SIGNET SPECIAL • 451-XE2088 • (CANADA $8.75) • U.S. $6.95

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

PORTER'S PROGRAMS FOR THE IBM® PCjr

FINANCIAL PLANNING
TIMEKEEPING
GAME CREATIONS
MATH MASTERY
GRAPHICS
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 IBM® PC*jr* 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 IBM PC*jr*'s capabilities. Many of these programs will also run on the IBM PC.

PORTER'S PROGRAMS FOR THE IBM® PC*JR*

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

PORTER'S PROGRAMS FOR THE IBM® PCJR

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

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, somone 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 IBM PC*jr*, but it's only a smattering of the possibilities. Not every program in it is for everybody; on the other hand, everybody will find something of value in it. For convenience, we've grouped the programs into broad categories such as math, money, measurements, and time. With a few exceptions that are noted as such, they're all standalone programs that you can 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 library of programs that put your computer to work doing useful and/or entertaining things for you. This book, then, is dedicated to people who don't quite know what to do with a computer or those who have one and want to do more with it.

The programs were all developed on, and therefore presuppose, the following configuration:

An IBM PC*jr* with 128K of memory
A color TV set as a monitor

One disk drive
IBM cartridge BASIC (by Microsoft Corp.)

One of the games also needs an IBM joystick, which is an inexpensive and fun addition to the system. Actually, many of these programs will also work on a Junior using cassette BASIC and a tape recorder. Most of these programs will also run on Junior's older brother, the IBM PC.

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 PC*jr*'s special features. For those things you have to turn to two other books that I also wrote: *Beginning with BASIC* (New York: NAL/Plume, 1984) and *Graphics and Music Programming on the IBM PCjr* (New York: NAL/Plume, forthcoming). I hasten to point out that programming skills and expertise in the technicalia of the computer are not requirements for using 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 them and you think they're a bunch of gobbledygook, they'll still work.

It is important, however, to understand the mechanics of working with the Junior. It has some features and commands that make it much more than a typewriter that prints on a screen. For that reason, the rest of this introduction is a tutorial that discusses the PC*jr* in general, diskettes, program input and editing, and the saving and fetching of programs on disk. If you already know about these things, skip the rest of this Introduction and we'll meet you among the programs.

Operating the IBM PC*jr*

As our point of departure on this journey into rudiments of working with the PC*jr*, we'll assume that you have hooked the parts together according to the instructions that came with the machine and inserted four AA batteries into the keyboard. Furthermore, we'll assume that you have never before touched a computer or even seen one in operation at close range.

Turn on the monitor (the TV set connected to the computer), and then power on the computer by operating the rocker switch at the left rear of the machine's housing. For a few moments you will see the IBM logo and a color bar on the screen. The machine beeps and the disk drive tests to see if there's a disk in it (there shouldn't be for this preliminary exercise). Then the screen clears and displays the message:

```
The IBM Personal Computer Basic
Version C1.20 Copyright IBM Corp 1981
62940 Bytes free
Ok
```

(The exact text of the message might be slightly different on your machine, but that doesn't matter.) The computer is now in cassette BASIC and ready to operate.

The first advice, before we try anything, is this: 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 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 the positive, quiet action of the keys. Press the ENTER key at the right end of the keyboard. The screen should show the letters you typed with the next lines saying "Syntax error" and "Ok." The random keys made no sense to the computer, and it said so with the "Syntax error" message. That is 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 there is absolutely no reason to fear the machine.

Now look at the screen again. Across the bottom are some words ("1LIST 2RUN" etc.) called the *key*. You don't need the key for this book, and since it's just in the way we'll get rid of it. Type the command

```
Key off
```

and press the ENTER button. The key instantly vanishes, and the computer responds "Ok." BASIC always says "Ok" when it is ready to accept another command.

Notice the flashing underscore. This is called the *cursor*, and it indicates where the next character you type will appear. The cursor roughly corresponds to the print mechanism on a typewriter. It should be under the last "Ok."

Customarily, BASIC programs are typed in upper-case letters. They don't have to be, but that's the convention, as the listings in this book illustrate. You can set the keyboard to all upper-case letters by pressing the CapsLock button to the right of the SPACE bar. Type some random letters and you'll see that they are all capitals. This is unlike the SHIFT LOCK on a typewriter in that it only shifts alphabetic letters. Type some numbers and they appear as numbers. To get the shifted number characters (such as the % sign, which is shifted 5), you have to hold down the SHIFT key; there's one at each end of the first full row of keys. The CapsLock mode on the PC*jr* also has another peculiarity. If you are in CapsLock and you hold down SHIFT when you type a letter, you get its lower-case equivalent.

Whenever you have typed a line of characters, you tell the computer that the line is complete by pressing the ENTER key. That's the only way the computer knows it's time to act on the command you've entered. A lot of computer novices fail to press ENTER and then wonder why the computer isn't doing anything. In fact it is doing something: It's waiting for you to finish the line.

You have a lot of garbage on the screen right now. You can get rid of it with a simple command. Type CLS and press ENTER. The screen clears and the cursor appears in the extreme upper-left-hand corner. You can always erase the display with CLS.

Oh, can't you see the cursor? There might be a little of it blinking up there on the edge. There's a simple solution: Hold down the buttons marked CTRL and ALT at the same time (lower left keyboard) and press the right-arrow key. It's one of the four keys clustered on the right end of the keyboard. The effect of this operation is to shift the whole display to the right. You can also shift it to the left by holding down CTRL and ALT and operating the left-arrow key. Move it several positions each way to get the idea. Incidentally, you don't have to press ENTER to execute these operations.

Setting Up for Disk Operation

Right now the machine is in cassette BASIC, the mode it automatically enters when you power up without a disk in the drive. Chances are that if you have a disk drive in your Junior, you have a more powerful version of the language called cartridge BASIC, under which all of the programs in this book will run. We will discuss these two "built-in" BASICs in more detail presently. For now, it's important only to know that to use the cartridge version you have to power up the computer differently than we did it this time. And in order to do that, you first have to set up a working disk.

The care and feeding of floppy disks

A *floppy disk* (also known as a diskette or simply a disk) is a circle of magnetic material encased in a sealed envelope. Information is recorded on the disk as magnetic spots arranged in concentric circles called *tracks*. The *disk drive* reads and writes on the surface via an oval slot on each side of the envelope. There is also a small hole near the central hub opening. A hole in the recording surface passes this outer hole, and the disk drive detects it and uses it as a point of reference for synchronizing its read/write operations.

The IBM PC*jr* uses both sides of the floppy disk for the storage of programs and data files, and it records at a high density. For this reason, you need double-sided double-density (DSDD) diskettes for your machine. Typically, these are top-of-the-line diskettes that cost a little more. You can get by with less expensive disks rated for single sided and/or single density, but that's a false economy. So are generic (no brand name) disks. You're entrusting a lot of work and perhaps valuable records to this thing. Stick with name brands rated for DSDD storage. A generic disk gave me enough grief that I finally threw it away. Spend $5 instead of $4 and be certain.

The diskette surface is extremely sensitive to dirt, dust, skin oils, and other contaminants. *Never touch the recording surface* visible through the oval slots. It will ruin the disk permanently. Also, don't leave diskettes lying around (the way people do in computer stores, who should know better or at least set an example). When the disk is not in use, store it in the jacket that came with it, and house your diskettes in a box of some sort that will ward off dust and tobacco smoke.

Diskettes are not usable as delivered, except when they contain a software product. The diskettes you purchase for your own use in saving programs and files must first be prepared for the PC*jr*. This process is called *formatting*. It writes the housekeeping information needed by the computer to find and save data and programs on the disk.

Starting the computer with a disk

The disk-equipped PC*jr* comes with several manuals, one of them marked "DOS," pronounced as a word and meaning Disk Operating System. In the back of the manual is an envelope containing two diskettes. Take out the one that is *not* marked "Supplemental Programs." Holding it by the upper edge, remove it from the IBM jacket and insert it into the disk drive with the label up and the oval slot away from you. Turn the small handle on the upper-left-hand corner of the drive down so that it prevents you from withdrawing the diskette, and then turn the computer power off and back on.

The IBM logo reappears, and after a moment the computer beeps and the drive makes some "put-put" sounds. The screen announces that the "Current date is Tue 1-01-1980" and asks you to "Enter new date." Type today's date in the form MM-DD-YYYY, so that if it's October 25, 1984, you'll enter 10-25-1984. After you press ENTER, the computer reports the current time according to its system clock (which begins running when you turn the power on), and asks you to "Enter new time." You can type just the hour and minute separated by a colon, though the machine reports time as HH:MM:SS, with the hundredth of a second appearing as a decimal number afterward. The time 03:08:15.04, for example, means 15.04 seconds after 3:08. If the time and date are unimportant to you, you can simply respond to both queries by pressing ENTER. The computer will then think it's January 1, 1980, and the time will be measured from when the machine was last powered on. The display on the screen reports that you have signed on to

```
IBM Personal Computer DOS
Version 2.10 (C)Copyright IBM Corp 1981,
1982, 1983
```

The exact text of this message may vary slightly.

The prompt "A>" appears under the signon messages. This is the indicator that DOS is ready to accept a command. It is comparable to the "Ok" we saw in BASIC.

