers of MICRO magazine, this new book is your guide to complete mastery of BASIC on your Commodore 64. Learn quickly how to write your own programs, modify them, and add a wide assortment of features—all on your own.

You'll learn by doing. Each of the eight BASIC projects listed below begins with a complete running program... a practical utility or highly entertaining and challenging game. Then, detailed, easy-to-follow instructions show you how to add features. You can actually create your own software for your Commodore!

Eight Basic Projects... Dozens of Programs

MICRO CALC: a miniature spreadsheet program that makes complex, repetitive calculations a breeze. Can be used to experiment with BASIC functions.

MASTER: a guessing game for one or two players. Teaches programming with random numbers and flags.

COMMODORE 64™ CLOCK: a deluxe clock that displays time in giant-size numerals on the screen. Teaches ON...GOSUB function and character graphics.

WORD DETECTIVE: lets you discover the computer's rule for accepting or rejecting the words you type. Also teaches string manipulation functions.

COMMODORE 64™ PLAYER: shows you how to play musical tunes using the keyboard like an organ. Teaches you how to LOAD or SAVE files, how to program music, and how to use cassette data files.

SORTING: a "bubble" sort is demonstrated in two programs—one uses colorful character graphics, the other sorts a telephone directory. Other sorting methods are also described.

BREAKUP: use a paddle, a joystick, or the keyboard to play this exciting bouncing-ball game and learn how animation is accomplished with PEEKs and POKEs to screen memory.

SCREEN EDITOR: a full-screen editor that lets you experiment with the built-in characters and colors, save and load complete display screens to tape or disk, use specially designed graphic characters to make simple and complex line drawings, change screen, border, and character colors, and explore the wonders of character graphics.

Cover design by Mike Freeland
PRENTICE-HALL, Inc.,
Englewood Cliffs, New Jersey 07632

