SIGNET SPECIAL • 451-XE2090 • (CANADA $8.75) • U.S. $6.95

THE BIGGEST ASSORTMENT EVER OF PROGRAMS FOR AMERICA'S MOST POPULAR HOME COMPUTER

PORTER'S PROGRAMS FOR THE COMMODORE® 64™

FINANCIAL PLANNING
TIMEKEEPING
GAME CREATIONS
MATH MASTERY
SOUND SYNTHESIZING
TEXT PROCESSING
AND A LIBRARY OF
MORE THAN 65
PROGRAMS

HIGH-PERFORMANCE SOFTWARE
BY AN ACCLAIMED PROGRAMMER

KENT PORTER

THE WIDE-RANGING LIBRARY OF PROGRAMS EVERYONE CAN AFFORD.

The question that confronts all who bring home the marvelously affordable Commodore 64 is how to save money on the costly array of software designed to utilize all the remarkable powers of this superlative piece of hardware.

This invaluable book, written by an expert, provides you with programs you can type right into your machine by yourself without additional expense. Included, too, are models that show you how to write your own programs, and all the aids for mastering the full range of the Commodore 64's capabilities. Many of these programs are compatible with the VIC®20,™ and all of them will run on the SX 64.™

PORTER'S PROGRAMS FOR THE COMMODORE® 64™

KENT PORTER is renowned both for his computer expertise and his remarkable ability to write about computers in language that everyone can understand. His widely aclaimed books include *The New American Computer Dictionary, Beginning with Basic: An Introduction to Computer Programming, Mastering Sight and Sound on the Commodore® 64™, and Porter's Programs for the IBM® PC jr.*

BOOKS BY KENT PORTER

Computers Made Really Simple (1976)

Building Model Ships from Scratch (1977)

New American Computer Dictionary (NAL/Signet, 1983)

Mastering Sight and Sound on the Commodore® 64™
(NAL/Plume, 1984)

Porter's Programs for the Commodore® 64™
(Signet, 1984)

Practical Programming in Pascal (NAL/Plume, forthcoming)

Porter's Programs for the IBM® PC jr
(NAL/Signet Special, 1984)

Mastering the Coleco® Adam™
(NAL/Plume, 1984)

Beginning with BASIC: An Introduction to Computer Programming
(NAL/Plume, 1984)

PORTER'S PROGRAMS FOR THE COMMODORE® 64™

BY KENT PORTER

A SIGNET BOOK

NEW AMERICAN LIBRARY

SIGNET, SIGNET CLASSIC, MENTOR, PLUME, MERIDIAN AND NAL BOOKS are published by New American Library,
1633 Broadway, New York, New York 10019

First Printing, July, 1984

1 2 3 4 5 6 7 8 9

PRINTED IN THE UNITED STATES OF AMERICA

CONTENTS

INTRODUCTION

General

When people find out that I write books about personal computers, they often tell me that they're thinking about getting one, but they're not quite sure what they'd do with it. Once in a while, too, someone says, "Okay, I bought one. Now what?" The answer to both of these questions is that you can do an astonishing number of useful and entertaining things with a computer; all it takes is a little imagination and some programming.

This book contains some imagination and quite a number of programs for the Commodore 64, but they represent only a smattering of the possibilities. Not every program included here is for everybody; on the other hand, everybody will find something of value, including owners of VIC-20, who can run most of these programs on their machines, too. For convenience, I've broken the programs into broad groupings, such as math, money, measurements, and time. With a few exceptions that are noted as such, they're all "stand-alone" programs that you can simply type into the computer, run, and save for later recall and use. Thus, you can pick those that interest you and bypass those that don't. It's an

easy way to accumulate a fairly large library of programs that put your computer to work doing useful things for you. This book, then, is dedicated to people who don't quite know what to do with a computer, or who have one and want to do more with it.

The programs were all developed on, and therefore presuppose, the following "bare-bones" system:

- A Commodore 64 computer
- A color TV set as a monitor
- A Commodore Datassette tape recorder

Actually, you don't even have to have the recorder, but if you don't, you'll spend much time typing and little time really enjoying your computer. Some of the games also need a joystick, which is an inexpensive and fun addition to the system.

Now let me tell you what this book won't do. Except by example, it won't teach you to program and it won't reveal the secrets of the Commodore 64's special features. For those things you have to turn to two other books that I also wrote: *Beginning with BASIC* (1984: NAL/Plume, New York) and *Mastering Sight and Sound on the Commodore 64* (1984: NAL/Plume, New York). I hasten to point out that programming knowledge and expertise in the technics of the computer are not a requirement for benefitting from this particular book, since its programs incorporate all those things. All you have to do is type them, and even if you don't understand and you think it's a bunch of gobbledygook, they'll still work.

It is important, however, to understand the mechanics of typing and editing programs with the Commodore 64, which has some features and commands that make it much more than a typewriter that prints on a screen. For that reason, a tutorial follows that will familiarize you with program input and repair, and with the means of saving and fetching programs on tape cassettes and disks. If you already know about that, skip the rest of this introduction and we'll meet you among the programs.

Keyboarding on the Commodore 64

As our point of departure on this journey into the rudiments of working with a computer, I'll assume that you have hooked the

parts together according to the instructions that came with the machine, and you have just turned it on for the first time. Furthermore, I'll assume that you have never before touched a computer or even seen one in operation at close range.

My first advice, before we even glance at the screen, is: Don't be afraid. There is absolutely nothing you can do to damage this computer by pushing the wrong button, and as long as you don't knock it to the floor or drop something heavy on it, it's not going to break. Also, it won't giggle at you and consider you an idiot if you make a mistake (and I won't tell a soul, either). Type a half-dozen keys at random and notice their pleasant, springy feel. Press the RETURN key at the right end of the keyboard. Chances are the screen now shows the letters you typed with the next lines saying "?SYNTAX ERROR" and "READY." The random keys made no sense to the computer, and it said so. That's about the worst you can expect. Not so bad, is it? Type confidently, even if you don't feel that way right now. There's almost nothing you can do that you can't subsequently undo, and no reason to fear the machine.

Now look at the screen again. Across the top the computer tells you who it is and some other irrelevant information. Maybe your eyes are better than mine, but I think light blue on darker blue is hard to read. Let's change it. Hold down the button marked CTRL (stands for CONTROL) and press the 2. Notice that the front of the 2 key bears the legend WHT, meaning white. Release both keys and look at the screen. The letters already there are still light blue, but the cursor—that blinking square under READY.—is now white. Type the word HELLO and press RETURN. Again you get the SYNTAX ERROR message and READY., in addition to the word that you typed, but now the letters are white.

You can make the letters any of the colors shown on the fronts of the number keys by holding down CTRL and pressing that color's key. Try some different ones and watch what happens to the color of the cursor. It always blinks with the color that subsequent letters will take.

You can also change the colors of the screen and the border. Type the command POKE 53281,0 and press RETURN. Shazam! The screen turns black, but everything else remains the same. Now type POKE 53280,0 and press RETURN. That changes the border to black. These two commands control the colors of the screen and border, respectively, and the number after the comma

is one less than the color number on the front of the keycaps. For example, black is 0 as we've seen, and black is on the 1 key. To make the screen red, then, you'd type POKE 53281,2, because red is the 3 key and 3 − 1 = 2. Experiment with different combinations of letter, screen, and border colors until you find one you like. Each time you start up the computer, you can key these commands to set up your particular screen preferences.

You can always put the screen back to its original setup with one simple operation: hold down the RUN/STOP key and press RESTORE. Instantly, the display goes back like it was when the machine was first turned on, except that the identifying message no longer appears across the top. Try it and see what I mean, then put the screen back to the combination you prefer.

Each time you've keyed a command, I've told you to press RETURN at the end. That's because the RETURN button tells the computer that you're done typing this line and it can now act on it. The term RETURN is a holdover from the typewriter, where it causes the carriage to move physically to a new line. Here it triggers a similar action in that the cursor moves to the start of the next line, but it also serves as a "go-ahead" signal to the computer. You can type things on the Commodore 64 until tomorrow morning, but nothing will happen until you press RETURN.

The Commodore 64 accepts four kinds of entries from the keyboard: control button operations (such as CTRL-2 to make the letters white), commands, program lines, and answers to questions asked by programs. We won't talk about the last kind here; they are obvious when they happen in the programs later in the book. The one- or two-button control commands play a part in editing as well as in changing letter colors, and we'll discuss them soon.

A *command* is an instruction that you expect the computer to act on immediately. The POKEs we did to change screen and border colors are commands because they told the computer to take an action and produce a discernible result. Other commands are:

RUN	Causes a program to operate.
NEW	Clears away program lines in the computer and prepares it to accept a new program.
LIST	Displays the program.
SAVE	Writes the program to tape or disk.
LOAD	Reads a program from tape or disk.

There are others as well, but these are the ones you'll use most.

A command doesn't always alter the status of the machine, however. As an example of one that doesn't, type

```
PRINT "HI, THERE"
```

(You didn't forget to press RETURN, did you?) The message

```
HI, THERE
```

immediately appears under your command, because that's what you told the computer to do.

You can use this ability to act on commands not only to control the computer, but to do useful work such as calculations. Suppose you want to know the sum of 15.25, 1091, and 66.81. You can have the computer tell you by typing

```
PRINT  15.25 + 1091 + 66.81
```

and as soon as you press RETURN, it displays the answer (1173.06). The arithmetic operators in BASIC are:

+	Addition
–	Subtraction
*	Multiplication
/	Division

Consequently, if you want to multiply the first two numbers and divide by the third, you can type

```
PRINT 15.25 * 1091 / 66.81
```

and the answer 249.030834 instantly appears. The PRINT instruction tells the computer to do whatever follows and display the result on the screen. Clever, *n'est-ce pas*? Incidentally, if you type simply

```
PRINT
```

you have told it to print nothing, and that's what it does; it just makes a blank line on the screen.

After every command is executed, the computer displays the word READY. This indicates that it is ready to accept your next bidding. When it doesn't understand the command, it says

```
?SYNTAX ERROR
READY.
```

and that means "Huh?"

You can put two or more commands on the same line if you separate them with colons (:). For instance, to change the screen and border to black in one fell swoop, type

```
POKE 53281,0: POKE 53280,0
```

and the computer will act on both at the same time.

The screen of the Commodore 64 holds 25 rows of 40 characters each. Sometimes a line is longer than 40 characters, but that doesn't matter. Just keep typing and the letters will begin to appear on the next row. Even if a word or a number gets split, the computer still understands it and thinks of it as though it were all together. A logical line on the Commodore 64 is up to 80 characters, or in other words, two full rows. If a line is so long it slops into a third row, you'll have trouble because that comes to more than 80 characters. A few lines in the programs in this book go to two rows, but none go to three. The listings show the spillover, and you can identify lines that have gone to a second row because the continuation part lacks a line number.

Single-line commands are good for getting quick results to simple actions, but clearly they won't do for complex processes involving numerous calculations and decisions and so on. For that we need multiple lines of instructions with some indication of the order in which we want them performed, and that's what a program is. Fundamentally, a program is a logical sequence of commands that the computer stores and acts on at some future time in order to produce a reasoned result. The Commodore 64 recognizes a program line by the fact that it has a number in front of it. An example is the "HI, THERE" command we did a while back. The command itself read

```
PRINT "HI, THERE"
```

We can make it a program, in contrast to a command, by typing

```
100  PRINT "HI, THERE"
```

When you press RETURN, nothing happens. Why not? Because the instruction has a number in front of it, and that's a program line, and program lines get stored up to be acted on later.

Now it's later and we can make the computer act on it with

```
RUN
```

And *voilà!*, the friendly little message appears. We have written one of the world's shortest computer programs, a one-liner.

Hold down the SHIFT key and press the CLR/HOME button near the upper right corner of the keyboard. Whoops! The screen went blank. Now let's suppose we can't remember whether there's a program in the computer, or if there is, what it is. To get it back, type

```
LIST
```

Our dinky little program appears. The screen may have forgotten what it held, but the computer still remembers the program. Clear the screen again and type

```
RUN
```

The message appears just as before, as a result of the program running.

A one-line program isn't worth a whole lot, so let's make it bigger. Type the following:

```
 90 PRINT CHR$(147)
110 PRINT
120 PRINT "I'M A DUMB OLD COMPUTER"
RUN
```

The result should be that the screen clears and the following appears on it:

```
HI, THERE
I'M A DUMB OLD COMPUTER
READY.
```

Now let's see the program as a whole by typing LIST. The computer displays the program, which reads:

```
 90 PRINT CHR$(147)
100 PRINT "HI, THERE"
110 PRINT
120 PRINT "I'M A DUMB OLD COMPUTER"
```

A couple of things need explanation. First, line 90 does the same thing as holding down SHIFT and pressing CLR/HOME, but by means of a program instruction. The computer can't reach out and push its own buttons, so all the control keys have corresponding

codes. Second, you'll recall that we typed line 100 first and ran it a few times before we typed lines 90, 110, and 120. The line numbers are significant in that they decree the order in which the computer performs the statements. The values of the line numbers are themselves of no real concern, but their numeric order is. The computer knows this, and so it automatically merges new lines with old lines in the proper numeric sequence. In this case, it put line 90 in front of the existing line 100, and the other lines after line 100 to achieve the proper order. This is important to know, because if you type a program and leave a line out by accident, you can always go back and type only the missing line and the computer will put it into the right place.

Similarly, you can replace one line with another that has the same line number. To illustrate, type

```
110  PRINT "LET ME INTRODUCE MYSELF"
RUN
```

The output now looks different. It says

```
HI, THERE
LET ME INTRODUCE MYSELF
I'M A DUMB OLD COMPUTER
```

You can see the effect of the new line 110 by listing the program. Where before line 110 simply read PRINT, now it holds the more recent instruction. The old line was replaced.

Finally, you can remove a line entirely from the program by typing its line number and pressing RETURN. Type 110 and hit RETURN, then run and list the program again. The "LET ME . . ." is gone, and so is line 110, without a trace.

One of the very first things we did in this tutorial was to make a mistake on purpose, to show you that the computer won't bite. We did that by typing a nonsense command, and the computer came back with a syntax error message. You can also get a syntax error from a bad program instruction. It doesn't show up when you type the statement, but it does when the computer encounters it as the program runs. To illustrate what happens, enter the following meaningless instruction into our little program:

```
110  SKLRGX
```

The computer accepts it initially, when you press RETURN, be-

cause it makes no attempt to interpret program lines as they are keyed. The interpretation happens after you type RUN. Do it and watch. See, the display reads

```
HI, THERE
?SYNTAX ERROR IN 110
READY.
```

The computer performed the instructions until it got to line 110. It couldn't understand that line, so it threw up its hands and quit, doing you the minimal courtesy of identifying the line that contains the error. The ?SYNTAX ERROR message is a catchall that means "I don't know what's wrong, but something is." The computer also issues other messages that are more specific. For example, it is mathematically impossible to divide by zero. The computer can diagnose this problem, so if it encounters a statement such as

```
PRINT 90 / 0
```

it tells you what's wrong. Type the command and see for yourself. Before we go further, erase line 110 from the program. Remember how? (*Hint:* Type the line number and hit RETURN.)

Just below the RETURN button are two keys marked CRSR, whose purpose is to move the cursor about the screen. You can use these keys to edit your programs and correct mistakes without having to retype entire lines. Study these keys and you'll see that the one on the right has arrows that point left and right, and the one on the left, up and down. The bottom arrow on each key shows which direction the cursor moves normally when you press the key. Push the right-left CRSR key and watch the cursor move right. Now push the up-down key and the cursor moves down. These are *auto-repeat keys,* meaning that if you hold them down, their action repeats until you release them. There is a slight delay after the first move, so that you have time to release before the cursor goes skittering. That way you can move just one position.

To reverse the actions of the CRSR keys, you have to hold the SHIFT button down. Press SHIFT and the up-down key to make the cursor move upward. Similarly, operate SHIFT and the right-left key to make the cursor go to the left. This arrangement seems awkward at first, but you'll soon be operating the CRSR keys without even looking at them.

Use the up-down CRSR key to move the cursor to the bottom of

the screen. As it reaches the bottom, watch what happens to the information displayed above. It moves upward one row for each attempt the cursor makes to move downward. The cursor can't move off the bottom of the screen, so each time it tries, it instead pushes the rest of the display up. This phenomenon is called *scrolling*. It's sort of the same idea as the paper advancing upward through a typewriter, but there's one big difference: you can't wind it back the other way (downward). A line that scrolls off the top of the screen is gone. If it's a program line, you can get it back, of course, with the LIST command, but other display lines simply evaporate from the top of the screen.

Scrolling is the computer's way of dealing with a full screen. In effect, it adds a line at the bottom and pushes everything else up. For an example of automatic high-speed scrolling, type the following command and try to watch the top and the bottom of the screen at the same time.

```
FOR L=1 TO 2000: PRINT "+" ;: NEXT
```

This command prints 2000 plus signs. Since the cursor was at the bottom of the screen when it began writing plus signs, it always placed its output on the bottom row. When the line filled up, it pushed the whole screen up and continued streaming symbols across the bottom. Eventually the screen filled with plus signs, up to 1000 of them, which is the capacity of the Commodore 64 display. It kept on for another 1000 (for a total of 2000), but the scrolling off the top was no longer apparent since each top line was replaced by another identical to it.

For another example of scrolling a more meaningful output, try this command:

```
FOR L=1 TO 500: PRINT L,: NEXT L
```

Now you get a list of all the numbers between 1 and 500, arranged across four tabular columns. The screen can't hold that much output, so it scrolls.

Now let's go back to some uses of the CRSR keys. Type LIST to view the little program still lurking in the computer's memory. Sure enough, there it is, reading:

```
 90  PRINT CHR$ (147)
100  PRINT "HI, THERE"
120  PRINT "I'M A DUMB OLD COMPUTER"
```

Let's say we want to change the word OLD in line 120 to NEW. To do this, hold down SHIFT and move the cursor up to line 120, then release SHIFT and move the cursor right until it's on top of the "O" in OLD. Type NEW right over the word OLD. The line now reads

```
120 PRINT "I'M A DUMB NEW COMPUTER"
```

and the cursor is flashing between NEW and COMPUTER. You can see the difference, but the computer regards it as only a tentative change. The way to make the change effective is to press RETURN. (If you used the CRSR controls to move the cursor to a different row, the computer would have disregarded the change in line 120.) The cursor is now over the "R" in READY. Move it down with the CRSR button and type RUN. Aha! The program now introduces itself as a DUMB NEW COMPUTER because we modified the program.

Maybe you dislike the idea of making the computer so disrespectful of itself, even though it has no feelings about the programs you give it to run. Let's let it defend itself against self-slander by inserting a word into line 120. Type LIST to get the program listing back, then move the cursor up to line 120 and right until it's on the "A" after the contraction I'M. (If you overshoot the mark, use the right-left CRSR key with SHIFT pressed to back up to the right spot.) Hold down SHIFT and press the INST/DEL key four times. This abbreviation means "INSERT/DELETE," and the key itself is in the upper right corner of the keyboard. Observe what happens each time you press it. The text starting at "A" shifts one position to the right, while the cursor itself and everything to its left remain in place. The effect is to open space in the line. Note that the "A" that was originally under the cursor also shifted. Now type the word NOT into the newly opened area. As before, you can make the modification "official" by pressing RETURN. With the down-arrow key, move the cursor off the word READY and type RUN. The computer now maintains that it is not dumb.

We've now seen how to make replacements and insertions with the CRSR keys. What about deletions; that is, taking things out of a program line? For that, we'll enclose a lesson within a lesson and begin by learning about a useful extension to the LIST command.

LIST as we've used it so far displays the entire program, which is fine for a bitty little one like our sample, but not so good when the program is bigger than the screen. If this program had 40 lines, LIST would never let us see more than 25 at a time, since that's how many rows there are on the display, and because of scrolling we'd never get a clear look at the first 15 lines or so and wouldn't be able to maneuver the cursor into them. For this reason, LIST permits us to display one line or a selected range of lines.

First type the simple command LIST to get the entire program on the screen. That way, you can more readily see the effect of the two exercises that follow. To single out one line for viewing, you can enter LIST with the line number after it. As an example, type

```
LIST 100
```

and the computer displays only that line, and not the rest of the program. Similarly, to select a range of lines, type LIST followed by the first and last line numbers you want to see, with a hyphen (minus sign) between them. You can view lines 90 through 100 with

```
LIST 90 -100
```

and the computer shows you all the lines within that range of line numbers. If we had a line 95, it would be there too, and so would 93 and 97, if they existed.

The original program that we're extracting from doesn't have to be on the screen, of course, as it is here. Were that the case, the LIST extensions wouldn't be very useful. Clear the screen by pressing SHIFT and the CLR/HOME button. Now it's empty. Get line 120 with the command

```
LIST 120
```

and it appears near the top of the screen, followed by READY.

As long as a line is on the screen, you can change it with the CRSR motions and methods we've described. As an illustration, and to demonstrate the deletion of characters from a line, let's take out the word NEW. Move the cursor up to the line and run it out until it's in the space after the "W" in NEW. Press the INST/DEL key one time and watch what it does. It deletes the character to its

left—the "W"—and closes the line as though the "W" had never existed. Press it four more times. Oops! We took out the "B" in DUMB, deleting too much from the line.

What to do? Simple: Insert a space by holding down SHIFT and pressing the INST/DEL key, then type a "B". You can continue editing this line for the rest of the week, and none of the changes become effective until you press RETURN. Do that, then move the cursor down from READY and type RUN. The computer says it's NOT A DUMB COMPUTER (with NEW no longer there), because we've deleted that word from the instruction.

Chances are you still feel a little timid about these editing steps, but soon they'll become second nature to you. If you mangle a line beyond redemption, you can always bail out by pressing RETURN, then moving the cursor down to an empty row and retyping the line entirely, even if it's out of sequence.

Saving and Retrieving Programs on Cassette Tape

The Commodore 64 *User's Guide* does a dubious job of explaining this subject, so I'll share with you the fruits of some hard-gained experience.

The Commodore Datassette tape recorder is a special device designed to work only with Commodore computers. It attaches via a cable to a slot in the rear of the computer and requires no external power source, instead drawing its current from the computer. The first rule is *don't connect or disconnect the recorder while the computer is powered on.* This can cause surges that will damage the innards of the Commodore 64. Thus, if you plan to use the recorder, attach it before you turn on the computer.

The recorder, though a special device, takes standard audio tape cassettes, and it records on both sides just like a normal tape unit. It does this by writing tones on the tape, and if you move a data tape to an audio player and listen to it, you will hear a dreadful racket. Audio signals are also recorded as tones, and the computer can't figure out which is which. Consequently, *keep data tapes separate from audio tapes and don't try to mix the two kinds of signals on the same cassette.*

Disk drives manage the placement and retrieval of program files

automatically, but on tapes it's up to you. You'll have to make a list of programs on each cassette and where they are.

The "where they are" part involves the counter on the Datassette unit. The first time you mount a cassette, rewind it to the start and, when the unit stops, push the button next to the counter. This sets the counter at 000. After you have saved a program and the recorder stops, note the counter setting. It shows the end of the used space on the cassette. The next time you save a program on this cassette, rewind it, push the counter button, and then advance with fast-forward to the end of the last program stored. Write the starting and ending counter setting of each program next to its name on the list of programs the cassette contains. That way, you'll be able to find it and load it quickly, without searching the tape.

Cassette operation is agonizingly slow. That's the price you pay for cheap storage. Some of the programs in this book take up to five minutes to save and retrieve, and during that time the screen is blank and the computer is out of touch. It's a long time, but it's the way it is. If you want faster operation, get a disk drive.

To save a program:

1. Mount the tape cassette and position it to the place where you want to write the program, as described above.
2. Type the word SAVE followed by the program name in double quotes. For example, to save the program named LOAN TERMS, type

   SAVE "LOAN TERMS"

3. The computer responds with the message

   PRESS PLAY AND RECORD ON TAPE

   which means you press those two buttons down at the same time. They lock in place.
4. The screen blanks and the recorder advances the tape, writing the program onto it.
5. After a time, the screen comes back with the message READY. Press the STOP button on the tape recorder to unlock PLAY and RECORD.
6. Note on the cassette's "table of contents" the name of

the program and its starting and ending points from the counter.

7. Rewind the tape and remove it.

Now let's say it's a few days later and you want to use the LOAN TERMS program. To retrieve it from tape:

1. Mount the cassette in the recorder.
2. Check your list and find out where the program is by the counter setting.
3. Make sure the tape is rewound and the counter reads 000.
4. Use fast forward to advance the tape to a point a little short of the program start.
5. Type the command

   LOAD "LOAN TERMS"

6. The computer responds

   PRESS PLAY ON TAPE

   You should do as it says.
7. The screen blanks and the recorder advances the tape. Eventually, the screen comes back with the message

   FOUND LOAN TERMS

8. Press the Commodore button on the keyboard (lower left corner, marked "C=").
9. The screen blanks again while the program is read from the tape into the computer. When it again turns on, it says READY.
10. Press STOP on the recorder, then rewind the tape back to the start.
11. You can now run the program, list it, make changes, etc.

Maintenance of the list of cassette contents is very important. Otherwise you won't know what's on your cassettes, or where. If you make changes to a program that was previously saved on tape, don't save it in the same place as the old copy. The reason is that, if the new version is longer than the old, it will overlay the start of the next program. Instead, save the new version at the end of the

tape and change the starting and ending points written on your list next to that program's name.

And that's what you need to know to gain the full benefit of all the nifty programs that fill the remainder of this book. Enjoy!