Preparing a working diskette

Type the command

```
FORMAT
```

and press the ENTER key. The computer responds with

```
Insert new diskette for drive A:
and strike any key when ready
```

Twist the lever on the disk drive back to its original position, and remove the DOS disk, carefully placing it back in its jacket. Now insert a new disk into the drive, close the lever, and press ENTER or the SPACE bar. The computer responds

```
Formatting . . .
```

and the drive goes to work. The formatting process takes about a full minute before the computer reports that it has completed the job. It also tells you the total amount of space on the disk and the available space (362,496 bytes in both cases), and asks

```
Format another (Y/N)?
```

Type Y for yes, N for no. If you type Y, the process repeats and you can insert another new disk, then another, and so on until you have formatted all your diskettes. It's a good idea to format the whole batch as soon as you buy diskettes, so that you can be assured that all diskettes you possess are machine ready. When you reply N to the query, DOS displays the "A>" prompt to tell you it's ready for the next command.

Your working disk has to contain DOS itself and all the command files you'll need to operate the computer. In other words, it has to be a duplicate of the DOS disk you used to begin formatting. Place the original DOS disk from the manual back into the drive and type the command

```
DISKCOPY A: B:
```

which means copy all the files from one disk to another. The original DOS disk will be referred to during the transfer process as the *source diskette* and the new working disk you're creating as the *target diskette*. It will be necessary to swap disks in the drive several times during DISKCOPY in response to messages from the computer such as

```
Insert source diskette in drive A:
Strike any key when ready
```

This means you should place the indicated diskette into the drive, close the lever, and only then press a key to resume the copying.

When the entire DOS delivery disk has been copied to your working diskette, this message appears:

```
Copy complete
Copy another (Y/N)?
```

For the BASIC working disk you need for this book, you need make only one copy, so type N.

Now—and this is important—put the original DOS disk back into its jacket and file it away in the DOS binder. *Never use the delivery diskette for any purpose except to make working copies!* Diskettes are subject to wear, tear, and deterioration, and if you wear out your original you won't get much sympathy from anybody. You also won't be able to use your computer until you obtain a replacement.

All diskettes look alike, so it's important to label the disks you set up. Take a sticky-back label and write

DOS WORKING MASTER

on it. Then attach it to the newly created diskette in a position corresponding to the location of the label on the original. Be careful not to touch the magnetic recording surface and to keep the label from interfering with any openings in the disk cover.

From now on, whenever you do any BASIC programming, you will use this working master to start the computer, as described next.

Starting Cartridge Basic

With your computer you received a cartridge—a black object a little smaller than a package of cigarettes—bearing a brown label that says "cartridge BASIC" on it. Turn off the computer and insert this cartridge, label up, in one of the two slots directly under the disk drive on the front (it doesn't matter which slot you use).

To start up the computer with cartridge BASIC, do the following:

1. Turn on the monitor.
2. Insert your DOS WORKING MASTER into the disk drive, slotted end first with the label up.
3. Close the lever on the disk drive.
4. Turn on the computer.
5. Furnish the current date and time in response to the DOS queries (or press ENTER twice to bypass the calendar/clock update).
6. When you see the "A>" prompt, type BASIC.

In cartridge BASIC, you get an initial screen similar to that for cassette BASIC, except the version number begins with the letter "J" (instead of "C") and you have about 3000 fewer bytes of free memory. This difference is taken up by the expanded features of cartridge BASIC.

BASIC program diskette

You will need a second diskette for your BASIC programs, since the working master hasn't enough free space to contain more than a few programs. Take a freshly formatted disk and label it

PORTER'S PROGRAM FOR THE IBM PC*JR*

so that you'll know what it contains.

Once you've gotten into BASIC using the procedure described above, you can remove the working master from the disk drive and file it in its jacket; then place the program disk into the drive. All the programs in this book will be saved on and loaded from this diskette.

Cassette versus cartridge BASIC

Cassette BASIC comes built into every Junior and, as we have seen, it comes up automatically when there is no diskette in the drive (or no drive on the computer). Cassette BASIC is suitable for elementary programming, as well as including many advanced features, but it has some serious limitations. For one thing, it does not work with a disk drive but instead, as the name suggests, with a cassette tape unit. Also, cassette BASIC does not work with DOS, thus denying your programs many important services available only through the operating system. Finally, some of the more exotic instructions for graphics and sound either don't work at all in cassette BASIC or work only to a limited extent. With cassette BASIC, the PC*jr* is on a par with low-end machines such as the Commodore 64 and Atari systems.

Cartridge BASIC, on the other hand, is one of the most extensive and powerful BASIC programming systems available. It makes full use of DOS, handles multiple disk drives and hard disk storage, and performs marvels of graphics and sound. It raises the PC*jr* to a level nearly equivalent to its big brother, the IBM PC. Like all interpreted BASICs, it is rather slow about some operations, but there is little it cannot do within a reasonable time. Cartridge BASIC is a "superset" of the industry-standard Microsoft BASIC (and is a product of Microsoft), meaning that it contains all the nuclear instructions of Microsoft BASIC plus some that are peculiar to this computer and its features. In my experience, which spans several dozen computers and programming languages, it is one of the best.

Conversing with the Computer

The PC*jr* accepts four kinds of entries from the keyboard: control button operations (CTRL/ALT/cursor right to shift the display), 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, as you will see later in this book. The control button operations play a part in editing, and we'll discuss them presently.

A *command* is an instruction that you expect the computer to act on immediately. The KEY OFF and CLS commands we entered earlier 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 disk
LOAD	Reads a program from disk

There are others, 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"
```

(Don't forget to press ENTER!) 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 it to you by typing.

```
PRINT 15.25 + 1Ø91 + 66.81
```

As soon as you press ENTER, it displays the answer (1173.Ø6).

The arithmetic operators in BASIC are

+	Addition
–	Subtraction
*	Multiplication
/	Normal division
DIV	Integer division
MOD	Remainder in division

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

```
PRINT 15.25 × 1Ø91 / 66.81
```

and the answer 249.Ø3Ø834 instantly appears. The PRINT instruction tells the computer to do whatever follows and to 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 prompt "Ok." This indicates that it is ready to accept your next bidding. When it doesn't understand the command, it says

```
Syntax error
Ok
```

and that means "Huh?"

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

```
KEY OFF: CLS
```

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

The screen of the IBM PC*jr* 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 in 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 Junior is up to 80 characters or, in other words, two full rows. A few program lines here and there in this book spill onto a second row. The listings show them as being on one line even though they don't look that way on the screen. But now that you're warned you won't be dismayed when it happens.

Single-line commands are good for getting quick results to simple actions, but clearly they won't do for complex processes involving numerous calculations, 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 PC*jr* recognizes a program line by the fact that it has a number in front of it. An example is the "HI, THERE" command we typed. The command itself reads

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

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

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

When you press ENTER, 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.

Type the CLS command. Whoops! The screen went blank. Let's suppose we can't remember if 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 like before, as a result of the program running.

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

```
90 CLS
110 PRINT
120 PRINT "I'M A DUMB OLD COMPUTER"
RUN
```

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

```
HI, THERE

I'M A DUMB OLD COMPUTER
Ok
```

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

```
90 CLS
100 PRINT, "HI, THERE"
110 PRINT
120 PRINT "I'M A DUMB OLD COMPUTER"
```

Note that the program includes line 100. You'll recall that you typed line 100 and ran it a few times as a one-liner before typing 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 9Ø in front of the existing line 1ØØ and lines 11Ø and 12Ø after line 1ØØ 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 11Ø by listing the program. Whereas before line 11Ø 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 ENTER. Type 11Ø and hit ENTER, then run and list the program again. The "LET ME . . ." is gone, and so is line 11Ø, 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 rather when the computer encounters it as the program runs. To illustrate what happens, enter the following meaningless instruction into our little program:

```
11Ø SKLRGX
```

The computer accepts it initially, when you press ENTER, because 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 11Ø
```

```
Ok
110 SKLRGX
```

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 displays the offending line with the cursor in its first position. This enables you to correct the error immediately.

BASIC issues other more specific messages. For example, it is mathematically impossible to divide by zero. The computer can diagnose this problem. Therefore, if it encounters a statement such as

```
PRINT 90 / 0
```

it tells you what's wrong. Press ENTER to get the cursor off line 110 and then type this command to see for yourself. It says

```
Division by zero
1.701412E+38
```

The number is *machine infinity*, the largest possible number the computer can represent and therefore the closest it can come to actual infinity.

Bring the program back onto the screen with the LIST command. It still has the bad instruction at line 110, and we'll use it to experiment with on-screen editing.

Adjacent to the ENTER button are four keys marked with arrows pointing north, south, east, and west. Their 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. Like all keys on the Junior, these have an auto-repeat action, meaning that if you hold it down its action repeats until you release it. 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 or several with a single keystroke.

Use the up-arrow key to move the cursor to line 110 and the right arrow to position it on the "S" in "SLKRGX." To replace the present instruction with a PRINT, simply type the word PRINT. The new letters replace those on the screen as you type them. The trouble is that we now have "PRINTX," with the "X" left over from the previous contents of the line. The simplest way to get rid of the extraneous letter is to type a space, thus making the PRINT correct. Press ENTER and the change is made. We'll talk about other on-screen editing operations in a few minutes.

Press the down-arrow key so that it auto-repeats and moves down the left edge of the screen to the bottom, and then watch what happens. It seems to stall there, neither moving nor doing anything but flickering. This is because the cursor controls only operate within the screen boundaries. Press the left arrow a few times and you'll see the cursor wrap backward to the end of the line above. A few pushes on the right-arrow key moves it back to the start of the bottom row.

Now press the ENTER key and watch what happens to the text at the top of the screen. It moves upward one row each time you strike ENTER. Since the cursor can't move off the bottom of the screen, it pushes the rest of the display upward with a phenomenon called *scrolling*. Scrolling is similar to the paper advancing upward through a typewriter with one big difference: You can't wind the screen back down. 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 the 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, it pushed the whole screen up and continued streaming symbols across the bottom. Eventually the screen was full of plus signs, up to 1000 of them, which is the capacity of the PC*jr*'s 64 display. It kept on for another 1000 (for a total of 2000), but the scrolling was no longer apparent since each top line was replaced by another line identical to it. For an example of scrolling a more meaningful output, try this command:

```
FOR L=1 to 200: PRINT L,: NEXT L
```

With this command you get a list of all the numbers between 1 and 200, arranged in two tabular columns of 100 lines of output. 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:

```
90 CLS
100 PRINT "HI, THERE"
110 PRINT
120 PRINT "I'M A DUMB OLD COMPUTER"
```

Let's say we want to change the world OLD in line 120 to NEW. To do this, use the arrow keys to position the cursor at the "O" in OLD. Type the word "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 ENTER. (If you used the arrow keys to move the cursor to a different row without first hitting ENTER, the computer would disregard the change you made in line 120.) The cursor is now one row below line 120. Run the program by typing the RUN command. You can see the effect of the change in line 120, because the program now introduces itself as "A DUMB NEW COMPUTER."

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 give it a chance to defend itself against self-slander by inserting a word into line 120. Type LIST to see the program again. Then move the cursor up to line 120 and to the right until it's on the "A" after the contraction I'M. (If you overshoot the mark, use the left-arrow key to go back to the correct position.) Now press the INSERT key, abbreviated INS and located two keys to the right of the SPACE bar. Notice that the cursor becomes larger as soon as you do so. This is to remind you that you are in Insert—rather than normal text—mode. Type the word NOT and watch what happens to the existing text each time you type a letter. Starting with "A," everything shifts right to make room for the new letter, and the cursor remains with the "A." That is because it indicates where the next inserted character will appear. The line should now read

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

We need a space between "NOT" and "A," so press the SPACE bar. As before, you have to make the change official by pressing ENTER. To see the effect of the change, type RUN. The computer now claims that it is not dumb.

We've seen how to make replacements and insertions with the cursor 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 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. In addition, 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 either 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, 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
```

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

The original program that we're extracting from doesn't have to be on the screen as it is here. Were that the case, the LIST extensions wouldn't be very useful. Clear the screen with the CLS command so that the program is no longer in view, and then get line 12Ø with the command.

```
    LIST 12Ø
```

It appears near the top of the screen, followed by "Ok."

As long as a line is on the screen, you can change it with the 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 to the space after the "W" in "NEW." Press the BACKSPACE key (above ENTER) one time, and watch what it does. It deletes the character to the left of the cursor—and 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 striking INS and then typing a "B." You can continue editing this line for the rest of the week, and none of the changes become effective until you press ENTER. Do that, then 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 they'll soon become second nature to you. If you mangle a line beyond redemption, you can always bail out by pressing ENTER, 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 Disk

You probably won't want to save every program you ever type, whether it is from this book or another book or written by yourself. Usually, however, you will save programs on disk so that you can call them back later and run them.

Each program on a disk is saved in a *file*, which is a unit of storage that is self-contained and known by a name. You can think of a file on disk as being analogous to a chapter in a book: It has an identifying title, a location, and takes as much space as it needs. Obviously, to look up a chapter in a particular book, you need to open that book and not another. Similarly, to retrieve a program file from a disk, that diskette must be in the disk drive.

DOS takes care of managing disk space and the placement of program files for you, as long as you observe certain rules and use the proper commands.

Program file names

The first rule is that a program file must have a name. The name can consist of up to eight characters (letters or numbers), with a letter in the first position. These are proper program file names:

BUZZER	R1234567
D	ZEBRA1

The following are *not* acceptable file names for the reasons given:

TRAINWHISTLE	Too many characters
3MICE	First character must be a letter
TOM&DICK	& is not a valid character

I have made the selection of program names easy for you by including, in the first line of each program in this book, a suggested name in parentheses. Exceptions are made when the first line is itself an acceptable file name, such as HOOP.

Another rule is that only one file of a given name can exist on a disk at a time. You cannot have two files named HOOP on the same diskette, because DOS would have no way of knowing which HOOP you meant. Thus, every file name on a given diskette must be unique. Although you can't have two files named HOOP on a diskette, it's okay to have HOOP1 and HOOP2, because each one is a unique name.

Ordinarily, DOS files have a suffix, which is an extension of up to three characters set apart from the main file name with a period. When you save BASIC program files, DOS automatically attaches the suffix ".BAS" to the file name. Consequently, when you type the command

```
SAVE "HOOP"
```

DOS writes it to disk with the file name "HOOP.BAS." As long as a program file has the ".BAS" suffix, you don't need to specify it when you save or fetch the program in BASIC. (If you create or work on the program with a text editor or word processor, however, you *do* have to specify the suffix.)

Saving a program on disk

Make sure your PORTER'S PROGRAMS disk, and not the DOS WORKING MASTER, is in the drive. We still have the little sample program in the computer, and we'll use it to demonstrate how to save a program on disk.

BASIC writes programs to disk storage in one of two formats. In *compressed format*, it squishes the program to make it smaller so that it uses minimal space on the disk. It's a bad idea, and here's why. If you later retrieve that program to make changes or fix *bugs* (errors that cause the program to malfunction), you cannot read some instructions because they've been changed to gibberish. Also, cosmetic spacing, loop indentions, and other tricks to make programs readable have all been removed, so that what you see is a mass of text and symbols that bears little resemblance to what you typed. The way to save the sample program in compressed format is with the command

```
SAVE "SAMPLE"
```

(*Note*: this program is so short and simple that the compressed format is identical to what was typed. This is not so in larger, more complex programs.) Don't use compressed format unless you enjoy frustration and annoyance.

The better way is to save programs in ASCII format. ASCII (pronounced "askey" and short for American Standard Code for Information Interchange) is the electronic alphabet the PC*jr* uses to represent your typing within its memory and circuitry. When you save a program in ASCII format, the computer writes it to disk exactly as it appears on the screen and later retrieves it as a faithful copy. It does not in any way tamper with your work. The command to save the sample file in ASCII format is

```
SAVE "SAMPLE",A
```

The only difference is the ",A" following the file name. This is a *parameter* that specifies ASCII. If you fail to include it in the command, the program gets compressed.

Suppose you just saved a program without the ",A" and then realized your mistake. As long as you haven't wiped the program out of memory (with a NEW command, a LOAD, or turning off the power), you can retype the same SAVE command *with* the ",A" and DOS will replace the compressed copy on disk with an ASCII copy.

Retrieving a program from disk

Turn the PC*jr* off with the power switch. This wipes the memory clean, thus assuring that the sample program no longer exists within the computer and proving that what we're about to do works. Now turn the machine back on and go through the signon procedure to get into BASIC (you'll have to put the DOS WORKING MASTER into the drive for start-up).

Replace the working master with the PORTER'S PROGRAMS diskette. If you do a LIST, you'll see that there is no program in memory at the moment.

The process of retrieving a program file from a disk is called *loading*, since it's analogous to loading a truck (the computer's memory) from a warehouse (the disk). As in the case of SAVE, you must specify the file name in double quotes. Happily, there are no options with LOAD as there are with SAVE, so to get the program back from disk, type the command

```
LOAD "SAMPLE"
```

The disk whirrs a moment and the computer says "Ok," meaning that the load is complete. Type the LIST command and—voilà! —the program reappears, identical to the one we had before we powered down. It has been fetched from the disk and restored in toto. You can run it, edit it, do anything to it that you could before.

Program updates

If you don't make any changes to the program, it is not necessary to resave it. A copy already exists on disk, and the LOAD operation doesn't affect it, just as the words on this page are unaffected by your having read them.

On the other hand, if you do modify the program and you want to save it as a new version, you can use the same file name in the SAVE command and DOS will replace the old copy on disk with the update. To save the new version without affecting the old, give the new one a different name. For example, an update to the base program SAMPLE might have the name SAMPLE1, a subsequent version SAMPLE2, and so on.

Loading and running a program on disk

You can take a shortcut when you want to fetch a program that's on disk and run it immediately. As an illustration, type the commands

```
NEW: LIST
```

which delete SAMPLE from memory and show that there is no program currently in the computer.

Now type the command

```
RUN "SAMPLE"
```

and you'll see that BASIC loads the program from disk and executes it right away. Afterward, type LIST and the program lines appear on the screen, since they remain in memory after execution ends. The RUN "SAMPLE" command is identical in effect to

```
LOAD "SAMPLE"
RUN
```

and merely abbreviates the process of starting a program from the disk.

Finding out what's on a disk

Sometimes you don't know what files you have on a particular diskette. To find out, type the BASIC command

```
FILES
```

The computer then searches an area on the disk called the *directory* and lists the names of all the files it contains. It also tells you how much space is left for storage on the diskette. In this case, the display should read

```
FILES
A:
SAMPLE .BAS
  361472 Bytes free
```

When you have many files on the diskette, their names will appear in two columns. As the screen fills, the computer will pause briefly to give you time to scan the list and will then resume until all the file names have been displayed.

The FILES command has no effect on any program lines currently in memory. Type LIST and you'll see that the sample program is still there.

Removing a file from the disk

There's no point in keeping a copy of the sample program, so we'll use it as a case study in deleting files from diskettes. In IBM/Microsoft nomenclature, the act of removing a file from storage is called *killing*, a distressing term that nonetheless accurately conveys what happens. When you kill a file, you remove all traces of it from the disk, as though it had never existed. The space formerly occupied by the file is made available for other files that you might add later.

To delete the SAMPLE program from the disk, type