SECTION 1

PROGRAMS HAVING TO DO WITH MATHEMATICS

ENIAC

The first machine that could accurately be called a computer by today's standards was the Electronic Numerical Integrator and Calculator, or ENIAC. It was developed during World War II at lavish expense, occupied a space 30 × 50 feet, stood about 12 feet high, and needed a legion of magicians to operate it. Programming it was a tedious task involving plugboards, whose multitudinous connections were designed by a tiny, select society of "high priests" according to principles so exotic that it seemed ordinary people would never be able to understand them. And when it was done, the government proudly announced that it could compute the cube root of 2589 to the 16th power "in a fraction of a second."

How far we have come in so short a time! Before you sits the Commodore 64, a machine that costs a few hundred dollars and that you can carry around under your arm. It takes no great genius to program it, and you can operate it all by yourself with only a little training, usually self-taught. And as for its speed in doing the much-publicized ENIAC calculation. . . . Well, enter the following brief, non-plugboard program and see for yourself how it stacks up against the ENIAC.

```
NEW
100 REM  ** ENIAC
110 T = TIME
120 X = EXP(LOG(2589) * 16 / 3)
130 T2 = TIME
140 PRINT "CUBE ROOT OF 2589 TO 16TH POW
ER =" X
150 PRINT "TIME" (T2 - T) / 100 "SEC"
160 END
RUN

Sample run:

        CUBE ROOT OF 2589 TO 16TH POWER = 1.59724278E+18
        TIME .04 SEC

        READY.
```

"A fraction of a second" indeed! The Commodore 64 does it in 4/100 of a second. The government didn't say how many fractions of a second, but surely this inexpensive little computer compares favorably with ENIAC. And it can do a lot more than ENIAC ever could, since that huge hulking collection of vacuum tubes and equipment bays was designed chiefly to calculate the trajectories of artillery shells.

Perhaps we should explain the answer the Commodore 64 came up with, since "E + 18" is a strange-looking suffix for a number. Now and again you might see numbers such as this in the output of the computer. It's a form of scientific notation, which in this case can be transformed into the more familiar $1.59724278 \times 10^{18}$. Still doesn't make much sense to you? It means "shift the decimal point 18 positions to the right," giving the number

1,597,242,780,000,000,000

which is approximately the number of miles traveled by a ray of light in 272,300 years. The Commodore 64, able to produce up to nine digits at a time, reports very large numbers in this format.

It also reports very small numbers in a similar way. The plus sign is replaced by a minus sign, meaning "shift the decimal point to the left by the number of positions indicated." For example, the number 1.59724278E − 18 translates into

0.00000000000000000159724278

(that's 17 leading zeros, plus the 1 for 18 places), which is in the realm of the weights of individual atoms.

The Commodore 64 can represent shifts of up to ±37 decimal points in this fashion. An electron weighs 4.14×10^{-31} pounds and the earth weighs $1.70481048 \times 10^{13}$ pounds, both of which are well within the Commodore 64's range of numbers.

ENIAC, for all its gigantic size and complexity and cost, couldn't express numbers this small and large. It couldn't do a lot of other things the Commodore 64 can, either. The programs that follow give a sampling of this computer's capabilities and furnish a useful and entertaining library.

Integral Factors of a Number

Integers are whole numbers (with no fractional or decimal component) such as 2, 9, and 5280. The number 3.14159 is not an integer because it has a fractional part. *Factors* are the divisors of a number; the factors of 6 are 2 and 3. It's also true that 4 and 1.5 are factors of 6, but they are not integral factors because one of them has a fractional part.

This program takes any number and computes all its integral factors (except the obvious one of 1 times the number itself). If the number has no integral factors, the program tells you so.

```
NEW
100 REM  ** FACTORS OF A NUMBER
110 PRINT CHR$(147): R = 0
120 PRINT "INTEGRAL FACTORS": PRINT
130 INPUT "NUMBER... "; N
140 FOR F1 = 2 TO (N / 2) -1
150 F2 = N / F1
160 IF F2 <> INT(F2) THEN 190
170 PRINT F1, F2
180 R = R + 1
190 NEXT F1
200 IF R = 0 THEN PRINT "NONE"
210 END
RUN
```

(continued)

```
Sample run:

        INTEGRAL FACTORS

      NUMBER... ? 63
       3          21
       7          9
       9          7
       21         3

      READY.
```

Prime Numbers

Primes are numbers that cannot be divided without producing a fraction, or, in other words, numbers only divisible by 1 or by themselves. An example is the number 5: there are no numbers that you can multiply except 1 and 5 to produce 5, and as a result, 5 is prime. In a sense, this is the opposite of the preceding program in that it finds numbers for which there are no factors.

A program that computes all the prime numbers from 1 to an upper limit (N) is called the Sieve of Eratosthenes. It does a gigantic amount of work, for an admittedly trivial result. Computer people use a Sieve to compare the speeds of two or more machines or of two or more programming languages running on the same computer, in order to find out which is the most efficient. This process is called "benchmarking," and it often plays an important part in selecting a particular product over others.

These are several ways of calculating prime numbers. It's not important which method is used in benchmarks, so long as the same method is used consistently for all tests. This program multiplies all the numbers from 2 to N/2 and marks a list for each product. The second phase then scans the list and prints all the entries not marked, which are the prime numbers. Consequently, the list has "holes" indicating the prime numbers, hence the analogy to a sieve.

Because the purpose of this kind of program is usually to measure running time for the calculations, the program times itself during that phase. At the end of the list, it reports how many

seconds it took to compute all nonprime numbers. Output time is not measured. If the screen fills, the program pauses until you press RETURN. *Beware:* The higher the limit, the longer this program runs, and time increases as the square of the limit. If you set the limit at something huge like *10000*, go to bed and come back in the morning; the Commodore 64 sets no speed records.

```
NEW
100 REM  ** ERATOSTHENES SIEVE
110 PRINT CHR$(147): PRINT
120 PRINT "PRIME NUMBERS"
130 INPUT "  UP TO WHAT"; N
140 DIM P%(N)
150 PRINT: PRINT "THINKING"
160 T1 = TIME
170 FOR X = 2 TO (N / 2)
180 FOR Y = 2 TO (N / 2)
190 I = X * Y
200 IF I > N THEN 220
210 P%(I) = 1
220 NEXT Y
230 NEXT X
240 T2 = TIME
250 REM  ** DISPLAY RESULTS
260 I = 0
270 FOR X = 1 TO N
280 IF P%(X) <> 0 THEN 320
290 PRINT X,
300 I = I + 1
310 IF I = 96 THEN GOSUB 370
320 NEXT X
330 ET = (T2 - T1) / 60
340 PRINT
350 PRINT "ELAPSED TIME =" TT "SEC"
360 END
370 REM  ** FULL SCREEN
380 PRINT "PRESS RETURN FOR MORE..."
390 GET X$: IF X$ = "" THEN 390
400 I = 0: RETURN
```

(continued)

```
RUN


Sample run:

        PRIME NUMBERS
          UP TO WHAT? 300

        THINKING
         1        2        3        5
         7        11       13       17
         19       23       29       31
         37       41       43       47
         53       59       61       67
         71       73       79       83
         89       97
        ELAPSED TIME = 21.75 SEC

        READY.
```

Powers of a Number

Everybody knows that 3^2 means "3 squared" or, putting it another way, "3 raised to the power of 2." And everybody knows how to do the calculation; you multiply the number by itself the number of times indicated by the power.

But what about raising 3 to the 3.14159 power? Or worse yet, 3 to the −3.14159 power? How can you multiply a number by itself a fractional and/or negative number of times? It's possible, but it ain't easy. Pity the poor Greeks, who figured out this kind of stuff without even the benefit of an adding machine. Computers make things like this simple. To wit:

```
NEW
100 REM  ** POWERS OF A NUMBER
110 PRINT CHR$(147)
120 PRINT "NUMBER RAISED TO A POWER:"
130 INPUT " WHAT IS THE NUMBER "; N
140 IF N = 0 THEN END
150 INPUT " RAISE TO WHAT POWER"; P
```

```
160 PRINT: PRINT "ANSWER IS" N ^ P
170 PRINT: GOTO 130
RUN

Sample run:

        NUMBER RAISED TO A POWER:
         WHAT IS THE NUMBER ? 3
         RAISE TO WHAT POWER? 3.14159

        ANSWER IS 31.5441888

         WHAT IS THE NUMBER ? 3
         RAISE TO WHAT POWER? -3.14159

        ANSWER IS .0317015602

         WHAT IS THE NUMBER ? 0

        READY.
```

Roots of Real Numbers

The last program concerned powers, and this one has to do with roots. They're really the same things viewed from opposite directions: if 2^4 is 16, then the 4th root of 16 is 2. But while raising a number to a power is somewhat intuitive (at least when the power is a whole number), it's quite a different ball game to find a root. Some we know by heart because we can work the times tables backward: the square root of 25 is 5, for instance. Cube roots are a little harder, but still possible, as in the cube root of 27 = 3. But what about the 17th root of 266.3? Or the −5.61 root of 427.616? Now we're in trouble.

Enter Our Hero, a program to find any root of a real number with a minimum of fuss and bother. This program, in line 170, divides the logarithm of the number by the magnitude of the root and converts the answer back to an ordinary number. That's something few of us can do in our heads, and not many more can do on paper, but you don't have to be a math wizard to work the program

(nor do you even have to understand the process). Suffice it to say that it works and gives an accurate answer.

Before proceeding, however, we need to point out that the term "real number" is significant in this case. No one has yet come up with a way to find the root of a negative number, which is why mathematicians call such phenomena "imaginary numbers." They exist in theory, but not in fact. Computers only work with facts, so this program will not and cannot provide roots of negative numbers. If you try to make it, it will kick out the answer "NO CAN DO" and ask you for another, more reasonable input.

```
NEW
100 REM  ** ROOT OF A NUMBER
110 PRINT CHR$(147)
120 PRINT "ROOT OF A REAL NUMBER:"
130 INPUT " WHAT IS THE NUMBER"; N
140 IF N = 0 THEN END
150 IF N < 0 THEN PRINT "NO CAN DO": GOT
O 190
160 INPUT " FIND WHAT ROOT     "; R
170 A = EXP(LOG(N) / R)
180 PRINT "ANSWER IS" A
190 PRINT: GOTO 130
```

```
Sample run:

         ROOT OF A REAL NUMBER:
          WHAT IS THE NUMBER? 266.3
          FIND WHAT ROOT     ? 17
         ANSWER IS 1.38889333

          WHAT IS THE NUMBER? 427.616
          FIND WHAT ROOT     ? -5.61
         ANSWER IS .339630298

          WHAT IS THE NUMBER? 0

         READY.
```

Trigonometry #1

The world abounds with practical applications of trigonometry, the mathematics of triangles and curves: figuring out how high something is, determining our position relative to other objects, surveying, and so on. Few of us think of the world in trigonometric terms, of course, instead living our lives by eyeballing and approximating. When we do come up against a situation demanding more accuracy than a guess, we are suddenly reminded how complex the world is.

Trigonometry is not a simple flavor of mathematics. It relies on obscure measurements of relativity called sines and cosines and tangents and what not, the derivations and purposes of which remain cloaked in confusion for most people. With this and the following two programs, which deal with classic problems of trigonometry, you don't have to know anything about the mechanics of triangles or trigonometric functions. These programs return accurate information about any form of triangle: scalene, obtuse, right, equilateral, or you name it.

A note regarding mathematical notation in triangles: the sides are called A, B, and C, and the angle opposite side A is called angle A, the angle opposite side B is angle B, etc. These programs use that customary notation.

The first program takes the lengths of two sides and the degrees of their included angle (the *included angle* is the one between the two sides whose measurements are given) and tells you the length of the other side and the degrees of the other two angles.

```
NEW
100 REM  ** TRIGONOMETRY #1
110 REM  ** 2 SIDES AND INCLUDED ANGLE
120 PRINT CHR$(147): PRINT
130 PRINT "TRIGONOMETRY #1:": PRINT
140 INPUT "  SIDE A    "; SA
150 IF SA = 0 THEN END
160 INPUT "  SIDE B    "; SB
170 INPUT "  ANGLE C   "; AC
180 IF AC > 180 THEN AC = 360 - AC
190 AC = AC * (π / 180)
```

(continued)

```
200 C2 = SA^2+SB^2-2*(SA*SB*COS(AC))
210 SC = SQR(C2)
220 BC = (SC^2+SA^2-SB^2)/(2*SA*SC)
230 BS = SQR(1 - BC^2)
240 AB = ATN(BS / BC)
250 IF AB < 0 THEN AB = AB + π
260 AA = π - (AB + AC)
270 PRINT: PRINT
280 PRINT "SIDE C      " SC
290 PRINT "ANGLE A     " AA / (π / 180)
300 PRINT "ANGLE B     " AB / (π / 180)
310 PRINT: PRINT
320 GOTO 130
RUN
```

Sample run:

```
        TRIGONOMETRY #1:

          SIDE A     ? 6
          SIDE B     ? 5.5
          ANGLE C    ? 112

        SIDE C       9.538031
        ANGLE B      35.6798008
        ANGLE C      32.3201992

        TRIGONOMETRY #1:

          SIDE A     ? 0

        READY.
```

Trigonometry #2

The second classic exercise in trigonometry resembles the first, except that this time it solves the triangle with *two* angles and *one* included side (between the two given angles). The same notation applies as before.

```
NEW
100 REM  ** TRIGONOMETRY #2
110 REM  ** 2 ANGLES AND INCLUDED SIDE
120 PRINT CHR$(147): PRINT
130 PRINT "TRIGONOMETRY #2:": PRINT
140 INPUT "  ANGLE A   "; AA
150 IF AA = 0 THEN END
160 INPUT "  ANGLE B   "; AB
170 INPUT "  SIDE C    "; SC
180 IF AA > 180 THEN AA = 360 - AA
190 AA = AA * (π / 180)
200 IF AB > 180 THEN AB = 360 - AB
210 AB = AB * (π / 180)
220 AC = π - (AA + AB)
230 SA = EXP(LOG(SC) - LOG(SIN(AC)) + LO
G(SIN(AA)))
240 SB = EXP(LOG(SC) - LOG(SIN(AC)) + LO
G(SIN(AB)))
250 PRINT: PRINT
260 PRINT "ANGLE C    " AC / (π / 180)
270 PRINT "SIDE A     " SA
280 PRINT "SIDE B     " SB
290 PRINT: PRINT
300 GOTO 130
RUN
```

Sample run:

```
        TRIGONOMETRY #2:

          ANGLE A   ? 35.68
          ANGLE B   ? 32.32
          SIDE C    ? 9.54

        ANGLE C     112
        SIDE A      6.00126769
        SIDE B      5.50110518
```

(continued)

```
TRIGONOMETRY #2:

  ANGLE A    ? 0

READY.
```

Trigonometry #3

The third and last classical trigonometry problem we'll consider. . . . Mathematicians love the word consider: "Consider the synthetic logarithm of the inverse hyperbolic cosecant of. . . ." How's that again? Anyway, the third case we'll take is that of a triangle in which the lengths of all three sides are known, and we need to find the angles.

```
NEW
100 REM  ** TRIGONOMETRY #3
110 REM  ** 3 SIDES
120 PRINT CHR$(147): PRINT
130 PRINT "TRIGONOMETRY #3:": PRINT
140 INPUT "  SIDE A    "; A
150 IF A = 0 THEN END
160 INPUT "  SIDE B    "; B
170 INPUT "  SIDE C    "; C
180 AC = (B^2 + C^2 - A^2) / (2 * B * C)
190 AS = SQR(1 - AC^2)
200 AA = ATN(AS / AC)
210 IF AA < 0 THEN AA = π + AA
220 BC = (C^2 + A^2 - B^2) / (2 * A * C)
230 BS = SQR(1 - BC^2)
240 AB = ATN(BS / BC)
250 IF AB < 0 THEN AB = π + AB
260 AC = π - (AA + AB)
270 PRINT "ANGLE A    " AA / (π / 180)
280 PRINT "ANGLE B    " AB / (π / 180)
290 PRINT "ANGLE C    " AC / (π / 180)
300 PRINT: PRINT
310 GOTO 130
RUN
```

```
Sample run:

        TRIGONOMETRY #3:

          SIDE A     ? 9.54
          SIDE B     ? 6
          SIDE C     ? 5.5
        ANGLE A      112.035176
        ANGLE B      35.6611013
        ANGLE C      32.3037226

        TRIGONOMETRY #3:

          SIDE A     ? 0

        READY.
```

Factorial!

In working with binomials and other algebraic mysteries, it is occasionally useful to calculate the factorial of a number, which mathematicians write as n!. A factorial is the product of all the integers leading up to the number in question, so that, for example, $4! = 1 \times 2 \times 3 \times 4 = 24$. This program computes any factorial up to 33!. Beyond that point, the product is too large a number for the computer to express. It repeats until you enter 0 in response to the prompt.

```
NEW
100 REM  ** FACTORIAL!
110 PRINT CHR$(147)
120 PRINT "PRODUCT OF A SERIES OF INTEGE
RS:"
130 INPUT "FACTORIAL"; N
140 IF N = 0 THEN END
150 F = 1
160 FOR X = 1 TO N
170   F = F * X
```

(continued)

```
180 NEXT X
190 PRINT N "! =" F
200 PRINT: PRINT: GOTO 130
RUN
```

```
Sample run:

        PRODUCT OF A SERIES OF INTEGERS:
        FACTORIAL? 5
         5 ! = 120

        FACTORIAL? 11
         11 ! = 39916800

        FACTORIAL? 0

        READY.
```

Distance Between Two Points

It's often useful when working with graphs and grids that have coordinate systems to find the distance between two points. That's a laborious task by hand, but simple for a computer, as the following program illustrates.

If you're uncertain about the meaning of X and Y coordinates, we'll explain briefly. A graph has a vertical line that represents the "0" point for horizontal measurements and a horizontal line representing the "0" point for vertical measurements. These reference lines are called the Y and X axes, respectively. Any point on the graph can be described by its horizontal distance from the vertical axis (its X coordinate) and by its vertical distance from the horizontal axis (its Y coordinate). A point 3 units to the right of the vertical axis has an X coordinate of 3, and if it's 5 units above the horizontal axis, its Y coordinate is 5. Thus, its position is defined as X = 3, Y = 5. Any point on the graph can be defined in this manner. By custom, X coordinates to the left of the vertical axis have negative numbers and Y coordinates below the horizontal axis are also negative. Consequently, a point opposite the one discussed above

(i.e., 3 units *left* of the vertical axis and 5 units *below* the horizontal axis) has the coordinates X = −, Y = − 5. The program finds the distance between any two points so defined.

```
NEW
100 REM  ** DISTANCE BETWEEN 2 POINTS
110 PRINT CHR$(147)
120 INPUT "POINT #1 X, Y..."; X1, Y1
130 INPUT "POINT #2 X, Y..."; X2, Y2
140 D = SQR((X1 - X2)^2 + (Y1 - Y2)^2)
150 PRINT: PRINT "DISTANCE =" D
160 PRINT: PRINT
170 INPUT "ANOTHER (Y/N)"; R$
180 IF R$="Y" THEN PRINT:PRINT:GOTO 120
190 END
RUN


Sample run:

        POINT #1 X, Y...? 3,5
        POINT #2 X, Y...? -3,-5

        DISTANCE = 11.6619038


        ANOTHER (Y/N)? Y


        POINT #1 X, Y...? 0,4
        POINT #2 X, Y...? 3,0

        DISTANCE = 5


        ANOTHER (Y/N)? N

        READY.
```

Cones, Buckets, and Lampshades

Everyone knows what a cone is, of course; you eat ice cream from it, or drink water, or wear it on your head New Year's Eve. But do you know what a frustum is?

Frustum is the geometric term for a cone with the upper end whacked off. It forms a shape like a lampshade or a bucket (hence the name of this program). A frustum is a pretty complicated geometric form, and calculating its dimensions by hand can be an odious (frustumating?) task. But no more: this program does it in a twinkle for both frustums and cones.

Maybe we should explain a few terms. *Height* as used here means the depth of the thing measured vertically, as in "how deep is the bucket?" *Slant height*, which the program figures out, is the height of the outside surface measuring from bottom to top at the slant of the side. *Lateral surface* means the area of the side if you cut the lampshade and spread it flat, and excludes the top and bottom areas, if any. Total surface *does* include the ends, as in the number of square inches of metal in the sides, bottom, and lid of a bucket.

This program is geared toward frustums, but you can use it for cones and get valid results if you tell the program that one of the diameters is 0.

```
NEW
100 REM ** CONES, BUCKETS AND LAMPSHADES
110 PRINT CHR$(147)
120 PRINT "CONICAL THINGS": PRINT
130 INPUT "HEIGHT.............. "; H
140 INPUT "BOTTOM DIAMETER... "; D1
150 INPUT "TOP DIAMETER...... "; D2
160 R1 = D1 / 2: R2 = D2 / 2
170 SH = SQR(H^2 + (R1 - R1)^2)
180 S  = π * SH * (R1 + R2)
190 B1 = π * R1^2
200 B2 = π * R2^2
210 A  = S + B1 + B2
220 V = ((π*H)/3)*((R1^2+R2^2)+(R1*R2))
230 PRINT
240 PRINT "SLANT HEIGHT"     TAB(25) SH
```

```
250 PRINT "LATERAL SURFACE" TAB(25) S
260 PRINT "BOTTOM SURFACE"  TAB(25) B1
270 PRINT "TOP SURFACE"     TAB(25) B2
280 PRINT "TOTAL SURFACE"   TAB(25) A
290 PRINT "VOLUME"          TAB(25) V
300 END
```

```
Sample run:

        CONICAL THINGS

        HEIGHT..... ? 12
        BOTTOM DIAMETER... ? 12
        TOP DIAMETER..... ? 7

        SLANT HEIGHT              12.2576507
        LATERAL SURFACE           365.83118
        BOTTOM SURFACE            113.097336
        TOP SURFACE               38.48451
        TOTAL SURFACE             517.413026
        VOLUME                    870.221165

        READY.
```

Spheres

This program takes the *diameter of a sphere* (defined as a straight line passing from one point on the surface to another through the center) and calculates the circumference, surface area, and volume.

```
100 REM  ** SPHERE
110 PRINT CHR$(147)
120 PRINT "SPHERE CALCULATIONS"
130 PRINT: INPUT "DIAMETER... "; D
140 C = π * D
150 A = D * C
160 V = (π * (D^3)) / 6
170 PRINT
```

(continued)

```
180 PRINT "CIRCUMFERENCE" TAB(25) C
190 PRINT "SURFACE AREA"  TAB(25) A
200 PRINT "VOLUME"        TAB(25) V
210 END
```

```
Sample run:

        SPHERE CALCULATIONS

        DIAMETER... ? 10

        CIRCUMFERENCE              31.4159265
        SURFACE AREA               314.159265
        VOLUME                     523.598776

        READY.
```

Cylinders

Almost anything you'd ever need to know about a cylinder can be determined from its diameter and height, which is what this program asks for. Use it to figure out the dimensions of tin cans, the pistons in your car, bulk petroleum storage tanks, and anything else of a cylindrical shape.

```
100 REM  ** CYLINDER
110 PRINT CHR$(147)
120 PRINT "CYLINDER CALCULATIONS"
130 PRINT: INPUT "DIAMETER... "; D
140 PRINT: INPUT "HEIGHT..... "; H
150 C = π * D: R = D / 2
160 S = C * H
170 A = π * D * (H + R)
180 B = (A - S) / 2
190 V = π * H * R^2
200 PRINT "CIRCUMFERENCE"   TAB(25) C
210 PRINT "LATERAL SURFACE" TAB(25) S
220 PRINT "TOTAL SURFACE"   TAB(25) A
230 PRINT "BASE AREA"       TAB(25) B
```

```
240 PRINT "VOLUME"              TAB(25) V
250 END

Sample run:

        CYLINDER CALCULATIONS

        DIAMETER... ? 3

        HEIGHT..... ? 10

        CIRCUMFERENCE                9.42477796
        LATERAL SURFACE              94.2477796
        TOTAL SURFACE                108.384947
        BASE AREA                    7.06858348
        VOLUME                       70.6858347

        READY.
```

Statistics #1

This program takes any number of numeric values and calculates their range, mean, variance, and standard deviation. It is a simple statistical program that does not weight the results for frequency distribution by groupings (see Statistics #2 for that). You can enter the values in any order. The last value has to be 0, to tell the program that the list is completed.

```
NEW
100 REM  ** STATISTICS #1
110 L = 99999999
120 PRINT CHR$(147)
130 PRINT "STATISTICS #1:": PRINT
140 PRINT "  ENTER VALUES ONE AT A TIME"
150 PRINT "  END LIST WITH 0"
160 REM  ** INPUT AND SUMMARIZE
170 INPUT "VALUE"; Q
180 IF Q = 0 THEN 250
```

(continued)

```
190 IF Q < L THEN L = Q
200 IF Q > H THEN H = Q
210 N = N + 1          :REM SAMPLE SIZE
220 T = T + Q          :REM TOTAL
230 D = D + Q^2        :REM SUM SQUARES
240 GOTO 170
250 REM  ** CALCULATIONS
260 M = T / N          :REM MEAN
270 V = (D / N) - M^2:REM VARIANCE
280 S = SQR(V)         :REM STD DEVIATION
290 REM  ** OUTPUT
300 PRINT "--------------------": PRINT
310 PRINT "SAMPLE SIZE          " N
320 PRINT "RANGE                " H - L
330 PRINT "   LOWEST      " L
340 PRINT "   HIGHEST     " H
350 PRINT "MEAN                 " M
360 PRINT "VARIANCE             " V
370 PRINT "STANDARD DEVIATION   " S
380 PRINT: END
RUN
```

Sample run (earnings per share of 7 stocks):

```
        STATISTICS #1:

          ENTER VALUES ONE AT A TIME
          END LIST WITH 0
        VALUE ? 2.34
        VALUE ? 4.86
        VALUE ? 5.51
        VALUE ? 3.03
        VALUE ? 9.44
        VALUE ? .39
        VALUE ? 2.99
        VALUE ? 0
        --------------------

        SAMPLE SIZE           7
        RANGE:                9.05
```

```
   LOWEST          .39
   HIGHEST         9.44
MEAN                        4.08
VARIANCE                    7.18817142
STANDARD DEVIATION          2.68107654
```

Statistics #2

This program has more sophistication than Statistics #1, because it can take grouped data and apply weights for the number of observations per class. The sample run here categorizes fleet cars by miles per gallon (mi/gal), with the first number the start of the range, the second the upper limit of the range, and the third the number of cars that get that mileage (for example, there are 13 cars that get between 19 and 19.9 mi/gal).

```
NEW
100 REM  ** STATISTICS #2
110 PRINT CHR$(147)
120 PRINT "STATISTICS #2:"
130 PRINT "  ENTER CLASS LOW, HIGH, OBSE
RVATIONS"
140 PRINT "  END LIST WITH THREE 0'S"
150 REM  ** INPUT AND SUMMARIZE
160 INPUT CL, CH, F
170 IF CL = 0 AND F = 0 THEN 240
190 N  = N + F
200 X  = CH - ((CH - CL) / 2)
210 FX = FX + (F * X)
220 F2 = F2 + (F * X^2)
230 GOTO 160
240 REM  ** CALCULATIONS
250 M = FX / N
260 V = (F2 - (N * M^2)) / N
270 S = SQR(V)
280 REM  ** OUTPUT
290 PRINT "--------------------": PRINT
300 PRINT "SAMPLE SIZE          " N
310 PRINT "MEAN                 " M
320 PRINT "VARIANCE             " V
```

(continued)

```
330 PRINT "STANDARD DEVIATION  " S
340 END
```

Sample run (miles per gallon of a group of fleet cars):

```
STATISTICS #2:
  ENTER CLASS LOW, HIGH, OBSERVATIONS
  END LIST WITH THREE 0'S
? 14, 14.9, 4
? 15, 15.9, 5
? 16, 16.9, 8
? 17, 17.9, 5
? 18, 18.9, 11
? 19, 19.9, 13
? 20, 20.9, 17
? 21, 21.9, 21
? 22, 22.9, 14
? 23, 23.9, 2
? 0,0,0
--------------------

SAMPLE SIZE          100
MEAN                 19.69
VARIANCE             5.36239981
STANDARD DEVIATION   2.3156856

READY.
```

In other words, of the 100 cars sampled, their mean (or average) miles per gallon is 19.69 plus or minus 2.316, although any car might vary by as much as 5.36 mi/gal from the mean.