```
KILL "SAMPLE.BAS"
```

Note that it *is* necessary in this case to include the ".BAS" suffix. If you simply type

```
KILL "SAMPLE"
```

the computer responds "File not found," which means no action has occurred. When the operation is successful, as in the first example, BASIC replies "Ok," indicating that the file was killed.

You can check to verify that it no longer exists with the FILE command. The computer will reply "File not found," in this case meaning that there are no files to find on the diskette or, in other words, that the diskette is blank. In fact this is so, since SAMPLE was the only program we saved, and it's gone now.

SAMPLE still exists in memory, however, as the LIST command proves. The KILL command only removed it from the diskette. We can also wipe it out of the computer's memory by typing NEW.

Much of this probably still seems mysterious and frightening, especially if the PC*jr* is your first computer. The point is that you have nothing to be afraid of. The computer is only a tool, and like any tool you have to learn to use it and gain the experience that will make you comfortable with it. I guarantee that after you've entered, run, and saved a few of the programs that follow, you'll begin to feel like a pro, and most of these new commands and operations will become automatic.

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 society of elites 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 IBM PC*jr*, a household-priced machine 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. As for its speed in doing the much-publicized ENIAC calculation . . . Well, enter the following brief program and see for yourself how Junior stacks up against the ENIAC.

```
NEW
100 ' ENIAC
110 T = TIMER
120 X = EXP(LOG(2589) * 16 / 3)
130 E = TIMER - T
```

```
140 PRINT "CUBE ROOT OF 2589 TO 16TH POWER =" X
150 PRINT USING "TIME #.## SEC"; E
160 END
RUN

Sample run:

        CUBE ROOT OF 2589 TO 16TH POWER = 1.597236E+18
        TIME 0.05 SEC
        Ok
```

"A fraction of a second" indeed! Junior does it in a mere 0.05 (5/100) second. The government didn't say how many fractions of a second, but surely this diminutive 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 Junior 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 1.597236×10^{18}. Still doesn't make sense to you? It means shift the decimal point 18 positions to the right, giving the number

1,597,236,000,000,000,000

which is approximately the number of miles traveled by a ray of light in 272,300 years. The PC*jr*, able to produce up to seven digits at a time (in single-precision format, which most of the programs in this book use), 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 1.597236E–18 translates into

0.000000000000000001597236

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

Junior can represent shifts of up to ±37 decimal points in this fashion. An electron weighs 4.1×10^{-31} pounds, and the Earth weighs $1.70481048 \times 10^{13}$ pounds, both of which are well within the computer'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 Junior 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; for example, 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.

The following 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 ' FACTORS OF A NUMBER (FACTORS)
110 CLS: KEY OFF: 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
```

Sample run:

```
INTEGRAL FACTORS

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

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; therefore, 5 is prime. In a sense, the program to find primes 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 system over others.

There 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. First, 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.

```
NEW
100 ' ERATOSTHENES SIEVE (SIEVE)
110 CLS: KEY OFF: PRINT
120 PRINT "PRIME NUMBERS"
130 INPUT "  UP TO WHAT"; N
140 DIM P%(N)
150 PRINT: PRINT "THINKING"
160 T1 = TIMER
170 FOR X = 2 TO (N / 2)
180  FOR Y = 2 TO (N / 2)
190   I = X * Y
200   IF I > N THEN 230
210   P%(I) = 1
220  NEXT Y
230 NEXT X
240 T2 = TIMER
250 '
260 ' DISPLAY RESULTS
270 I = 0
280 FOR X = 1 TO N
290  IF P%(X) <> 0 THEN 340
300  PRINT X,
310  I = I + 1
320  IF I = 48 THEN GOSUB 390
330 NEXT X
340 ET = T2 - T1
350 PRINT
360 PRINT USING "TIME ###.## SEC"; ET
370 END
380 '
390 ' FULL SCREEN PAUSE
400 PRINT "PRESS RETURN FOR MORE..."
410 X$ = INKEY$
420 IF X$ = "" THEN 410
430 I = 0: RETURN
RUN
```

Sample run:

```
      PRIME NUMBERS
        UP TO WHAT? 100

      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
      TIME   4.23 SEC
      OK
```

Powers of a Number

Everybody knows that 3^2 means 3 squared or 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, that is, 3 × 3.

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 isn'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, as you'll see by running the following program.

```
NEW
100 ' POWER OF A NUMBER (POWER)
110 CLS: KEY OFF: PRINT
```

(continued)

```
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 31.54419

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

        ANSWER IS 3.170156E-02

         WHAT IS THE NUMBER ? 0
        Ok
```

Roots of Real Numbers

The last program concerned powers, and the next 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 although 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 180, 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 ' ROOT OF A NUMBER (ROOT)
110 CLS: KEY OFF: PRINT
120 N$ = "No can do"
130 PRINT "ROOT OF A REAL NUMBER:"
140 INPUT " WHAT IS THE NUMBER"; N
150 IF N = 0 THEN END
160 IF N < 0 THEN PRINT N$: GOTO 200
170 INPUT " FIND WHAT ROOT    "; R
180 A = EXP(LOG(N) / R)
190 PRINT "ANSWER IS" A
200 PRINT: GOTO 140
```

```
Sample run:

        ROOT OF A REAL NUMBER:
         WHAT IS THE NUMBER? 266.3
         FIND WHAT ROOT     ? 17
        ANSWER IS 1.388893

         WHAT IS THE NUMBER? 427.616
         FIND WHAT ROOT     ? -5.61
        ANSWER IS .3396303
```

(continued)

```
    WHAT IS THE NUMBER? 0
   OK
```

Trigonometry No. 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 aspect 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 the following three 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*, and so on. 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 ' TRIGONOMETRY #1 (TRIG1)
110 ' 2 SIDES AND INCLUDED ANGLE
120 CLS: KEY OFF: PI = 3.1416
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 * (PI / 180)
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 + PI
260 AA = PI - (AB + AC)
270 PRINT: PRINT
280 PRINT "SIDE C       " SC
290 PRINT "ANGLE A      " AA / (PI/180)
300 PRINT "ANGLE B      " AB / (PI/180)
310 PRINT: PRINT
320 GOTO 130
RUN
```

Sample run:

```
        TRIGONOMETRY #1:

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

        SIDE C        9.538046
        ANGLE B       35.67999
        ANGLE C       32.32

        TRIGONOMETRY #1:

          SIDE A     ? 0
        Ok
```

Trigonometry No. 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 ' TRIGONOMETRY #2 (TRIG2)
110 ' 2 ANGLES AND INCLUDED SIDE
120 CLS: KEY OFF: PI = 3.1416
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 * (PI / 180)
200 IF AB > 180 THEN AB = 360 - AB
210 AB = AB * (PI / 180)
220 AC = PI - (AA + AB)
230 SA = EXP(LOG(SC) - LOG(SIN(AC)) + LOG(SIN(AA)))
240 SB = EXP(LOG(SC) - LOG(SIN(AC)) + LOG(SIN(AB)))
250 PRINT: PRINT
260 PRINT "ANGLE C     " AC / (PI/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       6.00129
        SIDE B       5.501127

        TRIGONOMETRY #2:

          ANGLE A    ? 0
        Ok
```

Trigonometry No. 3

The third and last classical trigonometry problem we'll consider is that of a triangle in which the lengths of all three sides are known and we need to find the angles.

```
NEW
100 ' TRIGONOMETRY #3 (TRIG3)
110 REM  **  3 SIDES
120 CLS: KEY OFF: PI = 3.1416
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 = PI + 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 = PI + AB
260 AC = PI - (AA + AB)
270 PRINT "ANGLE A     " AA / (PI/180)
280 PRINT "ANGLE B     " AB / (PI/180)
290 PRINT "ANGLE C     " AC / (PI/180)
300 PRINT: PRINT
310 GOTO 130
RUN
```

Sample run:

```
        TRIGONOMETRY #3:

           SIDE A     ? 9.54
           SIDE B     ? 6
           SIDE C     ? 5.5
```

(continued)

```
        ANGLE A      112.0353
        ANGLE B      35.66103
        ANGLE C      32.30365

        TRIGONOMETRY #3:

          SIDE A     ? 0
        Ok
```

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, for example, $4! = 1 \times 2 \times 3 \times 4 = 24$. The following program computes any factorial up to 33!. Beyond that point, the product is too large a number for the computer to express. The answer repeats until you enter *0* in response to the prompt.

```
NEW
100 ' FACTORIAL! (FACTORIL)
110 CLS: KEY OFF
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
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? 18
 18 ! = 6402373705728000

FACTORIAL? 0
OK
```

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 it's a simple one 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 Ø point for horizontal measurements and a horizontal line representing the Ø 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; a point 5 units above the horizontal axis, has a Y coordinate of 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 and Y coordinates below the horizontal axis have negative numbers. 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 = –3, Y = –5. The program finds the distance between any two points so defined.

```
NEW
100 ' DISTANCE BETWEEN POINTS (DISTANCE)
110 CLS: KEY OFF
120 PRINT "DISTANCE BETWEEN 2 POINTS:"
130 INPUT "POINT #1 X, Y..."; X1, Y1
140 INPUT "POINT #2 X, Y..."; X2, Y2
150 D = SQR((X1 - X2)^2 + (Y1 - Y2)^2)
160 PRINT: PRINT "DISTANCE =" D
170 PRINT: PRINT
```

(continued)

```
180 INPUT "ANOTHER (Y/N)"; R$
190 IF R$ <> "Y" THEN END
200 PRINT: PRINT: GOTO 130
RUN
```

Sample run:

```
        DISTANCE BETWEEN 2 POINTS:
        POINT #1 X, Y...? 3,5
        POINT #2 X, Y...? -3,-5

        DISTANCE = 11.6619


        ANOTHER (Y/N)? Y


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

        DISTANCE = 5


        ANOTHER (Y/N)? N
        OK
```

Cones, Buckets, and Lampshades

Everyone knows what a cone is, of course; you eat ice cream from it, or drink water from it, 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 fairly 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 frusta 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 a lampshade and spread it flat, excluding 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.

The following program is geared toward frusta, but you can use it for cones and get valid results if you tell the program that one of the diameters is Ø.

```
NEW
100 ' CONES, BUCKETS & LAMPSHADES (CONES)
110 CLS: KEY OFF: PI = 3.1416
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  = PI * SH * (R1 + R2)
190 B1 = PI * R1^2
200 B2 = PI * R2^2
210 A  = S + B1 + B2
220 V = (PI*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
```

(*continued*)

```
        BOTTOM DIAMETER... ? 12
        TOP DIAMETER...... ? 7

        SLANT HEIGHT                12.25765
        LATERAL SURFACE             365.832
        BOTTOM SURFACE              113.0976
        TOP SURFACE                 38.4846
        TOTAL SURFACE               517.4142
        VOLUME                      870.2231
        Ok
```

Spheres

The following 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 ' SPHERE
110 CLS: KEY OFF: PI = 3.1416
120 PRINT "SPHERE CALCULATIONS"
130 PRINT: INPUT "DIAMETER... "; D
140 C = PI * D
150 A = D * C
160 V = (PI * (D^3)) / 6
170 PRINT
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               31.416
        SURFACE AREA                 314.16
        VOLUME                       523.6
        OK
```

Cylinders

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

```
NEW
 100 ' CYLINDER
 110 CLS: KEY OFF: PI = 3.1416
 120 PRINT "CYLINDER CALCULATIONS:"
 130 PRINT: INPUT "DIAMETER... "; D
 140 PRINT: INPUT "HEIGHT..... "; H
 150 C = PI * D: R = D / 2
 160 S = C * H
 170 A = PI * D * (H + R)
 180 B = (A - S) / 2
 190 V = PI * 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
RUN
```

Sample run:

```
        CYLINDER CALCULATIONS:

        DIAMETER... ? 3

        HEIGHT..... ? 10
```

(continued)

```
       CIRCUMFERENCE               9.4248
       LATERAL SURFACE             94.248
       TOTAL SURFACE               108.3852
       BASE AREA                   7.0686
       VOLUME                      70.686
       OK
```

Statistics No. 1

The next 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 No. 2 for that). You can enter the values in any order. The last value has to be 0, which tells the program that the list is completed.

```
NEW
100 ' STATISTICS #1 (STATS1)
110 L = 9999999: H = -L
120 CLS: KEY OFF
130 PRINT "STATISTICS #1:": PRINT
140 PRINT " ENTER VALUES ONE AT A TIME"
150 PRINT " END LIST WITH 0"
160 '
170 ' INPUT AND SUMMARIZE
180 INPUT "VALUE"; Q
190   IF Q = 0 THEN 270
200   IF Q < L THEN L = Q
210   IF Q > H THEN H = Q
220   N = N + 1         :' SAMPLE SIZE
230   T = T + Q         :' TOTAL
240   D = D + Q^2       :' SUM SQUARES
250 GOTO 170
260 '
270 ' CALCULATIONS
280 M = T / N           :' MEAN
290 V = (D / N) - M^2:' VARIANCE
300 S = SQR(V)          :' STD DEV
```

```
310 '
320 ' OUTPUT
330 PRINT "--------------------": PRINT
340 PRINT "SAMPLE SIZE          " N
350 PRINT "RANGE                " H - L
360 PRINT "   LOWEST      " L
370 PRINT "   HIGHEST     " H
380 PRINT "MEAN                 " M
390 PRINT "VARIANCE             " V
400 PRINT "STANDARD DEVIATION   " S
410 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:                9.049999
           LOWEST      .39
           HIGHEST     9.439999
           MEAN                   4.08
           VARIANCE               7.18817
           STANDARD DEVIATION     2.681076
           Ok
```

Statistics No. 2

The following program is more sophisticated than STATISTICS No. 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 (mpg), 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 mpg).

```
NEW
100 ' STATISTICS #2 (STATS2)
110 CLS: KEY OFF
120 PRINT "STATISTICS #2:": PRINT
130 PRINT " ENTER CLASS LOW, HIGH, OBSERVATIONS"
140 PRINT " END LIST WITH THREE 0'S"
150 '
160 ' INPUT AND SUMMARIZE
170 INPUT CL, CH, F
180   IF CL = 0 AND F = 0 THEN 250
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 '
250 ' CALCULATIONS
260 M = FX / N
270 V = (F2 - (N * M^2)) / N
280 S = SQR(V)
290 '
300 ' OUTPUT
310 PRINT "---------------------": PRINT
320 PRINT "SAMPLE SIZE          " N
330 PRINT "MEAN                 " M
340 PRINT "VARIANCE             " V
350 PRINT "STANDARD DEVIATION   " S
360 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              5.362422
STANDARD DEVIATION    2.315691
OK
```

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 as much as 5.36 mpg from the mean.

SECTION 2

WEIGHTS AND MEASURES

The next several programs will deal with the rather serious and perplexing problem 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, we see 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. This 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, within the English system, between the English and the metric systems, and within metric itself, although the latter is usually unnecessary. The metric system classifies measurements by types, which allows us to convert instantly from any unit to any other within the classification.

All of these programs follow the same general model, and only the units vary from one to another. We do some slick stuff with them on the screen, 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 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. If you're converting 1Ø kilometers to miles, type 1Ø. The program instantly replies

```
1Ø 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 will take over. Therefore, be sure to store them on disk where you can readily get to them.

Distance Measurements

This program converts among the following units:

Inches
Feet
Yards
Miles
Centimeters
Meters
Kilometers

```
NEW
100 ' DISTANCE CONVERSION (DISTCON)
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$(Y), C(0,Y)
180 NEXT Y
190 '
200 ' CONVERSION FACTORS GRID
210 FOR X = 1 TO D
220   FOR Y = 1 TO D
230     C(X,Y) = C(0,Y) / C(X,0)
240   NEXT Y
250 NEXT X
260 '
270 ' SCREEN SETUP
280 CLS: KEY OFF
290 PRINT "DISTANCE CONVERSION:"
300 FOR X = 1 TO D
310   PRINT TAB(5); X; N$(X)
320 NEXT X: PRINT
330 '
340 ' GET INPUTS
350 INPUT "CONVERT TO"; Y
360 LOCATE CSRLIN-1, 12: PRINT N$(Y)
370 PRINT: INPUT "CONVERT FROM"; X
```

```
380 LOCATE CSRLIN-1, 14: PRINT N$(X)
390 PRINT: PRINT "HOW MANY " N$(X);
400 INPUT U: PRINT
410 '
420 ' RESULTS
430 YU = U / C(X,Y)
440 PRINT U; N$(X); " = "; YU; N$(Y)
450 PRINT: PRINT
460 INPUT "ANOTHER (Y/N)"; X$
470 IF X$ = "Y" THEN 270
480 END
490 '
500 ' UNITS IN CENTIMETERS
510 DATA "INCHES",      2.54
520 DATA "FEET",        30.48
530 DATA "YARDS",       91.44
540 DATA "MILES",       160934.4
550 DATA "CENTIMETERS", 1
560 DATA "METERS",      100
570 DATA "KILOMETERS",  100000
RUN
```

Area Measurements

This program converts among the following units:

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

Do yourself a favor. Reload the last program from disk and change lines 100, 110, 140, 170, 290, and the DATA statements from 510 onward. They're the only differences between the two programs. Save the new program under the name AREACON.

```
NEW
100 ' AREA CONVERSION (AREACON)
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$(Y),C(0,Y):C(0,Y) = C(0,Y)^2
180 NEXT Y
190 '
200 ' CONVERSION FACTORS GRID
210 FOR X = 1 TO D
220   FOR Y = 1 TO D
230     C(X,Y) = C(0,Y) / C(X,0)
240   NEXT Y
250 NEXT X
260 '
270 ' SCREEN SETUP
280 CLS: KEY OFF
290 PRINT "AREA CONVERSION:"
300 FOR X = 1 TO D
310   PRINT TAB(5); X; N$(X)
320 NEXT X: PRINT
330 '
340 ' GET INPUTS
350 INPUT "CONVERT TO"; Y
360 LOCATE CSRLIN-1, 12: PRINT N$(Y)
370 PRINT: INPUT "CONVERT FROM"; X
380 LOCATE CSRLIN-1, 14: PRINT N$(X)
390 PRINT: PRINT "HOW MANY " N$(X);
400 INPUT U: PRINT
410 '
420 ' RESULTS
430 YU = U / C(X,Y)
440 PRINT U; N$(X); " = "; YU; N$(Y)
450 PRINT: PRINT
460 INPUT "ANOTHER (Y/N)"; X$
```

```
470 IF X$ = "Y" THEN 270
480 END
490 '
500 ' UNITS IN CENTIMETERS
510 DATA "SQ INCHES",         2.54
520 DATA "SQ FEET",           30.48
530 DATA "SQ YARDS",          91.44
540 DATA "ACRES",             6361.4907
550 DATA "SQ MILES",          160934.4
560 DATA "SQ CENTIMETERS",    1
570 DATA "SQ METERS",         100
580 DATA "HECTARES",          10000
590 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 typing by reloading the distance program and changing lines 100, 110, 290, and the DATA statements from 500 onward.