SECTION 2

WEIGHTS AND MEASURES

General

The next several programs deal with the rather serious and perplexing process of converting among various common units of measurement. The United States is, by a global process of elimination, becoming the only nation that still uses the old English systems of measurements, which are irrational, arbitrary, and nonsensical (even if they are comfortable). More and more often, one sees measurements expressed in metric units, even though most of us still can't visualize how long a kilometer is or how much a kilogram really weighs. That irritates some people, who think there's an evil conspiracy to force the metric system on us, yet those same people have to stop and think for a long time to convert, say, cups into quarts. The object here is not to "sell metric," but to point out that our own system of measurements is essentially unfamiliar to us because it makes no logical sense.

The programs that follow try to overcome the hurdles of going from one unit to another, both within the English system and between it and the metric, and within metric itself, although the latter is usually unnecessary. They classify measurements by types,

and permit you to convert instantly from any unit to any other within the classification.

All these programs follow the same general model, and only the units vary from one to another. We do some slick stuff with the screen in them, so sample runs on paper can't show what you'll see. In the case of the first program, you get a list that says:

```
DISTANCE CONVERSION:
        1 INCHES
        2 FEET
        3 YARDS
        4 MILES
        5 CENTIMETERS
        6 METERS
        7 KILOMETERS
CONVERT TO?
```

and the cursor blinks after the question mark. The way to respond is to pick the number of the unit you want to convert to. Let's say you want to convert 10 kilometers to miles, so you type 4. What makes this tricky is that the number you typed disappears and the name of the unit replaces it, so the last line becomes

```
CONVERT TO MILES
```

The same thing happens with the next line, which asks

```
CONVERT FROM?
```

and leaves the cursor winking after the question mark. You're going from kilometers to miles, so type 7. The line then becomes

```
CONVERT FROM KILOMETERS
```

Finally, the program asks you

```
HOW MANY KILOMETERS?
```

It knows which unit to ask for because of the preceding question. Since you're converting 10 kilometers to miles, type 10. The program instantly replies

```
10 KILOMETERS = 6.21371192 MILES
```

and then it asks you

ANOTHER (Y/N)?

Type N (for No) to end the program, or Y (for Yes) to repeat it with different units or values.

These are handy programs and will become ever handier as the day approaches when metric takes over, so make a point of storing them on tape or disk where you can readily get to them.

Distance Measurements

This program converts among the following units:

1 Inches
2 Feet
3 Yards
4 Miles
5 Centimeters
6 Meters
7 Kilometers

```
NEW
100 REM  ** DISTANCE CONVERSION
110 D = 7
120 DIM C(D,D), N$(D)
130 FOR X = 1 TO D
140 READ X$, C(X,0)
150 NEXT X: RESTORE
160 FOR Y = 1 TO D
170 READ N$(Y), C(0,Y)
180 NEXT Y
190 REM  ** CONVERSION FACTORS GRID
200 FOR X = 1 TO D
210 FOR Y = 1 TO D
220 C(X,Y) = C(0,Y) / C(X,0)
230 NEXT Y
240 NEXT X
250 REM  ** SCREEN SETUP
260 PRINT CHR$(147)
270 PRINT "DISTANCE CONVERSION:"
```

(continued)

```
280 FOR X = 1 TO D
290 PRINT TAB(5); X; N$(X)
300 NEXT X: PRINT
310 INPUT "CONVERT TO"; Y
320 PRINT CHR$(145) "CONVERT TO " N$(Y)
330 PRINT: INPUT "CONVERT FROM"; X
340 PRINT CHR$(145) "CONVERT FROM "N$(X)
350 PRINT: PRINT "HOW MANY " N$(X);
360 INPUT U: PRINT
370 YU = U / C(X,Y)
380 PRINT U; N$(X); " = "; YU; N$(Y)
390 PRINT: PRINT
400 INPUT "ANOTHER (Y/N)"; X$
410 IF X$ = "Y" THEN 250
420 END
430 REM  ** UNITS IN CENTIMETERS
440 DATA "INCHES",       2.54
450 DATA "FEET",         30.48
460 DATA "YARDS",        91.44
470 DATA "MILES",        160934.4
480 DATA "CENTIMETERS", 1
490 DATA "METERS",       100
500 DATA "KILOMETERS",  100000
RUN
```

Area Measurements

This program converts among the following:

- Square inches
- Square feet
- Square yards
- Acres
- Square miles
- Square centimeters
- Square meters
- Hectares
- Square kilometers

To start, get back the last program from tape or disk. You can save some typing by changing lines 100, 110, 140, 170, 270, and the DATA statements from 430 onward. They're the only changes between the two programs. Save the new program under the name AREA MEASUREMENTS.

```
NEW
100 REM  ** AREA CONVERSION
110 D = 9
120 DIM C(D,D), N$(D)
130 FOR X = 1 TO D
140 READ X$, C(X,0): C(X,0) = C(X,0)^2
150 NEXT X: RESTORE
160 FOR Y = 1 TO D
170 READ N$(Y),C(0,Y):C(0,Y) = C(0,Y)^2
180 NEXT Y
190 REM  ** CONVERSION FACTORS GRID
200 FOR X = 1 TO D
210 FOR Y = 1 TO D
220 C(X,Y) = C(0,Y) / C(X,0)
230 NEXT Y
240 NEXT X
250 REM  ** SCREEN SETUP
260 PRINT CHR$(147)
270 PRINT "AREA CONVERSION:"
280 FOR X = 1 TO D
290 PRINT TAB(5); X; N$(X)
300 NEXT X: PRINT
310 INPUT "CONVERT TO"; Y
320 PRINT CHR$(145) "CONVERT TO " N$(Y)
330 PRINT: INPUT "CONVERT FROM"; X
340 PRINT CHR$(145) "CONVERT FROM "N$(X)
350 PRINT: PRINT "HOW MANY " N$(X);
360 INPUT U: PRINT
370 YU = U / C(X,Y)
380 PRINT U; N$(X); " = "; YU; N$(Y)
390 PRINT: PRINT
400 INPUT "ANOTHER (Y/N)"; X$
410 IF X$ = "Y" THEN 250
```

(continued)

```
420 END
430 REM  ** UNITS IN CENTIMETERS
440 DATA "SQ INCHES",       2.54
450 DATA "SQ FEET",         30.48
460 DATA "SQ YARDS",        91.44
470 DATA "ACRES",           6361.4907
480 DATA "SQ MILES",        160934.4
490 DATA "SQ CENTIMETERS", 1
500 DATA "SQ METERS",       100
510 DATA "HECTARES",        10000
520 DATA "SQ KILOMETERS",   100000
RUN
```

Weight Conversions

Use this program to convert weights among:

- Ounces
- Pounds
- Stone
- Tons
- Long tons
- Grams
- Kilograms
- Metric tons

As in the preceding program, you can save some trouble by reloading the distance program and changing lines 100, 110, 270, and the DATA statements from 430 onward.

```
NEW
100 REM  ** WEIGHT CONVERSION
110 D = 8
120 DIM C(D,D), N$(D)
130 FOR X = 1 TO D
140 READ X$, C(X,0)
150 NEXT X: RESTORE
160 FOR Y = 1 TO D
170 READ N$(Y), C(0,Y)
```

```
180 NEXT Y
190 REM  ** CONVERSION FACTORS GRID
200 FOR X = 1 TO D
210 FOR Y = 1 TO D
220 C(X,Y) = C(0,Y) / C(X,0)
230 NEXT Y
240 NEXT X
250 REM  ** SCREEN SETUP
260 PRINT CHR$(147)
270 PRINT "WEIGHT CONVERSION:"
280 FOR X = 1 TO D
290 PRINT TAB(5); X; N$(X)
300 NEXT X: PRINT
310 INPUT "CONVERT TO"; Y
320 PRINT CHR$(145) "CONVERT TO " N$(Y)
330 PRINT: INPUT "CONVERT FROM"; X
340 PRINT CHR$(145) "CONVERT FROM "N$(X)
350 PRINT: PRINT "HOW MANY " N$(X);
360 INPUT U: PRINT
370 YU = U / C(X,Y)
380 PRINT U; N$(X); " = "; YU; N$(Y)
390 PRINT: PRINT
400 INPUT "ANOTHER (Y/N)"; X$
410 IF X$ = "Y" THEN 250
420 END
430 REM  ** UNITS IN GRAMS
440 DATA "OUNCES",       28.3495
450 DATA "POUNDS",       453.592
460 DATA "STONE",        6350.288
470 DATA "TONS",         907184
480 DATA "LONG TONS",    1016046.1
490 DATA "GRAMS",        1
500 DATA "KILOGRAMS",    1000
510 DATA "METRIC TONS", 1000000
RUN
```

Liquid Measurements

This is a really easy program to set up. Just load the distance program into memory and change lines 100, 270, and the DATA statements. You'll then have a program that converts among:

- Cups
- Pints
- Quarts (U.S.)
- Imperial quarts
- Gallons (U.S.)
- Imperial gallons
- Liters

```
NEW
100 REM  ** LIQUID MEASUREMENTS
110 D = 7
120 DIM C(D,D), N$(D)
130 FOR X = 1 TO D
140 READ X$, C(X,0)
150 NEXT X: RESTORE
160 FOR Y = 1 TO D
170 READ N$(Y), C(0,Y)
180 NEXT Y
190 REM  ** CONVERSION FACTORS GRID
200 FOR X = 1 TO D
210 FOR Y = 1 TO D
220 C(X,Y) = C(0,Y) / C(X,0)
230 NEXT Y
240 NEXT X
250 REM  ** SCREEN SETUP
260 PRINT CHR$(147)
270 PRINT "LIQUID MEASUREMENTS:"
280 FOR X = 1 TO D
290 PRINT TAB(5); X; N$(X)
300 NEXT X: PRINT
310 INPUT "CONVERT TO"; Y
320 PRINT CHR$(145) "CONVERT TO " N$(Y)
330 PRINT: INPUT "CONVERT FROM"; X
340 PRINT CHR$(145) "CONVERT FROM "N$(X)
```

```
350 PRINT: PRINT "HOW MANY " N$(X);
360 INPUT U: PRINT
370 YU = U / C(X,Y)
380 PRINT U; N$(X); " = "; YU; N$(Y)
390 PRINT: PRINT
400 INPUT "ANOTHER (Y/N)"; X$
410 IF X$ = "Y" THEN 250
420 END
430 REM  ** UNITS
440 DATA "CUPS",                .236575
450 DATA "PINTS",               .47315
460 DATA "QUARTS",              .9463
470 DATA "IMPERIAL QUARTS",     1.1365
480 DATA "GALLONS",             3.7852
490 DATA "IMPERIAL GALLONS",    4.546
500 DATA "LITERS",              1
RUN
```

Recipe Changer

It's an age-old problem of cooks: The recipe serves 5 and you have 8 coming to dinner. How much should you increase such measurements as 1 ⅓ cups to keep the right proportions? This program saves the day (and a headache) because it increases or decreases any of the common kitchen measures in proportion to the number the recipe serves versus how many you plan to serve.

This is an unusual program in that it works with fractions, consistent with the way recipes are written. The fractions it calculates are approximate, rounded to the nearest common kitchen measure (nothing like 11/27, which no cup is calibrated to measure). This shouldn't be a problem, since no cook I know is *that* concerned about precision.

The program knows the proportions based on the first two questions, which establish the size of the recipe and the size of your dinner party. It doesn't care about cups or tablespoons or pounds, just numbers. When you enter the measurement for, say, cups of flour, it tells you how many to use based on the number you're serving. When you've entered all the measurements and want the program to stop, type 0.

```
NEW
100 REM  ** RECIPE CHANGER
110 PRINT CHR$(147)
120 PRINT "RECIPE CHANGER:": PRINT
130 INPUT "NUMBER RECIPE SERVES "; RQ
140 INPUT "NUMBER YOU'RE SERVING"; SQ
150 CF = SQ / RQ
160 REM  ** MAIN LOOP
170 PRINT: INPUT "QUANTITY"; Q$
180 IF Q$ = "0" THEN END
190 L = LEN(Q$): X = 1: SL = 1: Q = 0
200 FOR P = 1 TO L
210 IF MID$(Q$, P, 1) = "/" THEN 240
220 NEXT P
230 Q = VAL(Q$): GOTO 360
240 FOR P = 1 TO L
250 IF MID$(Q$,P,1) = " " THEN SL = P: G
OTO 280
260 NEXT P
270 GOTO 290
280 Q = VAL(LEFT$(Q$, P)): X = P + 1
290 FOR P = X TO L
300 IF MID$(Q$, P, 1) = "/" THEN 330
310 NEXT P
320 PRINT "BAD INPUT": GOTO 160
330 N = VAL(MID$(Q$, SL, P - SL))
340 D = VAL(RIGHT$(Q$, L - P))
350 F = N / D: Q = Q + F
360 REM  ** FIGURE QUANTITY
370 Q = Q * CF: W = INT(Q)
380 F = Q - W: F$ = ""
390 IF F >= .95 THEN W = W + 1
400 PRINT "CHANGE TO" W;
410 REM  ** FRACTION
420 IF F < .06 THEN 510
430 IF F < .95 THEN F$ = " 7/8"
440 IF F < .82 THEN F$ = " 3/4"
450 IF F < .7  THEN F$ = " 2/3"
460 IF F < .6  THEN F$ = " 1/2"
```

```
470 IF F < .4  THEN F$ = " 1/3"
480 IF F < .3  THEN F$ = " 1/4"
490 IF F < .18 THEN F$ = " 1/8"
500 PRINT F$
510 PRINT: PRINT: GOTO 160
RUN

Sample run:

        RECIPE CHANGER:

        NUMBER RECIPE SERVES ? 5
        NUMBER YOU'RE SERVING? 8

        QUANTITY? 1 1/3
        CHANGE TO 2  1/8

        QUANTITY? 3/4
        CHANGE TO 1  1/4

        QUANTITY? 2 1/2
        CHANGE TO 4

        QUANTITY? 0

        READY.
```

Temperature Conversion

Back in the olden days they called it Centigrade, but nowadays it's Celsius. No doubt someone knows why. At any rate, it's the metric way of measuring temperatures, in which water freezes at 0° and boils at 100°, rather than at the capricious Fahrenheit marks of 32° and 212°. This program converts from one system to the other, with a menu that lets you pick which converstion you want to make. Using it, you'll know whether to be hot or cold when the temperature is 40°C.

```
NEW
100 REM  ** TEMPERATURE CONVERSION
110 DEF FNC(X) = (5/9) * (X - 32)
120 DEF FNF(X) = ((9/5) * X) + 32
130 DEF FNR(X) = INT((X * 10) + .5) / 10
140 REM  ** CONVERSION LOOP
150 PRINT CHR$(147)
160 PRINT "TEMPERATURE CONVERSION:"
170 PRINT
180 PRINT "  1  CELSIUS TO FAHRENHEIT"
190 PRINT "  2  FAHRENHEIT TO CELSIUS"
200 PRINT: INPUT "       WHICH"; X
210 PRINT: INPUT "       TEMPERATURE"; T
220 PRINT: PRINT
230 IF X = 2 THEN 280
240 REM  ** CEL TO FAHR
250 PRINT T "DEGREES C =";
260 PRINT FNR(FNF(T)) "DEGREES F"
270 GOTO 310
280 REM  ** FAHR - CEL
290 PRINT T "DEGREES F =";
300 PRINT FNR(FNC(T)) "DEGREES C"
310 PRINT: INPUT "ANOTHER (Y/N)"; X$
320 IF X$ = "Y" THEN 140
330 END
RUN
```

Sample run:

```
TEMPERATURE CONVERSION:

  1  CELSIUS TO FAHRENHEIT
  2  FAHRENHEIT TO CELSIUS

     WHICH? 1

     TEMPERATURE? 40
```

(continued)

```
40 DEGREES C = 104 DEGREES F

ANOTHER (Y/N)? N

READY.
```

SECTION 3

SORTING AND TEXT PROCESSING

Sorting a Group of Numbers

Sorting is the process of rearranging unorganized information into an orderly sequence. This program takes numbers and puts them into ascending order (lowest to highest in value), with the results displayed on the screen in a left-to-right fashion. The display holds up to 88 numbers before lines begin to scroll off the top, so 88 is the limit of the sort size. When you enter 0, the program stops accepting numbers and begins sorting.

```
NEW
100 REM  ** NUMERIC SORT
110 PRINT CHR$(147)
120 PRINT "NUMERIC SORT": PRINT
130 PRINT "ENTER NUMBERS ONE AT A TIME"
140 PRINT "ENTER 0 TO START SORT"
150 DIM A(87)
160 N = 0
170 REM  ** INPUT LOOP
```

```
180 INPUT A(N)
190 IF A(N) = 0 THEN 220
200 N = N + 1
210 IF N <= 87 THEN 180
220 REM  **  SORT
230 L = 999999: M = 0
240 FOR P = 0 TO (N - 1)
250 IF A(P) < L THEN L = A(P): M = P
260 NEXT P
270 IF L = 999999 THEN END
280 PRINT L,
290 A(M) = 999999
300 GOTO 220
RUN
```

Sample run:

```
        NUMERIC SORT

        ENTER NUMBERS ONE AT A TIME
        ENTER 0 TO START SORT
        ? 91
        ? 73
        ? 55
        ? 37
        ? 19
        ? 28
        ? 46
        ? 64
        ? 82
        ? 0
         19          28          37          46
         55          64          73          82
         91
        READY.
```

Descending Sort

Sometimes it's useful to sort numbers into descending order (highest to lowest in value), which produces a result that is exactly opposite that of the preceding program. This program works the same in all other respects.

```
NEW
100 REM  ** DESCENDING SORT
110 PRINT CHR$(147)
120 PRINT "DESCENDING SORT": PRINT
130 PRINT "ENTER NUMBERS ONE AT A TIME"
140 PRINT "ENTER 0 TO START SORT"
150 DIM A(87)
160 N = 0
170 REM  ** INPUT LOOP
180 INPUT A(N)
190 IF A(N) = 0 THEN 220
200 N = N + 1
210 IF N <= 87 THEN 180
220 REM  ** SORT
230 L = -1: M = 0
240 FOR P = 0 TO (N - 1)
250 IF A(P) > L THEN L = A(P): M = P
260 NEXT P
270 IF L = -1 THEN END
280 PRINT L,
290 A(M) = -1
300 GOTO 220
RUN

Sample run:


        NUMERIC SORT

        ENTER NUMBERS ONE AT A TIME
        ENTER 0 TO START SORT
        ? 91
```

```
? 73
? 55
? 37
? 19
? 28
? 46
? 64
? 82
? 0
 91          82          73          64
 55          46          37          28
 19
READY.
```

Text Analysis

Although the word "computer" suggests (and the majority of usages confirms) a machine devoted exclusively to doing calculations, computers can also process and analyze text in various ways. This program demonstrates simple text analysis by taking any line you type and telling you the number of words it contains, the length of the line, and the number of occurrences of each character. The only character you cannot use is a comma, since the INPUT instruction thinks a comma separates two different data items entered on the same line. The counts of individual characters do not include SPACEs, although the number of SPACEs *is* included in the total. Also, if you have two or more SPACEs together, they will inflate the word count; the program assumes one SPACE between each word and no SPACE at the start of the text.

```
NEW
100 REM  ** TEXT ANALYSIS
110 PRINT CHR$(147)
120 DIM CH(255)
130 REM  ** GET AND ANALYZE TEXT
140 PRINT "ENTER A LINE OF TEXT"
150 INPUT TX$: PRINT
```

(continued)

```
160 LT = LEN(TX$)
170 FOR P = 1 TO LT
180 V = ASC(MID$(TX$, P, 1))
190 CH(V) = CH(V) + 1
200 NEXT P
210 REM  ** DISPLAY RESULTS
220 NW = CH(32) + 1
230 PRINT "CONTAINS" LT "CHARACTERS ";
240 PRINT "IN" NW "WORDS"
250 FOR P = 0 TO 255
260 IF P = 32 THEN 290
270 IF CH(P) = 0 THEN 290
280 PRINT TAB(8) CHR$(P) CH(P)
290 NEXT P
300 END
RUN
```

Sample run:

```
        ENTER A LINE OF TEXT
        ? THIS LINE IS GOING TO BE ANALYZED

        CONTAINS 33 CHARACTERS IN 7 WORDS
                A 2
                B 1
                D 1
                E 3
                G 2
                H 1
                I 4
                L 2
                N 3
                O 2
                S 2
                T 2
                Y 1
                Z 1

        READY.
```

Reversing Text

To see another example of text processing, type a line and the Commodore 64 immediately displays it backwards. It repeats as many times as you want. Stop it by entering the number 0.

```
NEW
100 REM  ** TEXT REVERSER
110 PRINT CHR$(147)
120 PRINT: PRINT "ENTER A LINE OF TEXT"
130 INPUT X$
140 IF X$ = "0" THEN END
150 PRINT "  ";
160 FOR P = LEN(X$) TO 1 STEP -1
170 PRINT MID$(X$, P, 1);
180 NEXT P
190 PRINT
200 GOTO 120
RUN

Sample run:

        ENTER A LINE OF TEXT
        ? THIS LINE WILL APPEAR BACKWARDS
          SDRAWKCAB RAEPPA LLIW ENIL SIHT

        ENTER A LINE OF TEXT
        ? 0

        READY.
```

Alphabetizing

The process of alphabetizing a list differs little from sorting a group of numbers. The computer compares entries in the list and selects the one whose ASCII value is the lowest. In this particular program, a selected entry is printed and then the program changes

that entry to the highest possible ASCII value so it won't be selected again. You can alphabetize up to 25 entries with this program. During the input phase, a 0 signals the end of the list, and the program then sorts and displays the list in alphabetic order.

```
NEW
100 REM  ** ALPHABETIZE
110 PRINT CHR$(147): X$ = CHR$(255)
120 PRINT "ALPHABETIZE A LIST"
130 PRINT " ENTER ITEMS ONE AT A TIME"
140 PRINT " ENTER 0 TO END LIST"
150 DIM L$(25)
160 N = 0
170 REM  ** INPUT LOOP
180 INPUT L$(N)
190 IF L$(N) = "0" THEN PRINT: GOTO 220
200 N = N + 1
210 IF N <= 25 THEN 170
220 REM  ** ALPHABETIZE
230 C$ = X$: M = 0
240 FOR P = 0 TO (N - 1)
250 IF L$(P) >= C$ THEN 270
260 C$ = L$(P): M = P
270 NEXT P
280 IF C$ = X$ THEN END
290 PRINT C$
300 L$(M) = X$
310 GOTO 220
RUN

Sample run:

        ALPHABETIZE A LIST
         ENTER ITEMS ONE AT A TIME
         ENTER 0 TO END LIST
        ? JOHN
        ? MARIANNE
        ? BETH
        ? MARY
```

```
? LUISA
? ZEKE
? ANNETTE
? PETER
? 0

ANNETTE
BETH
JOHN
LUISA
MARIANNE
MARY
PETER
ZEKE

READY.
```

Reverse Alphabetic Order

Just as you might sometimes want to sort numbers into descending order, so might you occasionally need to place a list into reverse alphabetic order (Z to A). This produces exactly the opposite results of the preceding program, but note that the programs themselves are almost identical. The only real instruction changes occur in lines 110 and 250; the others (lines 100 and 120) are cosmetic.

```
NEW
100 REM  ** REVERSE ALPHABETIC ORDER
110 PRINT CHR$(147): X$ = CHR$(0)
120 PRINT "REVERSE ALPHABETIZE A LIST"
130 PRINT " ENTER ITEMS ONE AT A TIME"
140 PRINT " ENTER 0 TO END LIST"
150 DIM L$(25)
160 N = 0
170 REM  ** INPUT LOOP
180 INPUT L$(N)
190 IF L$(N) = "0" THEN PRINT: GOTO 220
```

(continued)

```
200 N = N + 1
210 IF N <= 25 THEN 170
220 REM  ** ALPHABETIZE
230 C$ = X$: M = 0
240 FOR P = 0 TO (N - 1)
250 IF L$(P) <= C$ THEN 270
260 C$ = L$(P): M = P
270 NEXT P
280 IF C$ = X$ THEN END
290 PRINT C$
300 L$(M) = X$
310 GOTO 220
RUN
```

Sample run:

```
        REVERSE ALPHABETIZE A LIST
         ENTER ITEMS ONE AT A TIME
         ENTER 0 TO END LIST
        ? JOHN
        ? MARIANNE
        ? BETH
        ? MARY
        ? LUISA
        ? ZEKE
        ? ANNETTE
        ? PETER
        ? 0

        ZEKE
        PETER
        MARY
        MARIANNE
        LUISA
        JOHN
        BETH
        ANNETTE

        READY.
```

SECTION 4

PROGRAMS HAVING TO DO WITH MONEY

Loan Terms

So you're thinking about borrowing some money, are you? The big question is, how much will it cost you, given a certain interest rate applied to the amount over time? Since lenders usually calculate interest on the declining balance, estimating the monthly payment with any degree of accuracy is no simple task.

This program figures it for you within a penny, giving you the monthly and final payments and the total amount you'll repay the lender. That entails quite a lot of work, so there is a brief delay while the program "thinks." This is a very useful program that lets you play "what if" with differing interest rates, repayment terms, and borrowed amounts. It repeats until you tell it you want to borrow $0.

```
NEW
100 REM  ** LOAN TERMS
110 DEF FNM(X)=INT(((X*MP)*100)+.5)/100
120 PRINT CHR$(147)
130 PRINT "LOAN TERMS": PRINT
```

(continued)

```
140 INPUT "AMOUNT OF LOAN       "; A
150 IF A = 0 THEN END
160 INPUT "ANNUAL INTEREST RATE "; AI
170 INPUT "NUMBER OF PAYMENTS   "; NP
180 REM  ** COMPUTE TERMS
190 IF AI > 1 THEN AI = AI / 100
200 MP = AI / 12
210 F = MP + 1: T = F
220 FOR C = 2 TO NP
230 T = T * F
240 NEXT C
250 PF = 1 / T
260 SMP = A * (MP / (1 - PF))
270 SMP = (INT(SMP * 100)) / 100 + .01
280 CB = A
290 FOR C = 1 TO (NP - 1)
300 CB = INT((CB-SMP+FNM(CB))*100)/100
310 NEXT C
320 FP = CB + FNM(CB)
330 TP = (SMP * (NP - 1)) + FP
340 REM  ** PRINT RESULTS
350 PRINT
360 PRINT NP-1 "PAYMENTS OF" TAB(30) "$"
 SMP
370 PRINT " AND A FINAL PAYMENT OF" TAB(
30) "$" FP
380 PRINT
390 PRINT " FOR A TOTAL OF $" TP
400 PRINT: PRINT: GOTO 140
RUN
```

Sample run:

```
        LOAN TERMS

        AMOUNT OF LOAN       ? 3500
        ANNUAL INTEREST RATE ? 18
        NUMBER OF PAYMENTS   ? 36
```

```
 35 PAYMENTS OF                    $ 126.54
 AND A FINAL PAYMENT OF            $ 126.04

 FOR A TOTAL OF $ 4554.94


AMOUNT OF LOAN          ? 3500
ANNUAL INTEREST RATE    ? 18
NUMBER OF PAYMENTS      ? 30

 29 PAYMENTS OF                    $ 145.74
 AND A FINAL PAYMENT OF            $ 145.55

 FOR A TOTAL OF $ 4372.01


AMOUNT OF LOAN          ? 0

READY.
```

Loan Analysis

This is an extremely useful and powerful program for analyzing a loan based upon three of the following four factors:

- Principal amount borrowed
- Monthly payment (principal and interest)
- Number of payments
- Annual percentage interest rate

The program presents a questionnaire in which you fill in the terms you know and enter 0 for the unknown. It then calculates and gives you the "missing" item. You must answer three of the questions: if you answer all four, the program can't identify the unknown and therefore reports an error and asks for reentry; if you enter more than one unknown, it hasn't enough to go on and a crash will ensue.

The results are estimates and may not be exact, since a certain amount of rounding occurs. The program assumes the interest is applied monthly on the declining balance of principal.

```
NEW
100 REM  ** LOAN ANALYSIS
110 PRINT CHR$(147)
120 Q1$="PRINCIPAL AMOUNT  ": Q2$="MONTH
LY PAYMENT   "
130 Q3$="NUMBER OF PAYMENTS": Q4$="ANNUA
L % INTEREST "
140 GOTO 220
150 REM  ** GET FACTORS
160 PRINT Q1$;: INPUT V: RETURN
170 PRINT Q2$;: INPUT M: RETURN
180 PRINT Q3$;: INPUT N: RETURN
190 PRINT Q4$;: INPUT I
200 IF I > 1 THEN I = I / 100
210 I = I / 12: RETURN
220 REM  ** BEGIN HERE
230 PRINT "LOAN ANALYSIS:": PRINT
240 PRINT "ENTER 0 FOR THE UNKNOWN"
250 PRINT: GOSUB 160: IF V = 0 THEN 300
260 GOSUB 170: IF M = 0 THEN 360
270 GOSUB 180: IF N = 0 THEN 420
280 GOSUB 190: IF I = 0 THEN 480
290 PRINT: PRINT "  ERROR!": GOTO 240
300 REM  ** CALCULATE PRINCIPAL
310 GOSUB 170: GOSUB 180: GOSUB 190
320 V = M * (1 - (1 / (1 + I)^N)) / I
330 V = INT(V * 100) / 100
340 PRINT: PRINT Q1$ "$" V
350 GOTO 600
360 REM  ** CALCULATE PAYMENT
370 GOSUB 180: GOSUB 190
380 M = V * (I / (1 - (1 + I)^(-N)))
390 M = INT(M * 100) / 100
400 PRINT: PRINT Q2$ "$" M
410 GOTO 600
420 REM  ** CALCULATE # PAYMENTS
430 GOSUB 190
440 N = LOG(1/(1-I*V/M)) / LOG(1+I)
450 N = INT(N + .4999)
460 PRINT: PRINT Q3$ N
470 GOTO 600
480 REM  ** CALCULATE % INT RATE
490 PRINT "THINKING"
500 I1 = 2 * (N - V / M) / (N * (N + 1))
510 V1 = M * (1 - (1 + I1)^(-N)) / I1
520 IF V1 > (1.0000001 * V) THEN 570
530 IF V1 < (0.9999999 * V) THEN 570
540 I = I1 * 1200
550 PRINT: PRINT Q4$ I
560 GOTO 600
570 Y = (1 + I1)^(-N):  W = 1 - Y
580 I1 = I1 * (1-(((I1*V/M)-W) / (W-(N*I
1*Y / (1+I1)))))
```

```
590 GOTO 510
600 REM  ** QUIT OR REPEAT
610 PRINT: INPUT "ANOTHER RUN (Y/N)"; R$
620 IF R$="Y" THEN PRINT:PRINT:GOTO 220
630 END
RUN
```

Sample run: In the first case, the monthly payment is unknown. In the second run, the terms are the same but the annual percent interest rate is indicated as unknown. The "thinking" message appears because this computation causes a delay.

```
LOAN ANALYSIS:

ENTER 0 FOR THE UNKNOWN

PRINCIPAL AMOUNT  ? 10000
MONTHLY PAYMENT   ? 0
NUMBER OF PAYMENTS? 120
ANNUAL % INTEREST ? 10.05

MONTHLY PAYMENT   $ 132.42

ANOTHER RUN (Y/N)? Y


LOAN ANALYSIS:

ENTER 0 FOR THE UNKNOWN

PRINCIPAL AMOUNT  ? 10000
MONTHLY PAYMENT   ? 132.42
NUMBER OF PAYMENTS? 120
ANNUAL % INTEREST ? 0
THINKING

ANNUAL % INTEREST  10.0485992

ANOTHER RUN (Y/N)? N

READY.
```

Interest Paid Over a Period

Here's a lifesaver during income tax time, when you have to itemize interest deductions for the year on your home mortgage, car loan, and so on. This program figures up how much interest you paid over a period bracketed by two payments: say in January you made payment #19 and in December, payment 30. If you know the annual interest rate, the monthly payment, and the total number of payments over the life of the loan, the program tells you the interest cost for that period. As a fringe benefit, it also gives you the starting and ending balances for the period.

```
NEW
100 REM  ** INTEREST PAID
110 PRINT CHR$(147)
120 PRINT"INTEREST OVER A PERIOD:":PRINT
130 PRINT "MONTHLY PAYMENT" TAB(28);
140 INPUT P
150 IF P = 0 THEN END
160 PRINT"ANNUAL INTEREST RATE" TAB(28);
170 INPUT J
180 IF J > 1 THEN J = J / 100
190 I = J / 12
200 PRINT "TOTAL PAYMENTS IN LOAN";
210 PRINT TAB(28);: INPUT T
220 PRINT"# OF FIRST PMT IN PERIOD";
230 PRINT TAB(28);: INPUT P1: P1=P1-1
240 PRINT "# OF LAST PMT IN PERIOD";
250 PRINT TAB(28);: INPUT P2: PRINT
260 DEF FNI(X) = (1 + I)^X
270 DEF FNR(X) = INT(X * 100) / 100
280 IP = P * (P2 - P1 - (FNI(P2-T)/I) +
FNI(P1-T)/I)
290 B1 = (P / I) * (1 - FNI(P1 - T))
300 B2 = (P / I) * (1 - FNI(P2 - T))
310 PRINT "INTEREST OVER PERIOD: $";
320 PRINT FNR(IP): PRINT
330 PRINT "STARTING BALANCE:     $";
340 PRINT FNR(B1)
350 PRINT "ENDING BALANCE:       $";
```

```
360 PRINT FNR(B2)
370 PRINT: PRINT: GOTO 130
RUN


Sample run:

        INTEREST OVER A PERIOD:

        MONTHLY PAYMENT              ? 285.36
        ANNUAL INTEREST RATE         ? 8.5
        TOTAL PAYMENTS IN LOAN       ? 360
        # OF FIRST PMT IN PERIOD     ? 168
        # OF LAST PMT IN PERIOD      ? 179

        INTEREST OVER PERIOD: $ 2512.43

        STARTING BALANCE:       $ 29969.63
        ENDING BALANCE:         $ 29057.75


        MONTHLY PAYMENT:             ? 0

        READY.
```

Present Worth of a Future Amount

This and the following three programs deal with various aspects of the time value of money. If you have a certain amount of money today and interest applies to it, it will increase in value with time; conversely, if you want to have a certain amount of money several years from now, you can use the time value to calculate how much that amount is worth today. All of these programs assume continuous compounding, which most banks and financial institutions use. Inflation and taxes are not considered.

The present worth of a future amount, though self-explanatory, is a stuffy economists' term. What it means is: "If you want to have a certain amount of money after so many years, how much do you

have to put in the bank, given a fixed interest rate?" This program, then, tells you the lump sum to invest today in order to meet an objective several years from now.

```
NEW
100 REM  ** PRESENT WORTH OF FUTURE AMT
110 PRINT CHR$(147)
120 PRINT "PRESENT WORTH:": PRINT
130 INPUT "  FUTURE AMOUNT     "; F
140 IF F = 0 THEN END
150 INPUT "  INTEREST RATE     "; I
160 INPUT "  NUMBER OF YEARS   "; N
170 IF I > 1 THEN I = I / 100
180 J = LOG(1 + I) / LOG(2.71828)
190 PF = 1 / ((1 + I)^N)
200 PW = F * PF
210 REM  ** DISPLAY RESULTS
220 PW = INT(PW * 100) / 100
230 PRINT: PRINT
240 PRINT "PRESENT WORTH IS $"; PW
250 PRINT: PRINT "----------------"
260 PRINT: GOTO 130
RUN
```

Sample run:

```
        PRESENT WORTH:

          FUTURE AMOUNT     ? 20000
          INTEREST RATE     ? 10.5
          NUMBER OF YEARS   ? 10

        PRESENT WORTH IS $ 7368.97

        ----------------

          FUTURE AMOUNT     ? 20000
          INTEREST RATE     ? 11.5
          NUMBER OF YEARS   ? 10
```

```
PRESENT WORTH IS $ 6734.12

----------------

  FUTURE AMOUNT      ? 0

READY.
```

This sample run means that if you want $20,000 ten years from now, you have to invest $7368.97 at 10.5 percent interest or $6734.12 at 11.5 percent. Obviously, the amount of "up-front" money varies greatly with the interest rate.

Future Worth of a Present Amount

This is the opposite of the preceding program, but the same assumptions apply concerning continuous compounding. Future worth answers such questions as, "If I make a one-time deposit of a certain amount in a savings account, how much will it be worth after some number of years, given a fixed interest rate?"

```
NEW
100 REM  ** FUTURE WORTH OF PRESENT AMT
110 PRINT CHR$(147)
120 PRINT "FUTURE WORTH:": PRINT
130 INPUT "  PRESENT AMOUNT     "; P
140 IF P = 0 THEN END
150 INPUT "  INTEREST RATE      "; I
160 INPUT "  NUMBER OF YEARS    "; N
170 IF I > 1 THEN I = I / 100
180 J = LOG(1 + I) / LOG(2.71828)
190 FF = (1 + I)^N
200 FW = P * FF
210 REM  ** DISPLAY RESULTS
220 FW = INT(FW * 100) / 100
230 PRINT: PRINT
240 PRINT "FUTURE WORTH IS $"; FW
```

(continued)

```
250 PRINT: PRINT "----------------"
260 PRINT: GOTO 130
RUN

Sample run:

        FUTURE WORTH:

          PRESENT AMOUNT     ? 5000
          INTEREST RATE      ? 9.65
          NUMBER OF YEARS    ? 15

        FUTURE WORTH IS $ 19911.29

        ----------------

          PRESENT AMOUNT     ? 5000
          INTEREST RATE      ? 10.5
          NUMBER OF YEARS    ? 15

        FUTURE WORTH IS $ 22356.51

        ----------------

          PRESENT AMOUNT      ? 0

        READY.
```

This display means that if you have $5000 and you invest it at 9.65 percent interest, after 15 years it will be worth $19,911.29, and if you're fortunate enough to earn 10.5 percent on it, it will be worth $22,356.51 after 15 years. A small variation in the interest rate, then, can greatly influence the time value of money.

Saving Toward a Future Amount

Let's suppose you're a parent and you want to set up a savings plan to put the kid through college. Your objective is to have $30,000 fifteen years from now. How much do you have to save per month, given a fixed interest rate and continuous compounding,

to reach this goal? (Or let's say you're not a parent, but you want to take a fabulous trip around the world.) This program helps you establish a savings plan toward a specific amount.

```
NEW
100 REM  ** MONTHLY SAVINGS PLAN
110 PRINT CHR$(147)
120 PRINT "SAVING PLAN:": PRINT
130 INPUT "  YOUR GOAL          "; F
140 IF F = 0 THEN END
150 INPUT "  INTEREST RATE      "; I
160 INPUT "  NUMBER OF YEARS    "; N
170 IF I > 1 THEN I = I / 100
180 J = LOG(1 + I) / LOG(2.71828)
190 AF = J / (((1 + I)^N) - 1)
200 M  = (F * AF) / 12
210 REM  ** DISPLAY RESULTS
220 M  = INT(M * 100) / 100
230 PRINT: PRINT
240 PRINT "AMOUNT TO SAVE MONTHLY IS $"M
250 PRINT: PRINT "----------------"
260 PRINT: GOTO 130
RUN

Sample run:

        SAVINGS PLAN:

          YOUR GOAL          ? 30000
          INTEREST RATE      ? 8.65
          NUMBER OF YEARS    ? 15

        AMOUNT TO SAVE MONTHLY IS $ 83.93

        ----------------

          YOUR GOAL          ? 30000
          INTEREST RATE      ? 9.6
          NUMBER OF YEARS    ? 15
```

(continued)

```
AMOUNT TO SAVE MONTHLY IS $ 77.54

----------------

  YOUR GOAL          ? 0

READY.
```

Annuity From a Present Amount

This is somewhat different from the preceding programs having to do with the time value of money. They dealt with building toward the attainment of a measure of financial independence, whereas this problem assumes you've reached that enviable plateau and now you want to spend it.

Let's say you've come into a $100,000 inheritance and you have decided to live it up. You want it to last five years, at the end of which you won't have a penny left. How much can you spend per month, given compounding interest on the balance?

This program, of course, also pertains to establishing a pension, the idea being to set aside a certain amount that has to last for a specified period of time. In other words, this program answers the question "If I put $P in the bank at 1 percent, how much can I take out each month to have it last N years?"

```
NEW
100 REM  ** ANNUITY FROM A PRESENT AMT
110 PRINT CHR$(147)
120 PRINT "ANNUITY:": PRINT
130 INPUT "  INITIAL AMOUNT     "; P
140 IF P = 0 THEN END
150 INPUT "  INTEREST RATE      "; I
160 INPUT "  NUMBER OF YEARS    "; N
170 IF I > 1 THEN I = I / 100
180 J = LOG(1 + I) / LOG(2.71828)
190 AF = (J*(1+I)^N) / ((1+I)^N-1)
200 MA = (P * AF) / 12
210 REM  ** DISPLAY RESULTS
220 MA = INT(MA * 100) / 100
```

```
230 PRINT: PRINT
240 PRINT "MONTHLY PAYOUT IS $"; MA
250 PRINT: PRINT "----------------"
260 PRINT: GOTO 130
RUN

Sample run:

        ANNUITY:

          INITIAL AMOUNT     ? 100000
          INTEREST RATE      ? 7.5
          NUMBER OF YEARS    ? 5

        MONTHLY PAYOUT IS $ 1986.12

        ----------------

          INITIAL AMOUNT     ? 100000
          INTEREST RATE      ? 6.5
          NUMBER OF YEARS    ? 5

        MONTHLY PAYOUT IS $ 1942.81

        ----------------

          INITIAL AMOUNT     ? 0

        READY.
```

Lease v. Purchase Analysis

It has become increasingly popular for individuals to lease big-ticket items, such as cars, pianos, and entire suites of home furnishings. The assumption is that things wear out and have to be replaced, and besides, they're usually bought on credit, so leasing is about the same as buying.

But is it really? Which is the more advantageous to you, the consumer? The answer, of course, depends on many factors. This

program helps you analyze the real costs of leasing versus purchasing. It does not consider the tax advantages *pro* or *con* either alternative, as those vary widely with circumstances, nor does it take into account "hidden costs," such as the loss of interest if you withdraw money from savings to buy the item.

The "salvage" category covers the estimated trade-in value at the end of the item's expected life, or, in other words, how much you think you will be able to sell it for. "Depreciation" can be viewed as a reserve fund set aside to replace the item. It assumes an amount equivalent to the purchase price, without regard for inflation. Both of these are deferred noncash effects that you experience at the end of the purchase's life, and that is why they are separated on the analysis report.

```
NEW
100 REM  ** LEASE/PURCHASE ANALYSIS
110 PRINT CHR$(147)
120 PRINT "LEASE/PURCHASE COMPARISON:"
130 PRINT: INPUT "DESCRIPTION"; I$
140 PRINT
150 PRINT: PRINT "OPERATING EXPENSES:"
160 INPUT "FUEL AND OIL/YEAR    "; G1
170 INPUT "POWER/YEAR           "; G2
180 INPUT "SUPPLIES/YEAR        "; G3
190 INPUT "OTHER MISC/YEAR      "; G4
200 PRINT: PRINT "LEASE OPTION:"
210 INPUT "MONTHLY PAYMENT      "; L1
220 INPUT "INSURANCE/YEAR       "; L2
230 INPUT "TAXES/YEAR           "; L3
240 INPUT "LICENSES/YEAR        "; L4
250 INPUT "REPAIRS & MTNCE/YEAR"; L5
260 PRINT: PRINT "PURCHASE OPTION:"
270 INPUT "PURCHASE PRICE       "; P1
280 INPUT "EST. LIFE IN YEARS   "; Y
290 INPUT "FINANCING EXPENSE    "; P6
300 INPUT "INSURANCE/YEAR       "; P2
310 INPUT "TAXES/YEAR           "; P3
320 INPUT "LICENSES/YEAR        "; P4
330 INPUT "REPAIRS & MTNCE/YEAR"; P5
340 INPUT "SALVAGE VALUE        "; P7
```

```
350 REM  ¥¥  LEASE AND GENERAL COSTS
360 A1 = L1       : A2 = L2 / 12
370 A3 = L3 / 12 : A4 = L4 / 12
380 A5 = L5 / 12 : A6 = 0
390 A7 = G1 / 12 : A8 = G2 / 12
400 A9 = G3 / 12 : B1 = G4 / 12
410 L9 = A1+A2+A3+A4+A5+A7+A8+A9+B1
420 REM  ¥¥  PURCHASE COSTS
430 B2 = P2 / 12
440 B3 = P3 / 12 : B4 = P4 / 12
450 B5 = P5 / 12 : B6 = P6 / Y / 12
460 P9 = B2+B3+B4+B5+B6+A7+A8+A9+B1
470 C2 = P7/12/Y : C3 = P1 / Y / 12
480 T  = P9 - C2 + C3
490 REM  ¥¥  DISPLAY ANALYSIS
500 PRINT CHR$(147): PRINT
510 PRINT "LEASE/PURCHASE FOR " I$:PRINT
520 PRINT TAB(18) "LEASE" TAB(29) "PURCH
ASE"
530 X$="MO. PAYMENT ": X1=A1: X2=0 : GOS
UB 800
540 X$="INSURANCE":X1=A2:X2=B2:GOSUB 800
550 X$="TAXES"     :X1=A3:X2=B3:GOSUB 800
560 X$="LICENSES"  :X1=A4:X2=B4:GOSUB 800
570 X$="MAINTENANCE":X1=A5:X2=B5:GOSUB 8
00
580 X$="INTEREST"  :X1=A6:X2=B6:GOSUB 800
590 X$="FUEL AND OIL":X1=A7:X2=A7: GOSUB
 800
600 X$="POWER"     :X1=A8:X2=A8:GOSUB 800
610 X$="SUPPLIES"  :X1=A9:X2=A9:GOSUB 800
620 X$="MISC."     :X1=B1:X2=B1:GOSUB 800
630 PRINT TAB(18) "--------" TAB(29) "--
------"
640 X$=" CASH EXP/MO": X1=L9: X2=P9
650 GOSUB 800: PRINT
660 X$=" - SALVAGE":X1=0:X2=C2:GOSUB 800
670 X$=" + DEPREC.":X1=0:X2=C3:GOSUB 800
```

(continued)

```
680 PRINT TAB(18) "--------" TAB(29) "--
------"
690 X$="NET EXP/MO":X1=L9:X2=T:GOSUB 800
700 END
800 REM  ** FORMAT OUTPUT LINE
810 DEF FNR(X) = INT((X*100)+.499)/100
820 PRINT X$;
830 PRINT TAB(17) "$" FNR(X1);
840 PRINT TAB(28) "$" FNR(X2)
850 RETURN
```

Sample run:

```
        LEASE/PURCHASE COMPARISON

        DESCRIPTION? NEW CAR


        OPERATING EXPENSES:
        FUEL AND OIL/YEAR    ? 1200
        POWER/YEAR           ? 0
        SUPPLIES/YEAR        ? 250
        OTHER MISC/YEAR      ? 300

        LEASE OPTION:
        MONTHLY PAYMENT      ? 249.81
        INSURANCE/YEAR       ? 625
        TAXES/YEAR           ? 0
        LICENSES/YEAR        ? 55
        REPAIRS & MTNCE/YR   ? 100

        PURCHASE OPTION:
        PURCHASE PRICE       ? 11000
        EST. LIFE IN YEARS   ? 4
        FINANCING EXPENSE    ? 824.13
        INSURANCE/YEAR       ? 625
        TAXES/YEAR           ? 50
        LICENSES/YEAR        ? 55
        REPAIRS & MTNCE/YR   ? 375
        SALVAGE VALUE        ? 1500
```

```
Screen clears, then...

        LEASE/PURCHASE FOR NEW CAR

                          LEASE        PURCHASE
MO. PAYMENT             $ 249.81     $ 0
INSURANCE               $ 52.08      $ 52.08
TAXES                   $ 0          $ 4.17
LICENSES                $ 4.58       $ 4.58
MAINTENANCE             $ 8.33       $ 31.25
INTEREST                $ 0          $ 17.17
FUEL AND OIL            $ 100        $ 100
POWER                   $ 0          $ 0
SUPPLIES                $ 20.83      $ 20.83
MISC.                   $ 25         $ 25
                         --------     --------
 CASH EXP/MO            $ 460.64     $ 255.09

 - SALVAGE              $ 0          $ 31.25
 + DEPREC.              $ 0          $ 229.17
                         --------     --------
NET EXP/MO              $ 460.64     $ 453

READY.
```

Car Operating Expense

Most of us think of a car as something we have to replace now and again—always at a dramatically higher price than last time—so that we can get from one place to another. To some, it's also an

expression of individuality, and the vehicle not only for transportation but for personal freedom. We think about the cost of operating the car when we buy gas or get it worked on, but those seem like incidental expenses more in the category of petty drains on the pocketbook than as significant expenditures.