```
NEW
100 REM  ** WEIGHT CONVERSION (WGHTCON)
110 D = 8
120 DIM C(D,D), N$(D)
130 FOR X = 1 TO D
140   READ X$, C(X,0)
```

(continued)

```
150 NEXT X: RESTORE
160 FOR Y = 1 TO D
170   READ N$(Y), C(0,Y)
180 NEXT Y
190 '
200 '  CONVERSION FACTORS GRID
210 FOR X = 1 TO D
220   FOR Y = 1 TO D
230     C(X,Y) = C(0,Y) / C(X,0)
240   NEXT Y
250 NEXT X
260 '
270 ' SCREEN SETUP
280 CLS: KEY OFF
290 PRINT "WEIGHT CONVERSION:"
300 FOR X = 1 TO D
310   PRINT TAB(5); X; N$(X)
320 NEXT X: PRINT
330 '
340 ' GET INPUTS
350 INPUT "CONVERT TO"; Y
360 LOCATE CSRLIN-1, 12: PRINT N$(Y)
370 PRINT: INPUT "CONVERT FROM"; X
380 LOCATE CSRLIN-1, 14: PRINT N$(X)
390 PRINT: PRINT "HOW MANY " N$(X);
400 INPUT U: PRINT
410 '
420 ' RESULTS
430 YU = U / C(X,Y)
440 PRINT U; N$(X); " = "; YU; N$(Y)
450 PRINT: PRINT
460 INPUT "ANOTHER (Y/N)"; X$
470 IF X$ = "Y" THEN 270
480 END
490 '
500 ' UNITS IN GRAMS
510 DATA "OUNCES",       28.3495
520 DATA "POUNDS",       453.592
530 DATA "STONE",        6350.288
540 DATA "TONS",         907184
```

```
550 DATA "LONG TONS",   1016046.1
560 DATA "GRAMS",       1
570 DATA "KILOGRAMS",   1000
580 DATA "METRIC TONS", 1000000
RUN
```

Liquid Measurements

This is a really easy program to set up. Just reload the distance program and change lines 100, 290, and the DATA statements. You'll then have a program that converts among the following:

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

```
NEW
100 REM  **  LIQUID MEASUREMENTS (LIQMEAS)
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 '
200 ' CONVERSION FACTORS GRID
210 FOR X = 1 TO D
220   FOR Y = 1 TO D
230     C(X,Y) = C(0,Y) / C(X,0)
240   NEXT Y
250 NEXT X
```

(continued)

```
260 '
270 ' SCREEN SETUP
280 CLS: KEY OFF
290 PRINT "LIQUID MEASUREMENTS:"
300 FOR X = 1 TO D
310   PRINT TAB(5); X; N$(X)
320 NEXT X: PRINT
330 '
340 ' GET INPUTS
350 INPUT "CONVERT TO"; Y
360 LOCATE CSRLIN-1, 12: PRINT N$(Y)
357 PRINT: INPUT "CONVERT FROM"; X
380 LOCATE CSRLIN-1, 14: PRINT N$(X)
390 PRINT: PRINT "HOW MANY " N$(X);
400 INPUT U: PRINT
410 '
420 ' RESULTS
430 YU = U / C(X,Y)
440 PRINT U; N$(X); " = "; YU; N$(Y)
450 PRINT: PRINT
460 INPUT "ANOTHER (Y/N)"; X$
470 IF X$ = "Y" THEN 270
480 END
490 '
500 ' UNITS IN LITERS
510 DATA "CUPS",                .236575
520 DATA "PINTS",               .47315
530 DATA "QUARTS",              .9463
540 DATA "IMPERIAL QUARTS",     1.1365
550 DATA "GALLONS",             3.7852
560 DATA "IMPERIAL GALLONS",    4.546
570 DATA "LITERS",              1
RUN
```

Recipe Changer

There's an age-old problem of cooks: The recipe serves five people and you have eight coming to dinner. How much should you increase measurements such as 1⅓ cups to keep the right proportions? The following 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 people you plan to serve.

It's 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 cooking measurements don't require great precision.

The program keeps asking for quantities. 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 much to use based on the number you're serving. It knows the proportions based on the first two questions, which establish the size of the recipe and the size of your dinner party. When you've entered all the measurements and want the program to stop, type 0.

```
NEW
100 REM  **  RECIPE CHANGER (RECIPE)
110 CLS: KEY OFF
120 PRINT "RECIPE CHANGER:": PRINT
130 INPUT "NUMBER RECIPE SERVES "; RQ
140 INPUT "NUMBER YOU'RE SERVING"; SQ
150 CF = SQ / RQ
160 '
170 ' MAIN LOOP
180 PRINT: INPUT "QUANTITY"; Q$
190 IF Q$ = "0" THEN END
200 L = LEN(Q$): X = 1: SL = 1: Q = 0
210 FOR P = 1 TO L
220   IF MID$(Q$, P, 1) = "/" THEN 250
230 NEXT P
240 Q = VAL(Q$): GOTO 390
250 FOR P = 1 TO L
260   IF MID$(Q$, P, 1) = " " 290
270 NEXT P
280 GOTO 310
290 SL = P
```

(continued)

```
300 Q = VAL(LEFT$(Q$, P)): X = P + 1
310 FOR P = X TO L
320   IF MID$(Q$, P, 1) = "/" THEN 350
330 NEXT P
340 PRINT "BAD INPUT": GOTO 170
350 N = VAL(MID$(Q$, SL, P - SL))
360 D = VAL(RIGHT$(Q$, L - P))
370 F = N / D: Q = Q + F
380 '
390 ' FIGURE QUANTITY
400 Q = Q * CF: W = INT(Q)
410 F = Q - W: F$ = ""
420 IF F >= .95 THEN W = W + 1
430 PRINT "CHANGE TO" W;
440 '
450 ' FRACTION
460 IF F < .06 THEN 550
470 IF F < .95 THEN F$ = "7/8"
480 IF F < .82 THEN F$ = "3/4"
490 IF F < .7  THEN F$ = "2/3"
500 IF F < .6  THEN F$ = "1/2"
510 IF F < .4  THEN F$ = "1/3"
520 IF F < .3  THEN F$ = "1/4"
530 IF F < .18 THEN F$ = "1/8"
540 PRINT F$
550 PRINT: PRINT: GOTO 170
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
OK
```

Temperature Conversion

Back in the olden days they called it Centigrade, but nowadays it's Celsius. Someone, no doubt, knows why. At any rate, it's the metric way of measuring temperatures, whereby water freezes at 0°C and boils at 100°C, 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 conversion you want to make. Using it, you'll know whether to be hot or cold when the temperature is 40°C.

```
NEW
100 ' TEMPERATURE CONVERSION (TEMPCON)
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 '
150 ' CONVERSION LOOP
160 CLS: KEY OFF
170 PRINT "TEMPERATURE CONVERSION:"
180 PRINT
190 PRINT "   1  CELSIUS TO FAHRENHEIT"
200 PRINT "   2  FAHRENHEIT TO CELSIUS"
210 PRINT: INPUT "        WHICH"; X
220 PRINT: INPUT "        TEMPERATURE";T
230 PRINT: PRINT
240 IF X = 2 THEN 280
250 '
260 ' CEL TO FAHR
```

(continued)

```
270 PRINT T "DEGREES C =";
280 PRINT FNR(FNF(T)) "DEGREES F"
290 GOTO 340
300 '
310 ' FAHR - CEL
320 PRINT T "DEGREES F =";
330 PRINT FNR(FNC(T)) "DEGREES C"
340 PRINT: INPUT "ANOTHER (Y/N)"; X$
350 IF X$ = "Y" THEN 150
360 END
RUN
```

Sample run:

```
                    TEMPERATURE CONVERSION:

                      1  CELSIUS TO FAHRENHEIT
                      2  FAHRENHEIT TO CELSIUS

                           WHICH? 1

                           TEMPERATURE? 40

                     40 DEGREES C = 104 DEGREES F

                    ANOTHER (Y/N)? N
                    Ok
```

SECTION 3

SORTING AND TEXT PROCESSING

Sorting a Group of Numbers

Sorting is the process of rearranging unorganized information into an orderly sequence. The following 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 44 numbers before lines begin to scroll off the top; therefore, 44 is the limit of the sort size. When you enter 0, the program stops accepting numbers and begins sorting.

```
NEW
100 ' NUMERIC SORT (NUMSORT)
110 CLS: KEY OFF: SIZE = 43
120 PRINT "NUMERIC SORT": PRINT
130 PRINT "ENTER NUMBERS ONE AT A TIME"
140 PRINT "ENTER 0 TO START SORT"
150 DIM A(SIZE)
160 N = 0: MAX = 9999999
170 '
180 ' INPUT LOOP
190 INPUT A(N)
200 IF A(N) = 0 THEN 240
210 N = N + 1
220 IF N <= SIZE THEN 180
```

(continued)

```
230 '
240 ' SORT
250 PRINT: PRINT "SORTED LIST:"
260 L = MAX: M = 0
270 FOR P = 0 TO (N - 1)
280   IF A(P) < L THEN L = A(P): M = P
290 NEXT P
300 IF L = MAX THEN END
310 PRINT L,
320 A(M) = MAX
330 GOTO 260
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

        SORTED LIST:
         19           28
         37           46
         55           64
         73           82
         91
        Ok
```

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. The next program works the same in all other respects.