This program attempts to put a price on personal transportation by answering the question "What does it *really* cost to operate a car?" You might be quite surprised at the answer: for me, at least, it's nearly as much as the purchase price. The answer, of course, varies with individual circumstances, but it's probably more than you think.

This is more than just an academic, nice-to-know figure, too. If you have to drive 30 miles to take advantage of that big sale, how much do you have to save on the merchandise to recover the cost of travel? This program will give you a guide, and make you think before you hop in and dash off somewhere on a whim.

The program simply totals up the costs of operation that you provide and averages them over the projected lifetime and miles driven. It does build in an automatic "kicker" for unexpected repairs ($300 a year after you've driven 30,000 miles), but otherwise the costs are extrapolations of the amounts you furnish.

```
NEW
100 REM  ** CAR EXPENSE
105 DEF FNR(X)=INT((X * 100) + .5) / 100
110 PRINT CHR$(147)
120 PRINT "CAR EXPENSE:": PRINT
130 INPUT "EXPECTED LIFE IN YEARS"; LY
140 INPUT "MILES DRIVEN PER YEAR "; MY
150 M = LY * MY
160 INPUT "FINANCED (Y/N)"; X$
170 IF X$ = "Y" THEN 200
180 INPUT "PURCHASE PRICE "; PC
190 GOTO 240
200 INPUT "DOWN PAYMENT        "; P1
210 INPUT "NUMBER OF PAYMENTS "; P2
220 INPUT "MONTHLY PAYMENT     "; P3
230 PC = FNR(P1 + (P2 * P3))
240 INPUT "MILES PER GALLON    "; MG
250 INPUT "PRICE PER GALLON    "; GG
```

```
260 TG = FNR((M / MG) * GG)
270 INPUT "ANNUAL MTNCE COST  "; AM
280 AM = AM * LY
290 ER = LY * ((M - 30000) / M) * 300
300 IF ER > 0 THEN AM = FNR(AM + ER)
310 INPUT "COST OF A TIRE     "; TC
320 TC = FNR((M / 30000) * (TC * 4))
330 INPUT "INSURANCE COST/YEAR"; IP
340 IP = FNR(IP * LY)
350 INPUT "MISC EXPENSES      "; AA
360 C  = FNR(PC+TG+AM+TC+IP+AA)
370 CM = INT(((C / M)*1000) + .5)/1000
380 REM  **  OUTPUT RESULTS
390 PRINT CHR$(147)
400 PRINT "OPERATING EXPENSES:"
410 PRINT "  FUEL             $"; TG
420 PRINT "  UPKEEP           $"; AM
430 PRINT "  TIRES            $"; TC
440 PRINT "  INSURANCE        $"; IP
450 PRINT "  MISCELLANEOUS    $"; AA
460 PRINT
470 PRINT "OPERATING TOTAL    $"; C-PC
480 PRINT
490 PRINT "PURCHASE COST      $"; PC
500 PRINT
510 PRINT "TOTAL COST OF CAR  $"; C
520 PRINT
530 PRINT "MILES DRIVEN        "; M
540 PRINT
550 PRINT "COST PER MILE      $"; CM
560 END
RUN
```

(continued)

```
Sample run:
        CAR EXPENSE:

        EXPECTED LIFE IN YEARS? 4
        MILES DRIVEN PER YEAR ? 17500
        FINANCED (Y/N)? Y
        DOWN PAYMENT          ? 800
        NUMBER OF PAYMENTS    ? 36
        MONTHLY PAYMENT       ? 265.26
        MILES PER GALLON      ? 17.5
        PRICE PER GALLON      ? 1.26
        ANNUAL MTNCE COST     ? 225
        COST OF A TIRE        ? 90
        INSURANCE COST/YEAR?  425
        MISC EXPENSES         ? 200

(New screen)
        OPERATING EXPENSES:
          FUEL                $ 5040
          UPKEEP              $ 1585.71
          TIRES               $ 840
          INSURANCE           $ 1700
          MISCELLANEOUS       $ 200

        OPERATING TOTAL       $ 9365.71

        PURCHASE COST         $ 10349.36

        TOTAL COST            $ 19715.07

        MILES                   70000

        COST PER MILE         $ .282
```

Personal Net Worth

How much are you worth? Corporations and other businesses regularly assess and publish their net worth, but as individuals we seldom do so, and as a consequence we have only the vaguest idea how we're doing on making our fortunes in the world. This program lets you know how things are going.

We all earn money, spend it, and accumulate property during

our lives. Your *net worth* is the total of all you have less the total of all you owe. The things you own are called *assets;* the amounts you owe are *liabilities*. Thus your total assets are your *gross worth;* that is, the total value of all you have. Net worth supposes that you sell it all and pay off your obligations, and that's how much you have left. It is, in effect, the equity you have in yourself (and corporations, in fact, regard this difference as the "stockholders' equity" and view it as a liability).

The program, though long, is very simple. It asks you for amounts in various categories. Use whole dollars without cents, since any net-worth assessment is only an approximation anyway. It then summarizes this information and gives you the totals in each category and your net worth.

The program goes through three screens of questions and answers, and produces two screens of output. The first report screen lists your assets and "freezes"; advance to the liabilities and net-worth screen by pressing any key.

Net worth is a static figure. It does not attempt to place a value on the income-producing potential of holdings, such as investments. Instead, it deals with current "market values," consistent with the notion of liquidating everything today. The net-worth figure is useful for evaluating your financial progress on a regular basis—quarterly, for example—and comparing it with previous periods.

Try it. You'll probably find that you're richer than you think (and if you get an unpleasant surprise instead, maybe it will spur you to plan your way out of trouble).

```
NEW
100 REM  ** PERSONAL NET WORTH
110 DATA "CASH ON HAND"
120 DATA " IN BANK ACCOUNTS"
130 DATA " IN IRA/KEOGH PLANS"
140 DATA " IN RETIREMENT PLANS"
150 DATA "CASH VALUE OF SAVINGS BONDS"
160 DATA " OF LIFE INSURANCE"
170 DATA " OF STOCKS"
180 DATA "OTHER CASH VALUE ASSETS"
190 DATA "VALUE OF REAL ESTATE"
200 DATA " OF CARS"
```

(continued)

```
210 DATA " OF BOATS, PLANES, ETC."
220 DATA " OF BUSINESS EQUITY"
230 DATA "REPLACEMENT COST OF CLOTHING"
240 DATA " OF FURNITURE"
250 DATA " OF HOBBY EQUIPMENT"
260 DATA " OF JEWELRY AND FURS"
270 DATA " OF ART WORKS AND ANTIQUES"
280 DATA " OF TOOLS AND MACHINERY"
290 DATA " OF COLLECTIONS"
300 DATA "OTHER ASSETS OF VALUE"
310 DATA "SHORT-TERM DEBTS DUE"
320 DATA "BALANCE ON MORTGAGE"
330 DATA " ON CAR LOANS"
340 DATA " ON PERSONAL LOANS"
350 DATA " ON RETAIL CREDIT"
360 DATA " ON OTHER LONG-TERM DEBTS"
370 DATA "OTHER LIABILITIES"
380 REM ----------
390 CLR: DIM C(8), C$(8)
400 DIM A(12), A$(12), L(7), L$(7)
410 FOR X=1 TO  8: READ C$(X): NEXT X
420 FOR X=1 TO 12: READ A$(X): NEXT X
430 FOR X=1 TO  7: READ L$(X): NEXT X
440 REM ----------
450 POKE 53281, 12: POKE 53280, 13
460 PRINT CHR$(5): T = 28: GOSUB 1000
470 PRINT "CURRENT ASSETS:": PRINT
480 FOR X = 1 TO 8
490 PRINT C$(X) TAB(T);
500 INPUT C(X)
510 C(0) = C(0) + C(X)
520 NEXT X
530 GOSUB 1000  :REM  NEW SCREEN
540 PRINT "OTHER ASSETS:": PRINT
550 FOR X = 1 TO 12
560 PRINT A$(X) TAB(T);
570 INPUT A(X)
580 A(0) = A(0) + A(X)
590 NEXT X
600 GOSUB 1000  :REM  NEW SCREEN
```

```
610 PRINT "LIABILITIES:": PRINT
620 FOR X = 1 TO 7
630 PRINT L$(X) TAB(T);
640 INPUT L(X)
650 L(0) = L(0) + L(X)
660 NEXT X
670 REM ----------
680 GOSUB 1000  :REM  NEW SCREEN
690 PRINT "ASSETS:"
700 FOR X = 1 TO 8
710 IF C(X) = 0 THEN 730
720 PRINT C$(X) TAB(T) C(X)
730 NEXT X
740 PRINT TAB(T) "-------"
750 PRINT "TOTAL CURRENT ASSETS";
760 PRINT TAB(T) C(0): PRINT
770 FOR X = 1 TO 12
780 IF A(X) = 0 THEN 800
790 PRINT A$(X) TAB(T) A(X)
800 NEXT X
810 PRINT TAB(T) "-------"
820 PRINT "TOTAL OTHER ASSETS";
830 PRINT TAB(T) A(0)
840 PRINT TAB(T) "======="
850 PRINT "TOTAL ASSETS"
860 PRINT TAB(T) (C(0) + A(0))
870 GET X$: IF X$ = "" THEN 870
880 REM ----------
890 GOSUB 1000  :REM  NEW SCREEN
900 PRINT "LIABILITIES:"
910 FOR X = 1 TO 7
920 IF L(X) = 0 THEN 940
930 PRINT L$(X) TAB(T) L(X)
940 NEXT X
950 PRINT TAB(T) "-------"
960 PRINT "TOTAL LIABILITIES";
970 PRINT TAB(T) L(0): PRINT
980 PRINT "NET WORTH";
990 PRINT TAB(T) (A(0)+C(0)-L(0)): END
```

(continued)

```
1000 REM  ** SCREEN ADVANCE
1010 PRINT CHR$(147);
1020 PRINT "PERSONAL NET WORTH:"
1030 PRINT
1040 RETURN
RUN
```

Sample run:

```
        [Input screen #1]
        PERSONAL NET WORTH:

        CURRENT ASSETS:

        CASH ON HAND                    ? 366
         IN BANK ACCOUNTS               ? 2866
         IN IRA/KEOGH PLANS             ? 2000
         IN RETIREMENT PLANS            ? 6395
        CASH VALUE OF SAVINGS BONDS     ? 800
         OF LIFE INSURANCE              ? 1160
         OF STOCKS                      ? 1200
        OTHER CASH VALUE ASSETS         ? 0

        [Input screen #2]
        PERSONAL NET WORTH:

        OTHER ASSETS:

        VALUE OF REAL ESTATE            ? 85000
         OF CARS                        ? 6500
         OF BOATS, PLANES, ETC.         ? 0
         OF BUSINESS EQUITY             ? 0
        REPLACEMENT COST OF CLOTHING    ? 3500
         OF FURNITURE                   ? 9000
         OF HOBBY EQUIPMENT             ? 500
         OF JEWELRY AND FURS            ? 0
         OF ART WORKS AND ANTIQUES      ? 1000
         OF TOOLS AND MACHINERY         ? 2500
```

```
 OF COLLECTIONS                ? 0
OTHER ASSETS OF VALUE          ? 0

[Input screen #3]
PERSONAL NET WORTH:

LIABILITIES:

SHORT-TERM DEBTS DUE           ? 587
BALANCE ON MORTGAGE            ? 53000
 ON CAR LOANS                  ? 3450
 ON PERSONAL LOANS             ? 955
 ON RETAIL CREDIT              ? 740
 ON OTHER LONG-TERM DEBTS      ? 0
OTHER LIABILITIES              ? 800

[Output screen #1]
PERSONAL NET WORTH:

ASSETS:
CASH ON HAND                    366
 IN BANK ACCOUNTS               2866
 IN IRA/KEOGH PLANS             2000
 IN RETIREMENT PLANS            6395
CASH VALUE OF SAVINGS BONDS     800
 OF LIFE INSURANCE              1160
 OF STOCKS                      1200
                                -------
TOTAL CURRENT ASSETS            14787

VALUE OF REAL ESTATE            85000
 OF CARS                        6500
REPLACEMENT COST OF CLOTHING    3500
 OF FURNITURE                   9000
 OF HOBBY EQUIPMENT             500
 OF ART WORKS AND ANTIQUES      1000
 OF TOOLS AND MACHINERY         2500
                                -------
TOTAL OTHER ASSETS              108000
```

(continued)

```
TOTAL ASSETS                    122787

[Output screen #2]
PERSONAL NET WORTH:

LIABILITIES:

SHORT-TERM DEBTS DUE            587
BALANCE ON MORTGAGE             53000
 ON CAR LOANS                   3450
 ON PERSONAL LOANS              955
 ON RETAIL CREDIT               740
OTHER LIABILITIES               800
                                -------
TOTAL LIABILITIES               59532

NET WORTH                       63255

READY.
```

SECTION 5

CALENDARS AND CLOCKS

Day of the Week

On which day of the week were you born? What day was it when the Founding Fathers signed the Declaration of Independence? This program figures out the day of the week for any date, but there is one catch: in 1582, the calendar was changed in most of the world to the one we currently use, so any day of the week given for a date prior to that year is subject to doubt. It's still accurate in terms of our calendar, but October 12, 1492 was not necessarily a Wednesday under the old way of keeping track of the passage of time.

This program is fun and a little startling because of its instantaneous production of a day, no matter how many years before or hence. Incidentally, the New Year's Eve that will usher in the twenty-first century occurs on Friday. What a weekend that's going to be!

```
NEW
100 REM  ** DAY OF THE WEEK
110 PRINT CHR$(147)
120 DEF FNA(X) = INT(X / 4)
130 DEF FNB(X) = INT(X / 7)
140 DIM T(12), D$(7)
150 FOR X = 1 TO 12: READ T(X): NEXT X
160 FOR X = 1 TO 7: READ D$(X): NEXT X
170 PRINT "DAY OF THE WEEK FOR ANY DATE:
": PRINT
180 INPUT "DATE AS MM,DD,YYYY...";M,D,Y
190 IF M = 0 THEN END
200 GOSUB 300
210 PRINT: PRINT
220 PRINT "DATE IS A " D$(B)
230 PRINT "------------------": PRINT
240 GOTO 180
300 REM  ** FIGURE DAY OF WEEK
310 J = INT((Y - 1500) / 100)
320 A = (J * 5) + (J + 3) / 4
330 K = INT(A - FNB(A) * 7)
340 W = INT(Y / 100): V = INT(Y-W*100)
350 A = (V / 4) + V + D + T(M) + K
360 B = INT(A - FNB(A) * 7) + 1
370 IF M > 2 THEN 460
380 IF V = 0 THEN 440
390 T1 = INT(Y - FNA(Y) * 4)
400 IF T1 <> 0 THEN 460
410 IF B <> 0 THEN 430
420 B = 6
430 B = B - 1: GOTO 460
440 A = J - 1: T1 = INT(A-FNA(A)*4)
450 IF T1 = 0 THEN 410
460 IF B <> 0 THEN 480
470 B = 7
480 RETURN
490 DATA 0,3,3,6,1,4,6,2,5,0,3,5
500 DATA "SUNDAY", "MONDAY", "TUESDAY"
510 DATA "WEDNESDAY", "THURSDAY"
```

```
520 DATA "FRIDAY", "SATURDAY"
RUN


Sample run:

        DAY OF THE WEEK FOR ANY DATE:

        DATE AS MM,DD,YYYY...? 10,30,1983


        DAY IS A SUNDAY
        ----------------

        DATE AS MM,DD,YYYY...? 7,4,1776


        DATE IS A THURSDAY
        ------------------

        DATE AS MM,DD,YYYY...? 0,0,0

        READY.
```

Perpetual Calendar

Using this program, you can see the calendar for any month of any year. It's handy for history buffs, vacation planners, and anybody else who has to look ahead or behind farther than a year or so. The program uses the same day-finding subroutine (line 300 onward) as DAY OF THE WEEK to determine which day is the first of the month. And, like the preceding program, it's reliable for all years starting in 1582.

```
NEW
100 REM  **  PERPETUAL CALENDAR
110 PRINT CHR$(147):  D = 1
```

(continued)

```
120 DEF FNA(X) = INT(X / 4)
130 DEF FNB(X) = INT(X / 7)
140 DIM T(12), N(12)
150 FOR X = 1 TO 12: READ T(X): NEXT X
160 FOR X = 1 TO 12: READ N(X): NEXT X
170 INPUT "CALENDAR FOR MONTH, YEAR...";
M, Y
180 IF M = 0 THEN END
190 PRINT: GOSUB 300: PRINT
200 PRINT " SU  MO  TU  WE  TH  FR  SA"
210 H = N(M)
220 IF T1 = 0 AND M = 2 THEN H = H + 1
230 T = (B - 1) * 4
240 FOR X = 1 TO H
250 PRINT TAB(T) X;
260 T = T + 4
270 IF T > 25 THEN T = 0: PRINT
280 NEXT X
290 PRINT: PRINT: GOTO 170
300 REM  **  FIRST DAY OF MONTH
310 J = INT((Y - 1500) / 100)
320 A = (J * 5) + (J + 3) / 4
330 K = INT(A - FNB(A) * 7)
340 W = INT(Y / 100): V = INT(Y-W*100)
350 A = (V / 4) + V + D + T(M) + K
360 B = INT(A - FNB(A) * 7) + 1
370 IF M > 2 THEN 460
380 IF V = 0 THEN 440
390 T1 = INT(Y - FNA(Y) * 4)
400 IF T1 <> 0 THEN 460
410 IF B  <> 0 THEN 430
420 B = 6
430 B = B - 1: GOTO 460
440 A = J - 1: T1 = INT(A - FNA(A) * 4)
450 IF T1 = 0 THEN 410
460 IF B <> 0 THEN 480
470 B = 7
480 RETURN
490 DATA  0,3,3,6,1,4,6,2,5,0,3,5
500 DATA 31,28,31,30,31,30,31,31
```

```
510 DATA 30,31,30,31
RUN


Sample run:

        CALENDAR FOR MONTH, YEAR...? 12,1941

         SU  MO  TU  WE  TH  FR  SA
             1   2   3   4   5   6
         7   8   9   10  11  12  13
         14  15  16  17  18  19  20
         21  22  23  24  25  26  27
         28  29  30  31

        CALENDAR FOR MONTH, YEAR...? 0,0

        READY.
```

Elapsed Time in Days

How many sunrises have there been during your lifetime? How many days since Pearl Harbor? How many days did you live in your last home? did the revolutionary war last? remain in the twentieth century? Answers to questions such as these usually aren't the most compelling concerns of our lives, but it's interesting to find out.

This program computes the elapsed days between any two dates of all time, and it doesn't care in which order you enter them. It adjusts for the vagaries of leap years and century leap years, but because of rounding it might be one day off over a very long span of time (does it really matter whether it was 792,616 or 792,617 days?). It does *not* take into account the calendar change that occurred in 1582. Sorry, but again, it's hard to get excited over a minor discrepancy in that long a period.

Have fun. This is a neat program that will gladden the heart of any trivia lover.

```
NEW
100 REM  ** ELAPSED DAYS
110 DIM M(12)
120 FOR X = 1 TO 12
130 READ M(X)
140 NEXT X
150 REM  ** GET DATES
160 PRINT CHR$(147)
170 PRINT: PRINT "ELAPSED DAYS:": PRINT
180 INPUT "MONTH, DATE, YEAR"; MM,DD,YY
190 GOSUB 360
200 IF EC = 0 THEN 220
210 PRINT "** ERROR **": GOTO 180
220 D1 = J: PRINT
230 INPUT "MONTH, DATE, YEAR"; MM,DD,YY
240 GOSUB 360
250 IF EC = 0 THEN 270
260 PRINT "** ERROR **": GOTO 230
270 D2 = J: PRINT
280 REM  ** COMPUTE ELAPSED DAYS
290 ED = ABS(D1 - D2)
300 ED = ED - INT(ED / (100 * 365.25))
310 ED = ED + INT(ED / (400 * 365.25))
320 PRINT ED "DAYS BETWEEN DATES"
330 PRINT: INPUT "ANOTHER (Y/N)"; X$
340 IF X$ <> "Y" THEN END
350 PRINT: PRINT: PRINT: GOTO 180
360 REM  ** JULIAN DATE
370 EC = 0
380 IF DD > 31 THEN EC = 99: GOTO 460
390 IF MM > 12 THEN EC = 99: GOTO 460
400 X = YY * 365.25
410 J = INT(X)
420 L = X - J
430 J = J + M(MM) + DD
440 IF L <> 0 THEN 460
450 IF MM <= 2 THEN J = J - 1
460 RETURN
470 REM  ** MONTH ELAPSED DAYS
```

```
480 DATA 0, 31, 59, 90, 120, 151, 181
490 DATA 212, 243, 273, 304, 334
RUN

Sample run:

        ELAPSED DAYS:

        MONTH, DATE, YEAR? 11,18,1983

        MONTH, DATE, YEAR? 7,17,1941

         15464 DAYS BETWEEN DATES

        ANOTHER (Y/N)? Y

        MONTH, DATE, YEAR? 6,2,1969

        MONTH, DATE, YEAR? 8,14,1972

         1169 DAYS BETWEEN DATES

        ANOTHER (Y/N)? N

        READY.
```

Digital Clock

This program turns your computer into a digital clock that keeps extremely accurate time. In the center of the screen, it displays a digital readout showing the hour, minute, and second, surrounded by a box.

The program asks you to set the time by entering the hour and minutes, separated by a comma. It does not ask for seconds, so if the precise time is important, press the RETURN key at the instant the minute advances on the clock you're synchronizing with. This program will run forever, or until you press any key to stop it. It keeps the time on a 12-hour basis; if you want it to display military time instead, change 12 to 24 in line 360.

```
NEW
100 REM  ** DIGITAL CLOCK
110 PRINT CHR$(147)
120 PRINT "DIGITAL CLOCK:": PRINT
130 PRINT "ENTER CURRENT HOUR, MINUTE"
140 PRINT "  SEPARATED BY COMMA": PRINT
150 INPUT H, M
160 PRINT CHR$(147)
170 CT = TIME
180 FOR X = 1 TO 10: PRINT: NEXT X
190 PRINT TAB(14) CHR$(111);
200 FOR X = 1 TO 10: PRINT CHR$(183);: N
EXT X
210 PRINT CHR$(112): PRINT
220 PRINT TAB(14) CHR$(108);
230 FOR X = 1 TO 10: PRINT CHR$(175);: N
EXT X
240 PRINT CHR$(186) CHR$(145) CHR$(145)
250 GOSUB 390
260 REM  ** TIMER LOOP
270 GET X$: IF X$ <> "" THEN 510
280 NT = TIME
290 IF (NT - CT) < 60 THEN 260
300 REM  ** ADVANCE TIME
310 CT = NT
320 S  = S + 1
330 IF S < 60 THEN 370
340 M  = M + 1: S = 0
350 IF M > 59 THEN M = 0: H = H + 1
360 IF H > 12 THEN H = 1
370 GOSUB 390
380 GOTO 270
390 REM  ** DISPLAY TIME
400 H$ = STR$(H)
410 IF LEN(H$) = 2 THEN H$ = " " + H$
420 M$ = STR$(M)
430 IF LEN(M$) = 2 THEN M$ = "0" + RIGHT
$(M$, 1)
440 M$ = ":" + RIGHT$(M$,2)
```

```
450 S$ = STR$(S)
460 IF LEN(S$) = 2 THEN S$ = "0" + RIGHT
$(S$, 1)
470 S$ = ":" + RIGHT$(S$,2) + " "
480 T$ = CHR$(165)+H$+M$+S$+CHR$(167)
490 PRINT TAB(14) T$ CHR$(145)
500 RETURN
510 REM  ** QUIT
520 PRINT CHR$(147)
530 END
RUN
```

Chiming Digital Clock

The best of two worlds combine in this program: the accuracy and readability of a modern digital clock, with the charming and homey sound of old-fashioned chimes ringing the hour.

This program builds upon the digital clock in the last program, and, in fact, the lines listed here are additions to it and not a complete program. If you didn't type that one, go back and do so, then add these lines for the chimes.

```
100 REM  ** CHIMING DIGITAL CLOCK
350 IF M > 59 THEN M = 0: H = H+1: C = H
360 IF H > 12 THEN H = 1: C = H
375 IF C > 0 THEN GOSUB 540
495 IF C = 0 THEN POKE 54296, 0
540 REM  ** CHIME THE HOUR
550 V = 54272
560 POKE V + 1, 90: POKE V + 15, 37
570 POKE V + 5, 9:  POKE V + 24, 15
580 POKE V + 4, 21
590 FOR X = 1 TO 370: NEXT X
600 POKE V + 4, 20
610 C = C - 1
620 RETURN
RUN
```

Tick-Tock

This program is also a variation on the digital clock program. Instead of ringing the hour, however, it ticks each second. The lines below are additions to DIGITAL CLOCK and not a complete program.

```
170 CT = TIME: V = 54272
375 GOSUB 540
525 POKE V+4, 0
540 REM  ** TICK-TOCK
550 POKE V+1, 0:  POKE V+5, 0
560 POKE V+6, 0:  POKE V+24,15
570 POKE V+4,129: POKE V+23,240
580 POKE V+24, 0
590 RETURN
RUN
```

Countdown Alarm

Ovens, spacecraft launchers, and others use a countdown alarm to trigger some action at the end of an exact period of time. In this case, the countdown clock sounds a beep-beep tone at the end of the period you specify, which can be any number of hours, minutes, and seconds. Enter the values separated by commas. The display will give you an exact readout of the remaining time. When the alarm sounds, you can stop it by pressing any key.

If you're as absent-minded as I am, this program is a good thing to keep handy. You can customize it to give you a reminder message by replacing "TIME'S UP!" in line 310. For example, if Jane says "Call me back in 20 minutes," you can change line 310 to

```
310  PRINT: PRINT: PRINT "CALL JANE"
```

and when the alarm sounds, that's the message that will appear.

```
NEW
100 REM  ** COUNTDOWN ALARM
110 PRINT CHR$(147): PRINT
120 PRINT "COUNTDOWN ALARM:": PRINT
130 INPUT "HOURS, MINUTES, SECONDS"; HH,
 MM, SS
140 PRINT CHR$(147): PRINT: PRINT
150 PRINT "REMAINING TIME:": PRINT
160 M = TIME: S = 54272
170 REM  ** COUNTDOWN LOOP
180 GOSUB 390
190 SS = SS - 1
200 IF SS <> -1 THEN 250
210 SS = 59: MM = MM - 1
220 IF MM <> -1 THEN 250
230 MM = 59: HH = HH - 1
250 X$ = STR$(HH) + " HOURS," + STR$(MM)
260 X$ = X$ + " MIN,"+STR$(SS)+" SEC"
270 PRINT X$ + "    " + CHR$(145)
280 IF HH=0 AND MM=0 AND SS=0 THEN 300
290 GOTO 170
300 REM  ** THE MOMENT HAS ARRIVED
310 PRINT: PRINT: PRINT "TIME'S UP!"
320 GOSUB 430: GOSUB 390
330 GET X$: IF X$ <> "" THEN 360
340 GOSUB 480: GOSUB 390
350 GET X$: IF X$ = "" THEN 320
360 REM  ** SHUT DOWN
370 GOSUB 480
380 END
390 REM  ** ONE-SECOND TIMER
400 IF (TIME - M) < 60 THEN 400
410 M = TIME
420 RETURN
430 REM  ** SOUND THE ALARM
440 POKE S+6, 240:  POKE S+24, 15
450 POKE S+0, 35:   POKE S+1,  40
460 POKE S+4, 33
```

(continued)

```
470 RETURN
480 REM  ¥¥  SILENCE
490 FOR X = 0 TO 24: POKE S+X, 0: NEXT X
500 RETURN
RUN
```

SECTION 6
UTILITY PROGRAMS

General

Utilities are programs intended chiefly to aid the cause of programming itself, rather than to perform useful tasks for the "end user" (the consumer of programs). They make a programmer's life easier by removing some of the drudgery of the work. As a result, the programs in this section are a valuable addition to your software library, even if you only write programs for your own amusement.

There are five utilities here for making sprites, doing code conversions, and programming graphics. The graphics utilities are especially important because they provide a comprehensive environment in which to develop superb screen displays without having to remember all the rigamarole the Commodore 64 requires for its graphics. They consist of subroutines that support the important graphics functions, and that you can set up and call with simple instructions from your programs. The remaining five programs in this section demonstrate how to use the graphics utilities, and provide some striking displays as well.

Sprite-Making Tool #1

This is an extraordinarily useful tool for making sprites on the screen and seeing the results, and it lets you touch up and fiddle around with the sprite until you've got it right. And when you do, it gives you the values to put in your program's DATA statements. This program, along with all the theory that underlies it, is from *Mastering Sight and Sound on the Commodore 64*, by Yours Truly. No collection of programs for the Commodore 64 is complete without it.

How to use this program:

1. Type RUN. A grid of dots appears on the screen, representing the dots in the sprite image.
2. Use the cursor keys to move the cursor about the grid. Type asterisks (*) to form the sprite.
3. Move the cursor down to the line immediately below the last row of the grid and type RUN 200. Be careful not to shift the grid upward by moving the cursor too far down.
4. The sprite image forms in the lower right corner of the screen. In the upper right corner is a menu that says "K IF OK, N IF NOT", followed by a question mark.
5. If the sprite doesn't suit you yet, type N and rework the sprite according to steps 2 and 3. You can "undo" an asterisk (to make the dot transparent again) by typing a period to replace it. Repeat step 4 to see the revised image. You can repeat this step (5) as many times as necessary to get the sprite the way you want it.
6. When the sprite is satisfactory, type K. The grid vanishes and is replaced by the 63 values that represent the sprite image. Copy them and put them into your program.

One further note: If you want to expand the sprite in either direction (or both), use POKE commands from the keyboard before you run this program. Afterward, be sure to POKE the expansion byte(s) with 0.

```
NEW
100 REM  ** SPRITE MAKER #1
110 POKE 53269, 0: PRINT CHR$(147)
```

```
120 FOR Y = 1 TO 21
130 FOR X = 0 TO 23: PRINT ".";: NEXT X
140 PRINT: NEXT Y
150 PRINT CHR$(19)
160 PRINT TAB(28) "STEP 1:"
170 PRINT TAB(28) "RUN 200 NEXT"
180 FOR Y = 3 TO 20: PRINT: NEXT Y: END
190 REM ----------
200 REM  ¥¥  STEP 2 BUILDS SPRITE
210 SL=14336: SP=SL/64: PRINT CHR$(19)
220 PRINT TAB(28) "STEP 2:"
230 PRINT TAB(28) "K IF OK      "
240 PRINT TAB(28) "N IF NOT": PRINT TAB(
28);
250 FOR P=0 TO 63: POKE SL+P,0: NEXT P
260 POKE 2040, SP: POKE 53287, 1
270 POKE 53248, 255: POKE 53249, 200
280 POKE 53269, 1: E = 8: B = 0
290 FOR Y = 1 TO 21
300 FOR X = 0 TO 23: P=1024+X+(Y¥40)
310 E = E - 1: V = PEEK(P)
320 IF V = 46 THEN 340
330 POKE SL + B, PEEK(SL + B) OR 2^E
340 IF E = 0 THEN E = 8: B = B + 1
350 NEXT X: NEXT Y
360 INPUT X$
370 IF X$ = "K" THEN 410
380 PRINT CHR$(19)
390 FOR X = 1 TO 5: PRINT TAB(28);
400 PRINT" [12 spaces] ":NEXT X:GOTO 160
410 REM  ¥¥  PRINT VALUES FOR SPRITE
420 POKE 53269, 0: PRINT CHR$(147)
430 FOR Y = 0 TO 20
440 FOR X = 0 TO  2: P = X + (Y ¥ 40)
450 PRINT PEEK(SL + P),
460 NEXT X
470 PRINT: NEXT Y
RUN
```

Sprite-Making Tool #2

This is similar to the preceding program, except that it helps you make multicolor sprites. Like SPRITE-MAKING TOOL #1, it comes from *Mastering Sight and Sound on the Commodore 64*, and is included here to round out the collection of programs.

How to use this program:

1. Type RUN. A grid of dots appears on the screen, representing the dots in the sprite image.
2. Use the cursor keys to move the cursor about the grid. Type asterisks (*) to form the sprite.
3. Move the cursor down to the line immediately below the last row of the grid and type RUN 200. Be careful not to shift the grid upward by moving the cursor too far down.
4. The sprite image forms in the lower right corner of the screen. In the upper right corner is a menu that says "K IF OK, N IF NOT", followed by a question mark.
5. If the sprite doesn't suit you yet, type N and rework the sprite according to steps 2 and 3. You can "undo" an asterisk (to make the dot transparent again) by typing a period to replace it. Repeat step 4 to see the revised image. You can repeat this step (5) as many times as necessary to get the sprite the way you want it.
6. When the sprite is satisfactory, type K. The grid vanishes and is replaced by the 63 values that represent the sprite image. Copy them and put them into your program.

One further note: If you want to expand the sprite in either direction (or both), use POKE commands from the keyboard before you run this program. Afterward, be sure to POKE the expansion byte(s) with 0.

```
NEW
10 A$ = "MULTICOLOR SPRITE MAKER"
20 PRINT CHR$(147): PRINT A$
30 PRINT: PRINT
40 INPUT "NUMBER FOR COLOR 1"; C1
50 INPUT "NUMBER FOR COLOR 2"; C2
60 INPUT "NUMBER FOR COLOR 3"; C3
```

```
70 POKE 53285, C1: POKE 53286, C3
80 POKE 53287, C2
100 REM  ** WORK GRID
110 POKE 53269, 0: PRINT CHR$(147)
120 FOR Y = 1 TO 21
130 FOR X = 0 TO 23 STEP 2: PRINT ". ";:
 NEXT X
140 PRINT: NEXT Y
150 PRINT CHR$(19)
160 PRINT TAB(28) "STEP 1:"
170 PRINT TAB(28) "RUN 200 NEXT"
180 FOR Y = 3 TO 20: PRINT: NEXT Y: END
190 REM ----------
200 REM  ** STEP 2 BUILDS SPRITE
210 SL=14336: SP=SL/64: PRINT CHR$(19)
220 PRINT TAB(28) "STEP 2:"
230 PRINT TAB(28) "K IF OK     "
240 PRINT TAB(28) "N IF NOT": PRINT TAB(
28);
250 FOR P = 0 TO 63: POKE SL+P,0: NEXT P
255 POKE 53276, 1
260 POKE 2040, SP: POKE 53287, 1
270 POKE 53248, 255: POKE 53249, 200
280 POKE 53269, 1: E = 256: B = 0
290 FOR Y = 1 TO 21
300 FOR X = 0 TO 23 STEP 2: P = 1024 + X
+ (Y * 40)
310 E = E / 4: V = PEEK(P)
320 IF V = 46 THEN 340
325 V = V AND 3
330 POKE SL + B, PEEK(SL + B) OR (E * V)
340 IF E = 1 THEN E = 256: B = B + 1
350 NEXT X: NEXT Y
360 INPUT X$
370 IF X$ = "K" THEN 410
380 PRINT CHR$(19)
390 FOR X = 1 TO 5: PRINT TAB(28);
400 PRINT" [12 spaces] ":NEXT X:GOTO 160
410 REM  ** PRINT VALUES FOR SPRITE
```

(continued)

```
420 POKE 53269, 0: PRINT CHR$(147)
430 FOR Y = 0 TO 20
440 FOR X = 0 TO  2: P = X + (Y * 3)
450 PRINT PEEK(SL + P),
460 NEXT X
470 PRINT: NEXT Y
RUN
```

Code Convertor

The Commodore 64 uses two different sets of values to make the same characters, depending on how the program places the characters on the screen. For every possible character that the computer can make, there is an ASCII (American Standard Code for Information Interchange) code. The ASCII codes consist of numbers between 0 and 255 that stand for graphic symbols or control codes; ASCII code 65, for example, means the letter "A."

When you want to display the letter A, you can do it with the BASIC instruction

```
PRINT "A"
```

and the Commodore 64 knows and does exactly what you want. You can also use the CHR$ function to display an A. CHR$ means "make the character whose ASCII value follows." Thus, the instruction

```
PRINT CHR$(65)
```

also places an A on the screen. CHR$ is usually employed to generate nonprintable control characters, but you can use it this way as well.

Confusion arises, however, when you want to POKE a character into screen memory while building a graphics display. The reason is that, although most of the ASCII codes have a corresponding POKE code, the numeric values differ.

The Commodore 64 *User's Guide* shows these two character sets in Appendixes E and F, but it never really explains that there is a numeric correlation between them, so sometimes programmers

end up flipping back and forth between the appendixes, increasing the chances of introducing an error.

This program is an aid that shows, for any character you type, its ASCII code, its CHR$ code (always the same), and the corresponding POKE code. Stop the program by entering STOP in response to the prompt.

For a more comprehensive discussion of this subject, see *Mastering Sight and Sound on the Commodore 64,* Chapter 4.

```
NEW
100 REM  ** CODE CONVERSION
110 PRINT CHR$(147)
120 PRINT "CODE CONVERSION:"
130 PRINT: INPUT "CHARACTER"; C$
140 IF C$ = "STOP" THEN END
150 AC = ASC(C$)
160 IF AC = 255 THEN AC=126
170 IF AC > 223 THEN AC=AC-64
180 IF AC > 191 THEN AC=AC-96
190 IF AC > 159 THEN PC=AC-64:GOTO 250
200 IF AC > 127 THEN 240
210 IF AC >  95 THEN PC=AC-32:GOTO 250
220 IF AC >  63 THEN PC=AC-64:GOTO 250
230 IF AC >  31 THEN PC=AC   :GOTO 250
240 PC = -1
250 PC$ = STR$(PC)
260 IF PC < 0 THEN PC$ = "NONE"
270 PRINT "  ASCII        " AC
280 PRINT "  CHR$ CODE    " AC
290 PRINT "  POKE CODE    " PC$
300 GOTO 130
RUN
```

```
Sample run:

        CODE CONVERSION:

        CHARACTER? K
```

(continued)

```
   ASCII          75
   CHR$ CODE      75
   POKE CODE      11

CHARACTER? 7
   ASCII          55
   CHR$ CODE      55
   POKE CODE      55

CHARACTER? [
   ASCII          91
   CHR$ CODE      91
   POKE CODE      27

CHARACTER? STOP

READY.
```

High-Resolution Graphics Environment

This is not a complete program, but rather an "environment" in which to develop your own high-resolution graphics (HRG) programs. That means that it furnishes "canned" HRG services that you can call as subroutines from your programs, thereby relieving you of the tedium of coding all the POKEs and PEEKs and what not that the Commodore 64 uses to make graphics.

This programming environment deals with bit-mapped HRG, one of the two graphics modes available on the Commodore 64. The other mode, multicolor HRG (or pixel-mapped graphics) also has an environment that is presented a few listings hence. The differences between—and theories underlying—the graphics modes are discussed in detail in *Mastering Sight and Sound on the Commodore 64*.

The way to use this environment is as follows:

1. Type the program as shown in the following listing.
2. Save it on a disk or tape as "HRG".
3. Each time you want to write a graphics program using bit-

mapped mode, load the environment from storage into the computer.

4. After the load is complete, type your graphics program starting at line 1000. It must end before line 9960. Usage of the environment's services is discussed below.
5. Test the program to make sure it works properly.
6. Save it under a name other than HRG. This will save both the graphics services and your instructions as a complete program that you can then reload and run directly (without reloading HRG).

Using the HRG Environment's Services

The three programs following the environment listing illustrate the application of these services. This is a description of each of them.

1. *Background/foreground colors:* You automatically get white on a brown background with the HRG environment, as shown here. You can change this "default" combination by altering line 140, which currently reads

```
POKE 49172, (1 * 16) + 9
```

The 9 is brown, the 1 in the parentheses is white. Change these digits to the desired combination using the color numbers shown on page 61 of your Commodore 64 *User's Guide*. As an alternative, you can include an instruction in your program that will override this combination. Code it the same as shown here, but with your desired colors substituted. The next instruction is

```
SYS 49171
```

which calls a machine-language subprogram to reset the colors.

2. *Turning on a dot on the screen:* Graphics works one dot at a time on the Commodore 64. There are 320 dots horizontally, numbered 0–319, and 200 dots vertically (0–199). The horizontal position is called by the variable name "X" and the vertical by "Y." Therefore you must assign the desired position coordinates to "X" and "Y" and then issue the instruction

```
GOSUB 170
```

The effect of this is to "turn on" the dot (give it the foreground color) at the specified position on the screen.

3. *Changing foreground color:* Within a character cell of 8 × 8

dots, you may have any of the 16 colors as the *foreground* (the color assumed by a dot that is "turned on"). The environment sets all cells to the same foreground color; this routine is used to change it to something else within a given cell.

The Commodore 64 has 40 columns by 25 lines, for a total of 1000 character cells on the screen. The position within the line is given by the variable "X" and the line number by "Y." The columns number 0–39, the rows 0–24. Therefore, assign the column and row coordinates to "X" and "Y," respectively. Next assign the number (page 61 of the *User's Guide*) of the color you want to the variable "CN." Finally, issue the instruction

```
GOSUB 250
```

to put the change into effect. (HRG DEMO 1 illustrates this by changing four cells from white to blue.)

4. *Changing background color:* This is the same as item 3 above, except that it changes the background color in the cell specified by "X" and "Y." The *background color* is the color that a dot has when it is *not* turned on. Load the location into "X" and "Y" and the desired background color into "CN," then write

```
GOSUB 300
```

to alter the background color in the specified cell.

5. *Painting a line:* This routine turns on all the dots in a row between two dot "X" coordinates; in other words, it paints a solid horizontal line between two points. Specify the starting point's dot "X" coordinate (0–319) in the variable "SX" and the ending location in "EX." You must also give the row number (0–199) in variable "Y." Then issue the instruction

```
GOSUB 350
```

to paint the line. You can use this routine to fill in a large area by stepping one row at a time and painting in each row, in the manner of painting a wall with separate brush strokes. (HRG DEMO 1 uses this routine to make an hourglass figure.)

6. *Stopping the program:* The environment has instructions to end the program and restore normal graphics mode starting at line 9950. Have your program GOTO 9950 to stop, or else fall off the last instruction into 9950. Line 9960 is a loop that freezes the display until you tap the SPACE bar or another key. The environ-

ment then clears the screen and places the computer into the same condition it had before the program began.

The HRG environment is a great time-saver for graphics programmers, and it has everything you need to create stunning displays with your computer.

```
NEW
10 REM  ** HRG ENVIRONMENT
20 FOR A = 49152 TO 49204
30 READ B: POKE A, B: NEXT A
40 DATA 169,0,160,0,162,32,153,0,32
50 DATA 200,208,250,238,8,192
60 DATA 202,208,244,96
70 DATA 169,16,160,232,162,4,141,0,4
80 DATA 238,26,192,208,3,238,27,192
90 DATA 136,208,242,202,208,239,169,0
100 DATA 141,26,192,169,4,141,27,192,96
110 POKE 53265, PEEK(53265) OR 32
120 POKE 53272, PEEK(53272) OR 8
130 SYS 49152
140 POKE 49172, (1 * 16) + 9
150 POKE 53280, 0: SYS 49171
160 GOTO 1000   :REM  START PROGRAM
165 REM ----------
170 REM  ** TURN ON DOT AT XY
180 IF X < 0 OR X > 319 THEN 240
190 IF Y < 0 OR Y > 199 THEN 240
200 BA = (INT(Y/8)*320) + (INT(X/8)*8)
210 BA = BA + (Y AND 7) + 8192
220 BP = 7 - (X AND 7)
230 POKE BA, (PEEK(BA) OR 2^BP)
240 RETURN
245 REM ----------
250 REM  ** CHANGE FOREGROUND COLOR
260 CB = 1024 + (Y * 40) + X
270 POKE CB,(PEEK(CB) AND 15)+(CN * 16)
280 RETURN
290 REM ----------
300 REM  ** CHANGE BACKGROUND COLOR
```

(continued)

```
310 CB = 1024 + (Y * 40) + X
320 POKE CB,(PEEK(CB) AND 240) + CN
330 RETURN
340 REM ----------
350 REM  **  PAINT ONE LINE
360 CX = SX
370 IF SX <= EX THEN 390
380 SX = EX: EX = CX: CX = SX
390 REM  **  START OF LINE
400 X = CX: GOSUB 170
410 IF BP > (EX - SX + 1) THEN 550
420 PV = 2^(BP + 1) - 1
430 POKE BA, PEEK(BA) OR PV
440 CX = CX + BP + 1: BA = BA + 8
450 REM  **  WHOLE BYTE
460 BR = EX - CX + 1
470 IF BR < 8 THEN 500
480 POKE BA, 255
490 BA = BA + 8: CX = CX + 8: GOTO 450
500 REM  **  END OF LINE
510 IF BR < 0 THEN BR = 0
520 PV = 256 - 2^(8 - BR)
530 POKE BA, PEEK(BA) OR PV
540 RETURN
550 REM  **  BIT PLOT
560 FOR X = SX TO EX: GOSUB 170: NEXT X
570 RETURN
580 REM ----------
1000 REM  **  PROGRAM START

  (Your program goes here, starting at
       line 1000)

9950 REM  **  END OF PROGRAM
9960 GET X$: IF X$ = "" THEN 9960
9970 POKE 53265, PEEK(53265) AND 223
9980 POKE 53272, PEEK(53272) AND 247
9990 PRINT CHR$(147): END
```

HRG Demonstration #1

Now that you've typed in and saved the HRG environment, you're no doubt eager to try it and see how it works. This program is an insert to the "master," meaning that the HRG environment must already be in the computer's memory when you type it.

It demonstrates "painting" a horizontal line (line 1020), drawing a vertical line (lines 1040–1050), making an hourglass figure (lines 1070–1130), and changing foreground colors (lines 1150–1180).

The brevity and simplicity of this program ought to convince you of the HRG environment's worth; if you had to write a program from scratch to do even these relatively simple graphics, it would run to several screens and probably take you a couple of hours just looking up how to do this stuff. *With* the HRG environment, it's quite easy to create very striking "computer art" displays such as this one.

```
 (NOTE: the HRG environment must be in
    the computer's memory for this
    program to work.)

1010 REM  ** HORIZONTAL LINE
1020 Y=100: SX=0: EX=319: GOSUB 350
1030 REM  ** VERTICAL LINE
1040 X = 160
1050 FOR Y = 0 TO 199: GOSUB 170: NEXT Y
1060 REM  ** TOP OF HOURGLASS
1070 SX = 110: EX = 210: Y = 50
1080 GOSUB 350: Y=Y+1: SX=SX+1: EX=EX-1
1090 IF SX < EX THEN 1080
1100 REM  ** BOTTOM OF HOURGLASS
1110 Y = Y + 1: GOSUB 350
1120 SX = SX - 1: EX = EX + 1
1130 IF SX >= 110 THEN 1110
1140 REM  ** CHANGE FOREGROUND
1150 FOR Y = 8 TO 16 STEP 8
1160 FOR X = 19 TO 20
1170 CN = 6: GOSUB 250
1180 NEXT X: NEXT Y
RUN
```

HRG Demonstration #2

This program, like the preceding one, is an insert that works within the HRG environment. It draws a symmetrical pattern that looks like a corporate logo, a diamond with a sine wave in its center. The purpose of the program is to demonstrate the making of diagonal lines and curves using the HRG environment.

The subroutine between lines 1110 and 1170 plots a line between two points. The origin of the line is given by X1 and Y1, the destination by X2 and Y2. Lines 1120–1190 set up four sets of coordinates for the four lines that form the diamond.

Lines 1200–1270 plot a sine curve. Line 1210 provides positioning information, where H is the horizontal placement, V is the vertical placement, and P is the relative horizontal position at the moment. Lines 1220–1270 form a loop that steps through 360° counting by 2s. Line 1230 converts degrees to radians, an angle measurement required by the BASIC trigonometric function SIN, which appears in the next line. It calculates the vertical placement of the sine of the current angle. Line 1250 provides horizontal position and calls the environment to turn on the dot at that location. Line 1260 then steps the horizontal position a half-point to the right. The loop repeats through the entire cycle of the sine wave.

```
  (NOTE: the HRG environment must be in
     the computer's memory for this
     program to work.)

1010 REM  ** PLOT DIAGONAL LINES
1020 X1 =   0: Y1 = 100
1030 X2 = 160: Y2 =   0: GOSUB 1110
1040 X1 = 160: Y1 =   0
1050 X2 = 319: Y2 = 100: GOSUB 1110
1060 X1 = 319: Y1 = 100
1070 X2 = 160: Y2 = 199: GOSUB 1110
1080 X1 = 160: Y1 = 199
1090 X2 =   0: Y2 = 100: GOSUB 1110
1100 GOTO 1180
1110 REM  ** DRAW A LINE
1120 D = SQR((X1 - X2)^2 + (Y1 - Y2)^2)
```

```
1130 XV = (X2-X1) / D: YV = (Y2-Y1) / D
1140 X = X1: Y = Y1: D = INT(D + .5)
1150 FOR I = 1 TO D: GOSUB 170
1160 X = X + XV: Y = Y + YV: NEXT I
1170 RETURN
1180 REM  ** PAINT BISECTOR
1190 Y=100: SX=0: EX=319: GOSUB 350
1200 REM  ** PLOT SINE CURVE
1210 H = 115: V = 100: P = 0
1220 FOR D = 0 TO 360 STEP 2
1230 A = D * (π / 180)
1240 Y = V - INT((SIN(A) * 30) + .5)
1250 X = P + H: GOSUB 170
1260 P = P + .5
1270 NEXT D
RUN
```

HRG Demonstration #3

The last of the inserts for the HRG environment, this program demonstrates the effects of altering the background and foreground colors in cells. It produces a background of repeating stripes of shades from white to black, passing through the three grays, and superimposes a brilliantly striped Easter egg that seems to float in front of the background. The stripes in the background are vertical, those in the foreground horizontal.