```
NEW
100 ' DESCENDING SORT (DESORT)
110 CLS: KEY OFF: SIZE = 43
120 PRINT "DESCENDING SORT": PRINT
130 PRINT "ENTER NUMBERS ONE AT A TIME"
140 PRINT "ENTER 0 TO START SORT"
150 DIM A(SIZE)
160 N = 0: MAX = -1
170 '
180 ' INPUT LOOP
190 INPUT A(N)
200 IF A(N) = 0 THEN 240
210 N = N + 1
220 IF N <= 87 THEN 180
230 '
240 ' SORT
250 PRINT: PRINT "SORTED LIST:"
260 L = -1: M = 0
270 FOR P = 0 TO (N - 1)
280 IF A(P) > L THEN L = A(P): M = P
290 NEXT P
300 IF L = -1 THEN END
310 PRINT L,
320 A(M) = -1
330 GOTO 260
RUN
```

Sample run:

(continued)

```
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
OK
```

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 (except that it doesn't report the number of SPACE characters, which you can override by removing line 280).

```
NEW
100 ' TEXT ANALYSIS (TEXTAN)
110 CLS: KEY OFF
120 DIM CH(255)
130 '
140 ' GET AND ANALYZE TEXT LINE
150 PRINT "ENTER A LINE OF TEXT"
```

```
160 LINE INPUT TX$: PRINT
170 LT = LEN(TX$)
180 FOR P = 1 TO LT
190   V = ASC (MID$ (TX$, P, 1))
200   CH(V) = CH(V) + 1
210 NEXT P
220 '
230 ' DISPLAY RESULTS
240 NW = CH(32) + 1
250 PRINT "CONTAINS" LT "CHARACTERS ";
260 PRINT "IN" NW "WORDS"
270 FOR P = 0 TO 255
280   IF P = 32 THEN 310
290   IF CH(P) = 0 THEN 310
300   PRINT TAB(8) CHR$(P), CH(P)
310 NEXT P
320 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
        Ok
```

Reversing Text

To see another example of text processing, type a line and the PC*jr* immediately displays it backward. This repeats each time you enter another line. Stop it by entering Ø.

```
NEW
100 ' TEXT REVERSER (TEXTREV)
110 CLS: KEY OFF
120 PRINT: PRINT "ENTER A LINE OF TEXT"
130 LINE INPUT X$
140 IF X$ = "0" THEN END
150 FOR P = LEN(X$) TO 1 STEP -1
160   PRINT MID$ (X$, P, 1);
170 NEXT P
180 PRINT
190 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
        OK
```

Alphabetizing

The process of alphabetizing a list differs little from that of 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, first, 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 Ø signals the end of the list. The program then sorts and displays the list in alphabetical order.

```
NEW
100 ' ALPHABETIZE (ALFBTIZE)
110 CLS: KEY OFF: 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 '
180 ' INPUT LOOP
190 LINE INPUT "? "; L$(N)
200 IF L$(N) = "0" THEN PRINT: GOTO 240
210 N = N + 1
220 IF N <= 25 THEN 180
230 '
240 ' ALPHABETIZE LIST
250 C$ = X$: M = 0
260 FOR P = 0 TO (N - 1)
270   IF L$(P) >= C$ THEN 290
280   C$ = L$(P): M = P
290 NEXT P
300 IF C$ = X$ THEN END
310 PRINT C$
320 L$(M) = X$
330 GOTO 240
RUN
```

Sample run:

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

(continued)

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

        ANNETTE
        BETH
        JOHN
        LUISA
        MARIANNE
        MARY
        PETER
        ZEKE
        Ok
```

Reverse Alphabetical Order

Just as you might sometimes want to sort numbers into descending order, so might you occasionally need to place a list into reverse alphabetical order (Z to A). This program 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 ' REVERSE ALPH. ORDER (REVALPH)
110 CLS: KEY OFF: 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 '
180 ' INPUT LOOP
190 LINE INPUT "? "; L$(N)
200 IF L$(N) = "0" THEN PRINT: GOTO 240
210 N = N + 1
```

```
220 IF N <= 25 THEN 180
230 '
240 ' REVERSE ALPHABETIZE LIST
250 C$ = X$: M = 0
260 FOR P = 0 TO (N - 1)
270   IF L$(P) <= C$ THEN 290
280   C$ = L$(P): M = P
290 NEXT P
300 IF C$ = X$ THEN END
310 PRINT C$
320 L$(M) = X$
330 GOTO 240
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
        OK
```

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, 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.

The next program figures out how much a loan will cost within a few cents, 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 ' LOAN TERMS (LNTERMS)
110 DEF FNM(X)=INT(((X*MP)*100)+.5)/100
120 CLS: KEY OFF
130 PRINT "LOAN TERMS": PRINT
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 '
190 ' COMPUTE TERMS
200 IF AI > 1 THEN AI = AI / 100
210 MP = AI / 12
220 F = MP + 1: T = F
230 FOR C = 2 TO NP
240   T = T * F
250 NEXT C
260 PF = 1 / T
270 SMP = A * (MP / (1 - PF))
280 SMP = (INT(SMP * 100)) / 100 + .01
290 CB = A
300 FOR C = 1 TO (NP - 1)
310   CB = INT((CB-SMP+FNM(CB))*100)/100
320 NEXT C
330 FP = CB + FNM(CB)
340 TP = (SMP * (NP - 1)) + FP
350 '
360 ' PRINT RESULTS
370 PRINT: BEEP
380 PRINT NP-1 "PAYMENTS OF" TAB(30) "$" SMP
390 PRINT " AND A FINAL PAYMENT OF" TAB(30) "$" FP
400 PRINT
410 PRINT " FOR A TOTAL OF $" TP
420 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
```

(continued)

```
 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
Ok
```

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, the program tells you and you have to start over.

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

```
NEW
100 ' LOAN ANALYSIS (LOANAN)
110 CLS: KEY OFF
120 ON ERROR GOTO 730
130 Q1$="PRINCIPAL AMOUNT  ": Q2$="MONTHLY PAYMENT   "
140 Q3$="NUMBER OF PAYMENTS": Q4$="ANNUAL % INTEREST "
150 GOTO 250
160 '
170 ' GET FACTORS
180 PRINT Q1$;: INPUT V: RETURN
190 PRINT Q2$;: INPUT M: RETURN
200 PRINT Q3$;: INPUT N: RETURN
210 PRINT Q4$;: INPUT I
220 IF I > 1 THEN I = I / 100
230 I = I / 12: RETURN
240 '
250 ' BEGIN HERE
260 PRINT "LOAN ANALYSIS:": PRINT
270 PRINT "ENTER 0 FOR THE UNKNOWN"
280 PRINT: GOSUB 180: IF V = 0 THEN 340
290 GOSUB 190: IF M = 0 THEN 410
300 GOSUB 200: IF N = 0 THEN 480
310 GOSUB 210: IF I = 0 THEN 550
320 PRINT: PRINT "  MUST HAVE ONE UNKNOWN": GOTO 270
330 '
340 ' CALCULATE PRINCIPAL
350 GOSUB 190: GOSUB 200: GOSUB 210
360 V = M * (1 - (1 / (1 + I)^N)) / I
370 V = INT(V * 100) / 100
380 PRINT: PRINT Q1$ "$" V
390 GOTO 680
400 '
410 ' CALCULATE PAYMENT
420 GOSUB 200: GOSUB 210
430 M = V * (I / (1 - (1 + I)^(-N)))
440 M = INT(M * 100) / 100
450 PRINT: PRINT Q2$ "$" M
460 GOTO 680
470 '
480 ' CALCULATE # PAYMENTS
490 GOSUB 210
500 N = LOG(1/(1-I*V/M)) / LOG(1+I)
510 N = INT(N + .4999)
520 PRINT: PRINT Q3$ N
530 GOTO 680
540 '
550 ' CALCULATE % INT RATE
560 PRINT "THINKING"
570 I1 = 2 * (N - V / M) / (N * (N + 1))
580 V1 = M * (1 - (1 + I1)^(-N)) / I1
590 IF V1 > (1.000001 * V) THEN 640
```

(continued)

```
600 IF V1 < (0.999999 * V) THEN 640
610 I = I1 * 1200
620 PRINT: PRINT Q4$ I: BEEP
630 GOTO 680
640 Y = (1 + I1)^(-N):  W = 1 - Y
650 I1 = I1 * (1-(((I1*V/M)-W) / (W-(N*I1*Y / (1+I1)))))
660 GOTO 580
670 '
680 ' QUIT OR REPEAT
690 PRINT: INPUT "ANOTHER RUN (Y/N)"; R$
700 IF R$="Y" THEN PRINT:PRINT:GOTO 250
710 END
720 '
730 ' ERROR HANDLER
740 PRINT: PRINT " TOO MANY UNKNOWNS"
750 PRINT " TRY AGAIN"
760 PRINT: PRINT: RESUME 270
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
OK
```

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 how much interest you paid over a period bracketed by two payments such as payment No. 19 in January and Payment No. 30 in December. 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 ' INTEREST PAID (INTPAID)
110 CLS: KEY OFF: F$ = "$$##,###.##"
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 USING F$; FNR(IP): PRINT
330 PRINT "STARTING BALANCE:";
340 PRINT USING F$; FNR(B1)
350 PRINT "ENDING BALANCE:";
360 PRINT USING F$; FNR(B2)
370 PRINT: PRINT: GOTO 130
RUN
```

(continued)

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:      $2,512.43

        STARTING BALANCE:         $29,969.63
        ENDING BALANCE:           $29,057.75


        MONTHLY PAYMENT:             ? 0
        Ok
```

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, its value will increase 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 to set aside 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 economist's term. It means the following: 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 tells you the lump sum to invest today in order to meet an objective several years from now.