You can watch the first part of the program altering the background as the grays march across the screen. The egg emerges next in pure white. Finally, stripes color it as the foreground is altered by the program. It's actually changing the cells outside the egg as well, but since no foreground dots are turned on except within the egg, you can't see them.

```
  (NOTE: the HRG environment must be in
     the computer's memory for this
     program to work.)

1000 REM  ** HRG DEMO #3
```

(continued)

```
1010 POKE 53280, 0: N = 1
1020 FOR X = 0 TO 39
1030 FOR Y = 0 TO 24
1040 ON N GOTO 1050,1060,1070,1080,1090
1050 CN =  1: GOTO 1100
1060 CN = 15: GOTO 1100
1070 CN = 12: GOTO 1100
1080 CN = 11: GOTO 1100
1090 CN =  0
1100 GOSUB 300     :REM  B/G STRIPES
1110 NEXT Y
1120 N = N + 1: IF N = 6 THEN N = 1
1130 NEXT X: N = 1
1140 REM  ** EASTER EGG
1150 RC = 60: CH = 160: CV = 100
1160 FOR A = 90 TO 270 STEP .5
1170 RD = A * (π / 180)
1180 Y  = INT(RC * SIN(RD) + CV + .5)
1190 IF Y = LY THEN 1240
1200 SX = INT(RC * COS(RD) + CH + .5)
1210 EX = CH - SX + CH
1220 IF EX > (CH + 3) THEN GOSUB 350
1230 LY = Y
1240 NEXT A
1250 REM ----------
1260 REM  **  F/G STRIPES
1270 FOR Y =  4 TO 20
1280 FOR X = 10 TO 30
1290 ON N GOTO 1300,1310,1320,1330,1340
1300 CN =  2: GOTO 1350
1310 CN = 14: GOTO 1350
1320 CN =  8: GOTO 1350
1330 CN =  5: GOTO 1350
1340 CN =  7
1350 GOSUB 250
1360 NEXT X
1370 N = N + 1: IF N = 6 THEN N = 1
1380 NEXT Y
RUN
```

Multicolor High-Resolution Graphics Environment

This is also not a complete program, but is, instead, a set of "canned" services that surround and support your multicolor high-resolution graphics (MHRG) programs, and that you can call as subroutines. It is similar to the HRG environment, but the calls are a little different.

Multicolor (or pixel-mapped) HRG differs from standard (bit-mapped) HRG in a couple of fundamental ways. It permits you to have up to four different colors within any 8 × 8-dot character cell. To accomplish this, it sacrifices horizontal resolution by using pixels (short for picture elements), which consist of two dots side by side. Thus, an 8 × 8-dot cell is really 4 pixels wide by 8 high. Each pixel contains a 2-bit number (0–3) that indicates its color. These numbers (color pointers) refer to color numbers stored elsewhere. The 2-bit color pointer 0 is the screen background color, which is consistent for all cells. The other three colors are assigned to individual cells, and can potentially differ in each 4 × 8-pixel cell throughout the display. This creates enormous possibilities for graphics. In bit-mapped HRG, there are 320 horizontal dots (X coordinates) by 200 vertical dots (Y coordinates); in MHRG, there are 160 X coordinates by 200 Y coordinates. For more information and the theories of MHRG, see *Mastering Sight and Sound on the Commodore 64*.

Use this environment as follows:

1. Type the program as shown in the following listing.
2. Save it on a disk or tape as "MHRG."
3. Each time you want to write a graphics program using pixel-mapped mode, load the environment from storage into the computer.
4. After the load is complete, type your graphics program starting at line 1000. It must end before line 9950. Usage of the environment's services is discussed below.
5. Test the program to make sure it works properly.
6. Save it under a name other than MHRG. This will save both the graphics services and your instructions as a complete program that you can then reload and run directly (without reloading MHRG).

Using the MHRG Environment's Services

The two programs following the environment listing illustrate applications of the services described here.

1. *Background/foreground colors:* You automatically get white as color pointer #3 ("foreground") against a brown background (color pointer #Ø) with the MHRG environment as shown here. You can change this combination by altering line 15Ø, which now reads

```
POKE 49172, 1: POKE 53281, 9
```

The "1" is white and "9" is brown. Change these digits to the desired combination using the color numbers shown on page 61 of your Commodore 64 *User's Guide*.

As an alternative, you can include instructions in your program to override this combination. To change the background:

```
POKE 53281, [color number]
```

and to change the color associated with color pointer #3:

```
POKE 49179, 216: POKE 49172, [color number]
SYS 49171: POKE 49179, 4
```

which calls the machine-language subprogram given by the DATA statements in lines 4Ø–6Ø.

Change the background color (given by color pointer value #Ø) by POKEing the appropriate color number into 53281.

2. *Setting colors for color pointers #1 and #2:* The environment does not establish the colors associated with color pointers #1 and #2. Before using them, therefore, you have to set them up. The following instructions establish these colors for the screen as a whole:

```
POKE 49172, (16 * [color 1] + [color 2]
SYS 49171
```

where [color 1] is a number 0–15 that assigns a color to pointer #1 and [color 2] assigns a color to pointer #2.

3. *Turning on a pixel on the screen:* MHRG consists of pixels on the Commodore 64, a pixel being two dots side by side. There are 160 dots horizontally numbered Ø–159, and 200 pixels vertically (Ø–199). A pixel is twice as wide as it is high. Each horizontal position is given by the variable name "X" and the vertical by "Y". Therefore, you must assign the desired position coordinates to "X" and "Y" and the pointer value Ø–3 to "CP", then write

GOSUB 190

The effect of this is to "turn on" the pixel at the specified position on the screen, giving it the color pointed to by "CP". (MHRG DEMONSTRATION #1 uses this routine to place "stars.")

4. *Changing color #1:* Within a character cell of 4 × 8 pixels, color pointer value 1 may indicate any one of 16 colors. Item 2 above tells how to set all cells to the same color #1; this section tells how to change it within a given cell.

The Commodore 64 has 40 columns by 25 lines, for a total of 1000 character cells on the screen. The position within the line is given by the variable "X" and the line number by "Y". The columns number 0–39, the rows 0–24. Therefore, assign the column and row coordinates to "X" and "Y", respectively. Next assign the number (page 61 of the *User's Guide*) of the color you want color pointer value 1 to indicate to the variable "CN". Finally, issue the instruction

GOSUB 290

to put the change into effect within the cell.

5. *Changing color #2:* This is the same as item #4 above, except that it changes the color indicated by pointer value #2 in the cell at "X" and "Y." Load the location into "X" and "Y" and the color number to be given by pointer value #2 into "CN", then write

GOSUB 340

6. *Changing color #3:* This is the same as items #4 and #5 above, but it changes the color indicated by pointer value #3 in the cell at "X" and "Y". Do as above, but call the appropriate routine with

GOSUB 390

7. *Painting a line:* This routine turns on all the pixels in a row between two "X" coordinates; in other words, it paints a solid horizontal line between two points. Specify the starting pixel's "X" coordinate (0–159) in the variable "SX" and the ending location in "EX". You must also give the row number (0–199) in "Y" and the color pointer value for the line in CP, then write

GOSUB 350

to paint the line. You can use this routine to fill in a large area by

stepping one row at a time and painting in each row, in the manner of painting a wall with separate brush strokes. (MHRG DEMONSTRATION 1 uses this routine to make stripes on a flag bunting.)

8. *Stopping the program:* The environment has instructions to end the program and restore normal graphics mode starting at line 9950. Have your program GOTO 9950 to stop, or else fall off the last instruction into 9950. Line 9960 is a loop that freezes the display until you tap the SPACE bar or another key. The environment then clears the screen and places the computer into the same condition it had before the program began.

The MHRG environment is a great time-saver for graphics programmers, and it has everything you need to create stunning displays with your computer.

```
10 REM  ** MHRG ENVIRONMENT
20 FOR A = 49152 TO 49204
30 READ B: POKE A, B: NEXT A
40 DATA 169,0,160,0,162,32,153,0,32
50 DATA 200,208,250,238,8,192
60 DATA 202,208,244,96
70 DATA 169,16,160,232,162,4,141,0,4
80 DATA 238,26,192,208,3,238,27,192
90 DATA 136,208,242,202,208,239,169,0
100 DATA 141,26,192,169,4,141,27,192,96
110 POKE 53270, PEEK(53270) OR 16
120 POKE 53265, PEEK(53265) OR 32
130 POKE 53272, PEEK(53272) OR  8
140 POKE 49179, 216
150 POKE 49172, 1: POKE 53281, 9
160 SYS 49171: SYS 49152: POKE 49179,4
170 GOTO 1000   :REM  START PROGRAM
180 REM  ----------
190 REM  **  COLOR PIXEL AT XY
200 IF X < 0 OR X > 159 THEN 270
210 IF Y < 0 OR Y > 199 THEN 270
220 BA = (INT(Y/8)*320)+(INT((X*2)/8)*8)
230 BA = BA + (Y AND 7) + 8192
240 BP = 3 - (X AND 3)
250 PM = 4^BP: PP = 255 - (PM * 3)
```

```
260 POKE BA,((PEEK(BA) AND PP) OR CP*PM)
270 RETURN
280 REM  ----------
290 REM  **  CHANGE COLOR #1 IN CELL XY
300 CB = 1024 + (Y * 40) + X
310 POKE CB,(PEEK(CB) AND 15)+(CN * 16)
320 RETURN
330 REM  ----------
340 REM  **  CHANGE COLOR #2 IN CELL XY
350 CB = 1024 + (Y * 40) + X
360 POKE CB,(PEEK(CB) AND 240) + CN
370 RETURN
380 REM  ----------
390 REM  **  CHANGE COLOR #3 IN CELL XY
400 CB = 55296 + (Y * 40) + X
410 POKE CB, CN
420 RETURN
430 REM  ----------
440 REM  **  PAINT LINE IN COLOR CP
450 CX = SX
460 IF SX <= EX THEN 480
470 SX = EX: EX = CX: CX = SX
480 REM  **  START OF LINE
490 X = CX: GOSUB 190
500 IF BP > (EX - SX + 1) THEN 680
510 IF BP = 3 THEN 570
520 IF BP=0 THEN CX=CX+1:BA=BA+8:GOTO570
530 BR = BP
540 FOR PB = BR TO 0 STEP -1: X = CX
550 GOSUB 180: CX = CX + 1
560 NEXT PB: BA = BA + 8
570 REM  **  WHOLE BYTE
580 PB = 0
590 FOR PV = 3 TO 0 STEP -1
600 PB = PB + (CP * 4^PV)
610 NEXT PV
620 BR = EX - CX + 1
630 IF BR < 4 THEN 660
640 POKE BA, PB: BA = BA + 8
```

(continued)

```
650 CX = CX + 4: GOTO 620
660 REM  ** END OF LINE
670 IF BR <= 0 THEN 700
680 REM  ** PIXEL PLOT
690 FOR X = CX TO EX: GOSUB 180: NEXT X
700 RETURN
1000 REM  ** PROGRAM START

      (Your program goes here)

9950 REM  ** END OF PROGRAM
9960 GET X$: IF X$ = "" THEN 9960
9970 POKE 53265, PEEK(53265) AND 223
9980 POKE 53270, PEEK(53270) AND 239
9990 POKE 53272, PEEK(53272) AND 247
9999 PRINT CHR$(147): END
```

MHRG Demonstration #1

This program is an insert to the MHRG environment that illustrates use of some of its features for creating pixel-mapped graphics. It draws a patriotic figure consisting of a red, white, and blue flag bunting studded with stars and surrounded by an oval of stars against a black background. Specifically:

- Line 1010 sets the background and border to black.
- Line 1020 establishes red (2) as the color associated with color pointer value #1 and blue (6) as pointer value #2.
- Lines 1030–1120 draw stripes of red, white, and blue, using the painting routine.
- Lines 1040–1200 place stars at regular intervals in the red and blue stripes of the bunting.
- Lines 1210–1290 create an oval of white stars around the bunting.

```
  (NOTE: the MHRG environment must be in
     the computer's memory for this
     program to work.)

1000 REM  ** MHRG DEMO #1
1010 POKE 53281, 0: POKE 53280, 0
1020 POKE 49172,(16*2)+6: SYS 49171
1030 SX = 40: EX = 119: CP = 1
1040 REM  ** RED STRIPE
1050 FOR Y = 81 TO 90: GOSUB 440
1060 NEXT Y: CP = 3
1070 REM  ** WHITE STRIPE
1080 FOR Y = 91 TO 100: GOSUB 440
1090 NEXT Y: CP = 2
1100 REM  ** BLUE STRIPE
1110 FOR Y = 101 TO 110: GOSUB 440
1120 NEXT Y
1130 REM  ** STARS IN BUNTING
1140 FOR X = 50 TO 110 STEP 10
1150 FOR Y = 85 TO 86
1160 CP = 3: GOSUB 190
1170 NEXT Y
1180 FOR Y = 105 TO 106: GOSUB 190
1190 NEXT Y
1200 NEXT X
1210 REM  ** OVAL OF STARS
1220 XC = 80: YC = 100: RC = 60
1230 FOR A = 0 TO 350 STEP 10
1240 R = A * (π / 180)
1250 X = INT(SIN(R) * RC) + XC
1260 Y = INT(COS(R) * RC) + YC
1270 GOSUB 190
1280 Y = Y + 1: GOSUB 190
1290 NEXT A
RUN
```

MHRG Demonstration #2

This subprogram uses the MHRG environment to create a four-color needlepoint pattern. It works an octagonal figure repeated in six rows by eleven columns, all identical, and then it goes back and changes two of the three colors in the figures that are in the second position inside the pattern. The program demonstrates:

1. Creation of a pattern image (lines 1050–1140) in an array using data statements (lines 5000–5110) that give the color pointer value for each pixel.
2. Use of a pair of loops (lines 1150–1200) and subroutine (lines 1400–1520) to place the pattern stored in the array at a specific location on the screen. Note that lines 1460–1470 work two bytes at a time, stacking one atop the other, in order to make the octagonal figure "square" (since a pixel is twice as wide as its height). Each figure is thus 24 × 24 dots, filling an area of 3 × 3 cells. The coordinates PX and PY specify the upper left cell of the figure and the subroutine builds down and right from that point.
3. Use of the environment's services to change colors in selected cells. Lines 1210 through 1390 repeatedly call two subroutines to change the colors of the cells in the inner border areas from blue, red, white, and black to red, green, white, and black.

```
  (NOTE: the MHRG environment must be in
     the computer's memory for this
     program to work.)

1010 REM  ** MHRG DEMO #2
1020 POKE 49172,(16*6)+2: SYS 49171
1030 POKE 53280, 0: POKE 53281, 0
1040 DIM A(12,3)
1050 FOR R = 1 TO 12
1060 FOR C = 1 TO 3
1070 P = 0
1080 FOR B = 3 TO 0 STEP -1
1090 READ N
1100 P = P + (N * (4^B))
```

```
1110 NEXT B
1120 A(R,C) = P
1130 NEXT C
1140 NEXT R
1150 REM  ** NEEDLEPOINT PATTERN
1160 FOR PX = 3 TO 33 STEP 3
1170 FOR PY = 3 TO 18 STEP 3
1180 GOSUB 1400
1190 NEXT PY
1200 NEXT PX
1210 REM  ** INNER BORDER - HORIZ
1220 FOR X = 6 TO 32
1230 FOR Y = 6 TO 8
1240 CN=2: GOSUB 290: CN=5: GOSUB 340
1250 NEXT Y
1260 FOR Y = 15 TO 17
1270 CN=2: GOSUB 290: CN=5: GOSUB 340
1280 NEXT Y
1290 NEXT X
1300 REM  ** INNER BORDER - VERT
1310 FOR Y = 9 TO 14
1320 FOR X = 6 TO 8
1330 CN=2: GOSUB 290: CN=5: GOSUB 340
1340 NEXT X
1350 FOR X = 30 TO 32
1360 CN=2: GOSUB 290: CN=5: GOSUB 340
1370 NEXT X
1380 NEXT Y
1390 GOTO 9950
1400 REM  ** DISPLAY PATTERN AT PX, PY
1410 CP=0: X=PX*4: Y=PY*8: GOSUB 190
1420 BL = BA: SR = 1
1430 FOR W = 1 TO 3
1440 FOR C = 1 TO 3
1450 FOR R = SR TO (SR+3)
1460 POKE BL, A(R,C): POKE BL+1, A(R,C)
1470 BL = BL + 2
1480 NEXT R
1490 NEXT C
```

(continued)

```
1500 BL = BL + 296: SR = SR + 4
1510 NEXT W
1520 RETURN
5000 DATA 0,0,0,1,1,1,1,1,1,0,0,0
5010 DATA 0,0,1,0,0,0,0,0,0,1,0,0
5020 DATA 0,1,2,2,2,2,2,2,2,2,1,0
5030 DATA 1,2,2,2,3,3,3,3,2,2,2,1
5040 DATA 1,2,2,3,0,0,0,0,3,2,2,1
5050 DATA 1,2,2,3,0,0,0,0,3,2,2,1
5060 DATA 1,2,2,3,0,0,0,0,3,2,2,1
5070 DATA 1,2,2,3,0,0,0,0,3,2,2,1
5080 DATA 1,2,2,2,3,3,3,3,2,2,2,1
5090 DATA 0,1,2,2,2,2,2,2,2,2,1,0
5100 DATA 0,0,1,2,2,2,2,2,2,1,0,0
5110 DATA 0,0,0,1,1,1,1,1,1,0,0,0
RUN
```

SECTION 7
COMPUTER GAMES

General

When Univac produced the first commercial computers back in the early 1950s, computer pioneers immediately began talking on the radio and to newspapers about playing tick-tack-toe with the machine. Today, perhaps one of the most widely distributed programs for large-scale IBM mainframes is the game StarTrek, which is illicitly played by programmers who are supposed to be doing productive things at their terminals (when I was a systems programming manager, my people thought they were fooling me; maybe I thought I was fooling them, too, when I played it at the terminal in my office). The advent of personal computers has made computer games more visible and created an industry for them, but it's hardly a surprising development. The computer is unquestionably the most captivating plaything ever invented.

The first program presented in this section is not really a game, but instead a "gambler's aid" that uses probability theory to assess the odds and maximum reasonable bet to place on throws of the

dice. The other three programs are games of skill that offer challenges to players of all ages.

Good luck.

Odds and Risks in Dice

To shoot dice is to play with probabilities, and all the blowing on them and pat chants in the world won't change that. A pair of dice can fall in any of 36 different combinations, the sum of which is somewhere between 2 and 12. The probability of the dice adding up to any particular sum is determined by how many possible combinations can equal that sum, divided by 36.

So how much money should you risk on a throw of the dice? That depends on two things: the probability of throwing the sum that will win the pot, and the amount you stand to gain.

This program calculates the odds, the probability, and the safe bet for any pot and winning throw.

```
NEW
100 REM  ** ODDS AND RISKS IN DICE
110 PRINT CHR$(147): NC = 0
120 PRINT "LIFE'S A CRAP SHOOT": PRINT
130 INPUT "HOW MUCH IS THE POT"; POT
140 IF POT = 0 THEN END
150 INPUT "WHAT DO YOU HAVE TO THROW"; T
160 PRINT: PRINT "COMBINATIONS:"
170 FOR D1 = 1 TO 6
180 FOR D2 = 1 TO 6
190 IF D1 + D2 <> T THEN 220
200 PRINT D1, D2
210 NC = NC + 1
220 NEXT D2
230 NEXT D1: PRINT
240 IF NC <> 0 THEN 270
250 PRINT "NONE": PRINT
260 PRINT "DON'T BET": GOTO 320
270 P = NC / 36
280 BET = INT((P * POT) * 100) / 100
290 PRINT "ODDS ARE" NC "IN 36"
```

```
300 PRINT "PROBABILITY =" P
310 PRINT "SAFE BET LIMIT IS $" BET
320 PRINT: PRINT: NC = 0
330 GOTO 130
RUN
```

```
Sample run:

        LIFE'S A CRAP SHOOT

        HOW MUCH IS THE POT? 50
        WHAT DO YOU HAVE TO THROW? 11

        COMBINATIONS:
         5          6
         6          5

        ODDS ARE 2 IN 36
        PROBABILITY = .055555556
        SAFE BET LIMIT IS $2.77


        HOW MUCH IS THE POT? 0

        READY.
```

Obstacle Course

This game is a variation of the rat in the maze. To play it, you need to connect a joystick to Control Port 2 (right end of the computer next to the power switch).

Here is the situation: The screen fills with 300 randomly placed pegs. You may increase or decrease the difficulty of the game by changing the value for NP in line 120; the more pegs, the stiffer the challenge. Each time you play, the pegs are arranged differently, since they're placed by the Commodore 64's random number generator. At the upper left corner appears a white rectangle. In

the lower right corner is a small enclosure with an open doorway. The object of the game is to move the rectangle into the enclosure using the joystick.

Rules of play: You may move the rectangle in any direction in order to negotiate the course and avoid obstacles. If the rectangle touches any obstacle, including the edge of the doorway to the enclosure, you lose. The game is won by moving the rectangle well inside the enclosure.

To replay the game after you've won or lost, press the fire button on the joystick. Press any key on the keyboard to clear the display and end the game entirely.

Good luck.

```
NEW
100 REM  ** OBSTACLE COURSE
110 DEF FNP(X) = INT(RND(1) * X)
120 NP = 300: SM = 1024: CM = 55296
130 PRINT CHR$(147): GOTO 190
140 REM  ** PLACE A PEG AT XY
150 P = X + (Y * 40)
160 POKE SM + P, 81
170 POKE CM + P, 0
180 RETURN
190 REM  ** SET UP SCREEN
200 POKE 53280, 0: POKE 53281, 2
210 FOR L = 1 TO NP
220 X = FNP(40): Y = FNP(25)
230 GOSUB 140
240 NEXT L
250 FOR X = 35 TO 39
260 FOR Y = 21 TO 24
270 P = X + (Y * 40)
280 POKE SM + P, 32
290 IF Y > 21 THEN 310
300 POKE SM + P, 119: POKE CM + P, 1
310 NEXT Y: NEXT X
320 POKE SM + 875, 118: POKE CM + 875, 1
330 POKE SM + 995, 118: POKE CM + 995, 1
340 REM  ** MAKE SPRITE
350 SL = 12288: SP = SL / 64
```

```
360 FOR P = 0 TO 63:POKE SL+P,0:NEXT P
370 FOR P = 0 TO 18 STEP 3
380 POKE SL + P, 254
390 NEXT P
400 SX = 24: SY = 50
410 POKE SM+1016, SP: POKE 53264, 0
420 POKE 53248, SX:   POKE 53249, SY
430 POKE SM, 32:      POKE 53287, 1
440 POKE 53269, 1:    X = PEEK(53279)
450 GOTO 520
460 REM  ** SPRITE POSITION
470 POKE 53264, INT(SX / 256)
480 IF SX > 255 THEN POKE 53248, SX-256:
 GOTO 500
490 POKE 53248, SX
500 POKE 53249, SY
510 RETURN
520 REM  ** MOVE SPRITE PER JOYSTICK
530 DV = 15 - (PEEK(56320) AND 15)
540 IF DV = 0 THEN 530
550 ON DV GOTO 560, 570, 530, 580, 590,
 600, 530, 610, 620, 630
560 SY = SY - 1: GOTO 640
570 SY = SY + 1: GOTO 640
580 SX = SX - 1: GOTO 640
590 SY = SY - 1: SX = SX - 1: GOTO 640
600 SY = SY + 1: SX = SX - 1: GOTO 640
610 SX = SX + 1: GOTO 640
620 SX = SX + 1: SY = SY - 1: GOTO 640
630 SX = SX + 1: SY = SY + 1
640 GOSUB 460
650 IF PEEK(53279) <> 0 THEN 690
660 IF SX < 320 THEN 520
670 IF SY < 210 THEN 520
680 PRINT: PRINT "YOU WON!!!": GOTO 700
690 PRINT: PRINT "TOO BAD, YOU LOSE"
700 REM  ** REPLAY OR QUIT
710 FB = 16 - (PEEK(56320) AND 16)
720 IF FB <> 0 THEN 100
```

(continued)

```
730 GET X$: IF X$ = "" THEN 700
740 POKE 53281, 0: POKE 53280, 0
750 POKE 53269, 0
760 PRINT CHR$(147)
770 END
RUN
```

Math Drill

This program is as much an educational tool as a game. It tests the player's ability to do the basic arithmetic operations: addition, subtraction, multiplication, and division. The numbers are all integers taken at random in the range −99 to +99. In division, the result is carried to three decimal places (both the computer's answer and the player's, so if the player enters more than three decimal places, the program ignores the extras).

Play this game as follows:

- Start the program by typing RUN.
- The computer displays the instructions.
- Press the RETURN key (or any other except "S") to begin the game.
- The computer poses a problem and asks you to type the answer. It then tells you whether you were right or, if you gave the wrong answer, what you should have typed.
- Press RETURN for the next problem, or "S" to stop.

The computer maintains the score at the top of the screen. It is updated each time you begin a new problem to reflect the current number of right and wrong answers you have given.

When you stop the game by typing "S", the computer tells the total number of right and wrong answers, the percentage score, and the "grade" (A through F). You flunk with fewer than 60 percent, and the grade rises by one letter for each 10 percent above that (60 percent and above is a D, 70 percent and above is a C, etc.).

Notes for teachers: Feel free to use this program in your classroom. It's a great exercise and the kids will love it. The program doesn't timeout waiting for an answer, so there's no pressure and the student can take as much time as necessary to figure out the answer on paper. For younger children you can simplify the problems by holding them to one-digit calculations of positive numbers with the following replacement line:

```
120  DEF FNR(X) = 9 - INT(RND(0)*9)
```

If you have not yet covered fractional quotients in division, you can eliminate division problems altogether with the following replacement line:

```
420  M = INT(RND(0) * 4)
```

Finally, to change the grading system to correspond with yours, replace the "cut points" in lines 930–960. For example, if you give an A for 95 percent or better, change the 90 in line 960 to 95.

Even those (of us) who aren't teachers will find this a useful exercise for brushing up rusty arithmetic skills.

```
NEW
100 REM  ** MATH DRILL
110 DEF FNA(X) = INT(X * 1000) / 1000
120 DEF FNR(X) = 100-INT(RND(0) * 200)
130 POKE 53281, 9: POKE 53280, 3
140 PRINT CHR$(147)
150 PRINT "MATH DRILL:": PRINT
160 PRINT "COMPUTER GIVES A PROBLEM.";
170 PRINT "  YOU TYPE THE"
180 PRINT "ANSWER. IF YOU ARE RIGHT,";
190 PRINT " THE COMPUTER"
200 PRINT "TELLS YOU SO, AND IF YOU";
210 PRINT " GIVE THE WRONG"
220 PRINT "ANSWER, THE COMPUTER TELLS";
230 PRINT " YOU WHAT THE"
240 PRINT "CORRECT ANSWER IS."
250 PRINT " AFTER EACH EXERCISE, TAP";
260 PRINT " 'RETURN' TO   CONTINUE OR";
```

(continued)

```
270 PRINT " THE LETTER 'S' TO STOP."
280 PRINT "THE COMPUTER WILL THEN ";
290 PRINT "GRADE YOUR PER-"
300 PRINT "FORMANCE."
310 PRINT " DECIMALS ARE CARRIED OUT ";
320 PRINT "TO 3 PLACES."
330 GOTO 550
340 REM ----------
350 REM  ** CONTINUE/STOP QUERY
360 PRINT "S TO STOP, RETURN FOR NEXT"
370 GET X$: IF X$ = "" THEN 370
380 IF X$ = "S" THEN 830
390 RETURN
400 REM ----------
410 REM  ** SELECT OPERATION
420 M = INT(RND(0) * 5)
430 IF M < 1 THEN 420
440 RETURN
450 REM ----------
460 REM  ** COMPUTATION
470 N1 = FNR(0): N2 = FNR(0)
480 ON M GOTO 490, 500, 510, 520
490 A = N1 + N2: GOTO 530
500 A = N1 - N2: GOTO 530
510 A = N1 * N2: GOTO 530
520 A = FNA(N1 / N2)
530 RETURN
540 REM ----------
550 REM  ** MAIN PROGRAM
560 C = 0: I = 0
570 PRINT: PRINT "TAP RETURN TO BEGIN"
580 GOSUB 370
590 REM  ** MAIN LOOP
600 PRINT CHR$(147) TAB(14);
610 PRINT "MATH DRILL:" CHR$(158)
620 PRINT "  RIGHT:" C; TAB(25);
630 PRINT "WRONG:" I; CHR$(5)
640 PRINT: PRINT
650 PRINT "PROBLEM:"
660 GOSUB 410: GOSUB 460
```

```
670 PRINT TAB(5) N1;
680 ON M GOTO 690, 700, 710, 720
690 PRINT " + ";: GOTO 730
700 PRINT " - ";: GOTO 730
710 PRINT " X ";: GOTO 730
720 PRINT " / ";
730 PRINT N2: PRINT: PRINT
740 INPUT "ANSWER"; Y: PRINT
750 Y = FNA(Y)
760 IF Y <> A THEN 790
770 PRINT "THAT'S RIGHT!"
780 C = C + 1: GOTO 810
790 PRINT "SORRY, THE ANSWER IS "; A
800 I = I + 1
810 PRINT: PRINT: GOSUB 350
820 GOTO 590
830 REM  ** GRADE PERFORMANCE
840 PRINT CHR$(147) TAB(14);
850 PRINT "YOUR SCORE:"
860 POKE 53281, 2: POKE 53280, 0
870 PRINT: PRINT: PRINT
880 PRINT "   NUMBER RIGHT:  " C
890 PRINT "   NUMBER WRONG:  " I
900 T = C + I: A = FNA(C / T * 100)
910 PRINT "   PERCENT RIGHT: " A
920 G$ = "F"
930 IF A >= 60 THEN G$ = "D"
940 IF A >= 70 THEN G$ = "C"
950 IF A >= 80 THEN G$ = "B"
960 IF A >= 90 THEN G$ = "A"
970 PRINT
980 PRINT "   YOUR GRADE IS " G$
990 PRINT: PRINT: PRINT: END
RUN
```

Shooting Gallery

You don't need a joystick to play this game.

Here's the situation: you are in a shooting gallery, and you have

a gun with a fixed aim. Targets move across your line of fire at different distances from the gun. Your task is to judge their speed relative to the speed of the bullet, so that you fire at the moment when the two will intercept and you'll score a hit. Fire the gun by pressing the SPACE bar. It's a game of timing.

You have ten bullets in each game. The top of the screen tells you how many hits you have scored and how many shots remain. After the last shot, the game lets you know how you did.

```
NEW
100 REM  ** SHOOTING GALLERY
110 PRINT CHR$(147) "SHOOTING GALLERY:"
120 PRINT"THIS IS A GAME OF SKILL."
130 PRINT"YOU HAVE 10 BULLETS TO FIRE"
140 PRINT"AT MOVING TARGETS. PRESS THE"
150 PRINT"SPACE BAR TO FIRE THE GUN."
160 PRINT: INPUT "YOUR NAME"; N$
170 GOSUB 300    :REM SET UP SCREEN
180 GOSUB 500    :REM PLAY THE GAME
190 PRINT CHR$(147) CHR$(5)
200 PRINT "GAME IS OVER": PRINT: PRINT
215 PRINT H "HITS": PRINT: PRINT
220 IF H >= 8 THEN W$ = "TERRIFIC"
230 IF H <  8 THEN W$ = "GOOD"
240 IF H <  5 THEN W$ = "FAIR"
250 IF H <  3 THEN S$ = "LOUSY"
260 PRINT "YOU'RE A " W$ "SHOT, " N$
270 POKE 53280, 0: POKE 53281, 0
280 POKE 53269, 0
290 END
295 REM ----------
300 REM  ** SET UP SCREEN
310 PRINT CHR$(147) CHR$(152)
320 POKE 53280, 8: POKE 53281, 0
330 FOR X = 29 TO 39: POKE X + 1504, 224
340 POKE X + 55776, 12: NEXT X
350 SL = 12288: SP = SL / 64
360 FOR X = 0 TO 63: POKE SL+X,0: NEXT X
370 FOR X = 0 TO 7: Y = X * 3
380 POKE SL + Y, 255: NEXT X
```

```
390 POKE 2040, SP:  POKE 2041, SP
400 POKE 53287, 2:  POKE 53288, 1
420 POKE 53248,255: POKE 53249, 146
420 POKE 53250, 0:  POKE 53251, 0
430 X = PEEK(53278)
440 POKE 53275, 255
450 POKE 53269, 3
460 RETURN
470 REM ----------
500 REM  ** PLAY THE GAME
510 H=0: P=10: S0=0: B=254: T=38: MI=4
520 DEF FNR(R)=INT(RND(0) * (R+1))
530 POKE 53249, 255: POKE 53249, 146
540 IF P = 0 THEN 750
550 REM  ** SET UP TARGET
560 DT=RND(0): GOSUB 800: X=PEEK(53278)
570 IF DT >= .5 THEN 590
580 S1 =  MI: S = T: F = B: GOTO 600
590 S1 = -MI: S = B: F = T
600 TX = (FNR(5) * 16) + 31
610 POKE 53250, TX: POKE 53251, S
620 REM  ** ADVANCE SPRITES
630 GET X$       :REM FIRING CHECK
640 IF X$<> "" THEN S0 = -MI*4: P=P-1
650 TY = PEEK(53251) + S1
660 IF TY = F THEN 530
670 POKE 53251, TY
680 PX = PEEK(53248)
690 IF PX < -S0 THEN S0 = 0: PX = 0
700 POKE 53248, PX + S0
710 IF PEEK(53278) = 0 THEN 620
720 REM  ** GOT A HIT
730 H = H + 1: S0 = 0
740 IF P <> 0 THEN 530
750 RETURN
800 REM  ** DISPLAY SCORE
810 PRINT CHR$(19);
820 PRINT TAB(10) "HITS" H;
```

(continued)

```
830 PRINT TAB(20) "  SHOTS LEFT ";
840 PRINT STR$(P) + "  "
850 RETURN
RUN
```

SECTION 8

SOUND EFFECTS AND COMPUTER MUSIC

General

The Commodore 64's synthesizer is capable of an endless variety of sounds, from the noises of nature to quite sophisticated music in three-part harmony. What I give here is but a sampling. Some of these sounds are merely interesting, and others can be adapted for inclusion in programs such as games. We have also used sounds in the clock programs for effects such as ticking and chiming the hour. Few computers offer such wide-ranging sound capabilities—and most offer none at all—with the result that software has heretofore been mostly silent. Not any more. Let your imagination take off after you try these programs and hear for yourself the auditory potential at your disposal.

One caution: these programs don't work at all if you fail to turn up the volume on your monitor. Whereas that might seem a statement of the obvious, I once spent a couple of hours vilifying the computer for its refusal to make even the simplest sound, only to discover that a simple twist of the knob was all it needed. Turn it up to the same volume you normally use when watching TV, and then enjoy the new magic of computer sound effects.

Breaking Surf

Those of us lucky enough to live on the beach (a Pacific one in my case) hear this sound all the time. It sounded the same when I lived near the Atlantic, too. Clearly, I have a love affair with the sea, and this is my way of sharing it with you no matter where you live.

You can stop the surf at any time by pressing the SPACE bar. The wave has to finish running its hissing foam up on the beach, but the program will remember that you asked it to stop and the sea will fall silent before the next comber breaks.

```
NEW
100 REM  ** BREAKING SURF
110 S = 54272: FQ = 12000: WF = 129
120 FOR X = 0 TO 24: POKE S+X,0: NEXT X
130 POKE S + 23, 241: POKE S + 24, 47
140 POKE S +  6, 240: POKE S +  4, WF
150 BF = FQ: TF = FQ * 2
160 SV = (TF - BF) / 128
170 FOR F = BF TO TF STEP SV
180 GOSUB 220    :REM   SURF
190 NEXT F
200 GET X$: IF X$ = "" THEN 170
210 GOTO 999
220 REM  ** SURF SOUND
230 HF = INT(F/256): LF = F-(HF * 256)
240 POKE S + 22, HF
250 POKE S +  1, HF: POKE S, LF
260 FOR T = 1 TO 40: NEXT T
270 RETURN
999 FOR X = 0 TO 24: POKE S+X,0: NEXT X
RUN
```

Emergency!

Here's a sound effect that will electrify anybody within earshot and send them to the windows looking for flashing lights. It repeats

full cycles until you tap any key, and then the siren winds down to silence. Use it by itself or build it into other programs, such as games.

```
NEW
100 REM  ** EMERGENCY!
110 S = 54272: FQ = 9000
120 FOR X = 0 TO 24: POKE S+X, 0: NEXT X
130 POKE S+24, 15: POKE S+6, 240
140 POKE S+ 4, 33
150 BF = FQ: TF = FQ * 2
160 SV = (TF - BF) / 128
170 FOR F = BF TO TF STEP SV
180 GOSUB 300
190 NEXT F
200 FOR F = TF TO BF STEP -SV
210 GOSUB 300
220 NEXT F
230 GET X$: IF X$ = "" THEN 170
240 FOR F = (BF - SV) TO 0 STEP -SV
250 GOSUB 300
260 NEXT F
270 FOR X = 0 TO 24: POKE S+X, 0: NEXT X
280 END
300 REM  ** SOUND OFF
310 HF = INT(F/256): LF = F - (HF*256)
320 POKE S, LF: POKE S+1, HF
330 FOR T = 1 TO 20: NEXT T
340 RETURN
RUN
```

Ringing Telephone

Here's one that will send the family scurrying to answer a phone call. It sounds just like the new electronic telephones, and not very different from the old standard (Ma) Bell. The phone continues to ring until you tap the SPACE bar, and then it stops.

```
NEW
100 REM  ** MA BELL
110 S = 54272: F1 = 10814: F2 = 8583
120 H1 = INT(F1/256): L1 = F1-(H1*256)
130 H2 = INT(F2/256): L2 = F2-(H2*256)
140 FOR X = 0 TO 24: POKE S=X,0: NEXT X
150 POKE S + 14, 165: POKE S + 15,  4
160 POKE S +  5,   9: POKE S + 24, 15
170 POKE S +  6, 240
180 REM  ** ONE RINGY-DINGY
190 FOR C = 1 TO 30
200 POKE S + 1, H1: POKE S, L1
210 POKE S + 4, 17
220 FOR T = 1 TO 20: NEXT T
230 POKE S + 1, H2: POKE S, L2
240 POKE S + 4, 17
250 NEXT C
260 POKE S + 4, 0
270 FOR T = 1 TO 2300: NEXT T
280 GET X$: IF X$ = "" THEN 180
290 FOR X = 0 TO 24: POKE S+X,0: NEXT X
300 END
RUN
```

R2D2

Everybody loves R2D2, that cute little robot from the Star Wars films. We can't understand what he says, but usually we know what he means. Even his sounds are cute, maybe because they remind us of how we used to think computers sounded before we learned that they're actually silent.

This program relies heavily on random numbers to generate not only the actual sounds, but the duration of R2D2's "speech." You can run this program a million times and it'll never be the same twice.

```
NEW
100 REM  ** R2D2
110 S = 54272
120 FOR X = 0 TO 24: POKE S+X,0: NEXT X
130 POKE S + 6, 240: POKE S + 24, 15
140 FOR N = 1 TO (RND(0) * 80)
150 HF = INT(RND(0) * 256)
160 LF = INT(RND(0) * 256)
170 POKE S + 1, HF: POKE S, LF
180 POKE S + 4, 17
190 FOR T = 1 TO 45: NEXT T
200 NEXT N
210 FOR X = 0 TO 24: POKE S+X,0: NEXT X
220 END
RUN
```

Instrument Sampler

This program imitates ("synthesizes") nine different musical instruments, all playing the C major scale. As each instrument performs, its name is listed on the screen. The purpose here is to give you a sampling of the computer's ability to mimic a wide variety of "natural" sounds realistically.

```
NEW
100 REM  ** INSTRUMENT SAMPLER
110 S = 54272: DIM N(8)
120 FOR X = 0 TO 24: POKE S+X,0: NEXT X
130 REM  ** BUILD SCALE
140 FOR X = 1 TO 8
150 READ N(X)
160 NEXT X
170 POKE S + 24, 15: POKE S + 2, 255
180 REM  ** INSTRUMENTS
190 FOR I = 1 TO 9
200 READ I$, W, A, D, U, R
210 PRINT I$
```

(continued)

```
220 POKE S + 5, (16 * A) + D
230 POKE S + 6, (16 * U) + R
240 FOR X = 1 TO 8
250 GOSUB 330
260 NEXT X
270 FOR X = 7 TO 1 STEP -1
280 GOSUB 330
290 NEXT X
300 NEXT I
310 FOR X = 0 TO 24: POKE S+X,0: NEXT X
320 END
330 REM  **  PLAY A NOTE
340 HF = INT(N(X) / 256)
350 LF = N(X) - (HF * 256)
360 POKE S + 1, HF: POKE S, LF
370 POKE S + 4, W
380 FOR T = 1 TO 150: NEXT T
390 POKE S + 4, W - 1
400 FOR T = 1 TO 80: NEXT T
410 RETURN
420 DATA 4291, 4817, 5407, 5728
430 DATA 6430, 7217, 8101, 8533
440 DATA "PIPE ORGAN",  17, 0, 0,15, 8
450 DATA "HORN",        33, 3, 6, 8, 0
460 DATA "ACCORDION",   17, 6, 6,15, 4
470 DATA "MARIMBA",     17, 0, 9, 0, 0
480 DATA "CALLIOPE",    17, 0, 0,15, 0
490 DATA "HARPSICHORD", 33, 0, 9, 0, 0
500 DATA "CLARINET",    17, 4, 8, 8, 0
510 DATA "GUITAR",      65, 0, 9, 0, 0
520 DATA "FLUTE",       17, 4,12,12, 4
RUN
```

Bugle Call

Afficionados of the sport of kings will feel their blood pressure rise at this bugle call. It's the signal for the start of a horse race. It also demonstrates music-making with one voice of the Commodore 64's synthesizer, set up to emulate a horn.

If the music is too fast—or too slow—to suit you, you can change the tempo by altering the value of B in line 110; the higher the value, the slower the bugler plays.

```
NEW
100 REM  ** THE SPORT OF KINGS
110 S = 54272: B = 25
120 FOR X = 0 TO 24: POKE S+X,0: NEXT X
130 POKE S +  5, 54: POKE S + 6, 128
140 POKE S + 24, 15
150 REM  ** BLOW THE BUGLE
160 READ L, H, D
170 IF D = 0 THEN 240
180 POKE S, L: POKE S + 1, H
190 POKE S + 4, 33
200 FOR T = 1 TO (D * B): NEXT T
210 POKE S + 4, 32
220 FOR T = 1 TO (D * B / 2): NEXT T
230 GOTO 150
240 REM  ** THEY'RE OFF!
250 FOR X = 0 TO 24: POKE S+X,0: NEXT X
260 END
1000 DATA  96,22,2, 223,29,2, 162,37,2
1010 DATA 193,44,2, 193,44,1, 193,44,1
1020 DATA 193,44,2, 162,37,2, 162,37,1
1030 DATA 162,37,1, 162,37,2, 223,29,2
1040 DATA 162,37,2, 223,29,2,  96,22,6
1050 DATA  96,22,2, 223,29,2, 162,37,2
1060 DATA 193,44,2, 193,44,1, 193,44,1
1070 DATA 193,44,2, 193,44,2, 162,37,2
1080 DATA 223,29,2,  96,22,2,  96,22,1
1090 DATA  96,22,1,  96,22,2, 223,29,2
1100 DATA 223,29,1, 223,29,1, 223,29,2
1110 DATA 223,29,9,   0, 0,0
RUN
```

Three-Part Harmony

The Commodore 64's three voices can act independently but in concert, much as the separate instruments in a "combo," to produce music of such quality that it's difficult to tell it was made by computer and not by human players. The following program, taken from *Mastering Sight and Sound on the Commodore 64,* is an example. In it, a "flute" plays the melody accompanied by a guitar. The composition is the familiar Melody in F by Anton Rubinstein, popularized in the song "Welcome, Sweet Springtime."

Three-part harmony is rather complex, and this leads to a correspondingly complex program. In fact, the concepts underlying the program are fairly simple, but their implementation in a computer leads to some exotic instructions. The trickiest part has to do with synchronization; one voice keeps playing a note while another plays a new note and the third falls silent, etc. This program is the basis for a general-purpose music maker that you can adapt for your own musical arrangements. The secret is in the DATA statements, which correspond to symbols and notation on a musical score; the book this program comes from explains how it works. Note that the "score" comprises over half the total length of the program listing. It's long, but after you type and run it, you'll no doubt be inspired to try some musicmaking of your own with your Commodore 64. It's quite a musical instrument in its own right.

```
NEW
100 REM  ** WELCOME, SWEET SPRINGTIME
110 DIM NN(3), N(3,4): CLR
120 BT = 20: RO = 3: S = 54272
130 FOR X = 0 TO 24: POKE S+X,0: NEXT X
140 POKE S+24, 12
150 N(1,4)=17: N(2,4)=65: N(3,4)=65
160 POKE S+5, (4 * 16) + 8
170 POKE S+6, (8 * 16) + 0
180 POKE S+9, 255: POKE S+12,9
190 POKE S+16,255: POKE S+19,9
200 REM  ** SET UP NEXT NOTE(S)
210 FOR X = 1 TO 3: NN(X) = 0: NEXT X
220 READ V, NV, N(V,3), A: NN(V) = 1
```

```
230 IF A < 0 THEN 590
240 IF NV = 0 THEN FQ = 0: GOTO 270
250 NP = NV + (RO * 12)
260 FQ = 2^(NP/12) * 268.234
270 N(V,2) = INT(FQ / 256)
275 N(V,1) = FQ - (N(V,2) * 256)
280 IF A = 0 THEN 220
290 REM  **  HOLD NOTES TO END OF BEAT
300 IF (TIME - T1) < D THEN 300
310 REM  **  PROCESS NEW NOTES
320 FOR V = 1 TO 3: VL = (V - 1) * 7
330 IF NN(V) = 0 THEN 370
340 POKE S + VL + 4, N(V,4) - 1
350 POKE S + VL + 0, N(V,1)
360 POKE S + VL + 1, N(V,2)
370 NEXT V
380 REM  **  DECR DURATION OF OLD NOTES
390 FOR V = 1 TO 3: VL = (V - 1) * 7
400 IF NN(V) <> 0 THEN 440
410 N(V,3) = N(V,3) - (D / BT)
420 IF N(V,3) > 0 THEN 440
430 POKE S + VL + 4, N(V,4) - 1
440 NEXT V
450 REM  **  START NEW NOTES
460 FOR V = 1 TO 3: VL = (V - 1) * 7
470 IF NN(V) = 0 THEN 490
480 POKE S + VL + 4, N(V,4)
490 NEXT V
500 REM  **  GET TIME FOR TEMPO
510 T1 = TIME
520 REM  **  FIND SHORTEST DURATION
530 D = 999
540 FOR V = 1 TO 3
550 IF N(V,3) <= 0 THEN 570
560 IF N(V,3) < D THEN D = N(V,3)
570 NEXT V
580 D = D * BT: GOTO 200
590 REM  **  STOP THE MUSIC
```

(continued)

```
600 IF (TIME - T1) < D THEN 600
610 FOR X = 0 TO 24: POKE S+X,0: NEXT X
620 END
999 REM  ----------
1000 DATA 1,12,2,0, 2,0,1,9
1001 DATA 2,9,1,0,  3,5,1,9
1002 DATA 1,11,1,9
1003 DATA 1,12,1,9
1004 DATA 1,12,2,0, 2,0,1,9
1005 DATA 2,10,1,0, 3,7,1,9
1006 DATA 1,11,1,9
1007 DATA 1,12,1,9
1008 DATA 1,17,2,0, 2,0,1,9
1009 DATA 2,9,1,9
1010 DATA 1,16,1,9
1011 DATA 1,17,1,9
1012 DATA 1,24,4,0, 2,0,1,9
1013 DATA 2,6,1,9
1014 DATA 2,15,1,9
1015 DATA 2,6,1,9
1016 DATA 1,22,2,0, 2,0,1,9
1017 DATA 2,7,1,9
1018 DATA 1,21,1,0, 2,9,1,9
1019 DATA 1,19,1,0, 2,10,1,9
1020 DATA 1,21,2,0, 2,12,1,9
1021 DATA 2,16,1,9
1022 DATA 1,19,2,0, 2,10,1,9
1023 DATA 2,16,1,9
1024 DATA 1,17,2,0, 2,0,1,9
1025 DATA 2,9,1,9
1026 DATA 1,16,1,0, 2,8,1,9
1027 DATA 1,14,1,0, 2,11,1,9
1028 DATA 1,16,2,0, 2,7,1,9
1029 DATA 2,10,1,9
1030 DATA 1,14,2,0, 2,4,1,9
1031 DATA 2,10,1,9
1032 DATA 1,12,2,0, 2,0,1,9
1033 DATA 2,9,1,0, 3,5,1,9
1034 DATA 1,11,1,9
1035 DATA 1,12,1,9
```

```
1036 DATA 1,12,2,0, 2,0,1,9
1037 DATA 2,10,1,0, 3,7,1,9
1038 DATA 1,11,1,9
1039 DATA 1,12,1,9
1040 DATA 1,17,2,0, 2,0,1,9
1041 DATA 2,9,1,9
1042 DATA 1,16,1,9
1043 DATA 1,17,1,9
1044 DATA 1,26,2,0, 2,0,1,9
1045 DATA 2,6,1,9
1046 DATA 1,24,2,0, 2,15,1,9
1047 DATA 2,6,1,9
1048 DATA 1,22,2,0, 2,0,1,9
1049 DATA 2,7,1,9
1050 DATA 1,18,1,0, 2,9,1,9
1051 DATA 1,19,1,0, 2,10,1,9
1052 DATA 1,21,2,0, 2,12,1,9
1053 DATA 2,16,1,9
1054 DATA 1,19,2,0, 2,10,1,9
1055 DATA 2,16,1,9
1056 DATA 1,17,4,0, 2,0,1,9
1057 DATA 2,9,1,9
1058 DATA 2,12,1,9
1059 DATA 2,9,1,9
1060 DATA 1,29,2,0, 2,9,2,0, 3,5,2,9
9999 DATA 0,0,0,-1
RUN
```

SOME OF THE THINGS YOU CAN DO WITH KENT PORTER'S PROGRAMS— AND YOUR COMMODORE 64

FIGURE OUT THE REAL INTEREST YOU ARE PAYING WHEN YOU BUY ON THE INSTALLMENT PLAN OR TAKE OUT A LOAN

CREATE YOUR OWN SAVINGS PLAN TO REACH THE AMOUNT YOU SEEK BY A CERTAIN DATE

CALCULATE THE ODDS IN GAMES OF CHANCE

PRODUCE A WIDE RANGE OF AMAZINGLY REALISTIC SOUNDS

ALPHABETIZE LONG LISTS

PLAY YOUR OWN VIDEO GAMES

TURN YOUR COMPUTER INTO A DIGITAL TIMEPIECE AND CALENDAR

USE THE COLOR CAPACITY OF THE COMMODORE 64

EMPLOY NUMEROUS SUBROUTINES TO IMPROVE THE QUALITY OF YOUR PROGRAMS

AND MUCH MORE!

WITH THE WIDE RANGE OF PROGRAMS INCLUDED IN THIS BOOK, YOU WILL NOT NEED TO BUY EXPENSIVE SOFTWARE!

PORTER'S PROGRAMS FOR THE COMMODORE 64

0 71162 00695 82090

ISBN 0-451-82090-8