The sample run shows that if you want $20,000 10 years from now, you have to invest $7368.97 at 10.5% or $6734.12 at

11.5% interest. Obviously, the amount of up-front money varies greatly with the interest rate.

```
NEW
100 ' PRES WORTH OF FUTURE AMT (PRESWRTH)
110 CLS: KEY OFF
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 '
220 ' DISPLAY RESULTS
230 PW = INT(PW * 100) / 100
240 PRINT: PRINT
250 PRINT "PRESENT WORTH IS $"; PW
260 PRINT: PRINT "----------------"
270 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
```

(continued)

```
    PRESENT WORTH IS $ 6734.12

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

      FUTURE AMOUNT      ? 0
    OK
```

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 questions such 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 ' FUTURE WORTH OF PRESENT AMT (FUTWRTH)
110 CLS: KEY OFF
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 '
220 ' DISPLAY RESULTS
230 FW = INT(FW * 100) / 100
240 PRINT: PRINT
250 PRINT "FUTURE WORTH IS $"; FW
260 PRINT: PRINT "----------------"
270 PRINT: GOTO 130
RUN

Sample run:
```

```
FUTURE WORTH:

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

FUTURE WORTH IS $ 19911.3

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

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

FUTURE WORTH IS $ 22356.51

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

  PRESENT AMOUNT      ? 0
  OK
```

This display means that if you have $5000 and you invest it at 9.65% interest, after 15 years it will be worth $19,911.30. If you're fortunate enough to earn 10.5% 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 your kid through college. Your objective is to have $30,000 15 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 perhaps 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 ' MONTHLY SAVINGS PLAN
110 CLS: KEY OFF
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 '
220 ' DISPLAY RESULTS
230 M  = INT(M * 100) / 100
240 PRINT: PRINT
250 PRINT "AMOUNT TO SAVE MONTHLY IS ";
260 PRINT USING "$$##.##"; M
270 PRINT: PRINT "----------------"
280 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
        AMOUNT TO SAVE MONTHLY IS  $77.54

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

          YOUR GOAL          ? 0
        OK
```

Annuity From a Present Amount

The next program is somewhat different from the preceding ones having to do with the time value of money. They dealt with attaining 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 have come into a $100,000 inheritance and have decided to live it up. You want it to last 5 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 following questions: "If I put $*P* in the bank at *I*%, how much can I take out each month to have it last *N* years?"

```
NEW
100 ' ANNUITY FROM AN AMT (ANNUITY)
110 CLS: KEY OFF
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 '
```

(continued)

```
220 ' DISPLAY RESULTS
230 MA = INT(MA * 100) / 100
240 PRINT: PRINT
250 PRINT "MONTHLY PAYOUT IS ";
260 PRINT USING "$$###.##"; MA
270 PRINT: PRINT "----------------"
280 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 $1942.80

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

          INITIAL AMOUNT     ? 0
        Ok
```

Lease Versus Purchase Analysis

It has become increasingly popular for people 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, since they 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 to 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 ' LEASE/PURCHASE ANALYSIS (LSPURCH)
110 CLS: KEY OFF: U$ = "----------"
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
```

(continued)

```
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 '
360 ' LEASE AND GENERAL COSTS
370 A1 = L1       : A2 = L2 / 12
380 A3 = L3 / 12 : A4 = L4 / 12
390 A5 = L5 / 12 : A6 = 0
400 A7 = G1 / 12 : A8 = G2 / 12
420 A9 = G3 / 12 : B1 = G4 / 12
420 L9 = A1+A2+A3+A4+A5+A7+A8+A9+B1
430 '
440 ' PURCHASE COSTS
450 B2 = P2 / 12
460 B3 = P3 / 12 : B4 = P4 / 12
470 B5 = P5 / 12 : B6 = P6 / Y / 12
480 P9 = B2+B3+B4+B5+B6+A7+A8+A9+B1
490 C2 = P7/12/Y : C3 = P1 / Y / 12
500 T  = P9 - C2 + C3
510 '
520 ' DISPLAY ANALYSIS
530 CLS: PRINT
540 PRINT "LEASE/PURCHASE FOR " I$:PRINT
550 PRINT TAB(18) "LEASE" TAB(29) "PURCHASE"
560 X$="MO. PAYMENT ":X1=A1:X2=0:GOSUB 800
570 X$="INSURANCE":X1=A2:X2=B2:GOSUB 800
580 X$="TAXES"     :X1=A3:X2=B3:GOSUB 800
590 X$="LICENSES"  :X1=A4:X2=B4:GOSUB 800
600 X$="MAINTENANCE":X1=A5:X2=B5:GOSUB 800
610 X$="INTEREST"  :X1=A6:X2=B6:GOSUB 800
620 X$="FUEL AND OIL":X1=A7:X2=A7:GOSUB 800
630 X$="POWER"     :X1=A8:X2=A8:GOSUB 800
640 X$="SUPPLIES"  :X1=A9:X2=A9:GOSUB 800
650 X$="MISC."     :X1=B1:X2=B1:GOSUB 800
660 PRINT TAB(18) U$ TAB(29) U$
670 X$=" CASH EXP/MO": X1=L9: X2=P9
680 GOSUB 800: PRINT
690 X$=" - SALVAGE":X1=0:X2=C2:GOSUB 800
```

```
700 X$=" + DEPREC.":X1=0:X2=C3:GOSUB 800
710 PRINT TAB(18) U$ TAB(29) U$
720 X$="NET EXP/MO":X1=L9:X2=T:GOSUB 800
730 END
799 '---------------------------
800 ' FORMAT OUTPUT LINE
810 PRINT X$;
820 PRINT TAB(17);
830 PRINT USING "$$#,###.##"; X1;
840 PRINT TAB(28);
850 PRINT USING "$$#,###.##"; X2
860 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
```

(continued)

```
        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        $0.00
        INSURANCE            $52.08       $52.08
        TAXES                 $0.00        $4.17
        LICENSES              $4.58        $4.58
        MAINTENANCE           $8.33       $31.25
        INTEREST              $0.00       $17.17
        FUEL AND OIL        $100.00      $100.00
        POWER                 $0.00        $0.00
        SUPPLIES             $20.83       $20.83
        MISC.                $25.00       $25.00
                          ---------    ---------
         CASH EXP/MO        $460.64      $255.09

         - SALVAGE            $0.00       $31.25
         + DEPREC.            $0.00      $229.17
                          ---------    ---------
        NET EXP/MO          $460.64      $453.00
        OK
```

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 a 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 in the category of petty drains on the pocketbook rather than as significant expenditures.

This program attempts to put a price on personal transportation by answering the following question: What does it *really* cost to operate the 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 includes an automatic kicker for unexpected repairs ($300 a year after you've driven 30,000 miles) and assumes that tires have to be replaced every 30,000 miles, but otherwise the results are extrapolations of the amounts you furnish.

```
NEW
100 ' CAR EXPENSE (CAREXP)
110 CLS: KEY OFF
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 = P1 + (P2 * P3)
240 INPUT "MILES PER GALLON    "; MG
250 INPUT "PRICE PER GALLON    "; GG
260 TG = (M / MG) * GG
```

(continued)

```
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 = (M / 30000) * (TC * 4)
330 INPUT "INSURANCE COST/YEAR"; IP
340 IP = IP * LY
350 INPUT "MISC EXPENSES      "; AA
360 C  = PC + TG + AM + TC + IP + AA
370 CM = INT(((C / M)*100) + .5)/100
380 '
390 ' OUTPUT RESULTS
400 M$ = "\                \$$####.##"
410 CLS: PRINT "OPERATING EXPENSES:"
420 PRINT USING M$;"  FUEL", TG
430 PRINT USING M$;"  UPKEEP",AM
440 PRINT USING M$;"  TIRES", TC
450 PRINT USING M$;"  INSURANCE", IP
460 PRINT USING M$;"  MISCELLANEOUS",AA
470 PRINT: OT = C - PC
480 PRINT USING M$;"OPERATING TOTAL",OT
490 PRINT
500 PRINT USING M$;"PURCHASE COST",PC
510 PRINT
520 PRINT USING M$;"TOTAL COST OF CAR",C
530 PRINT
540 PRINT "MILES DRIVEN       "; M
550 PRINT
560 PRINT USING M$;"COST PER MILE",CM
570 END
RUN
```

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                $5040.00
          UPKEEP              $1585.71
          TIRES                $840.00
          INSURANCE           $1700.00
          MISCELLANEOUS        $200.00

        OPERATING TOTAL       $9365.71

        PURCHASE COST        $10349.36

        TOTAL COST           $19715.07

        MILES                  70000

        COST PER MILE            $0.28
```

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. 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, 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 call this difference "stockholders' equity" and regard 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. 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. You can then 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 ' PERSONAL NET WORTH (NETWORTH)
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"
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 ' ----------
390 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 ' ----------
450 M$ = "\                              \$$###,###"
460 T = 30: GOSUB 1000
470 PRINT "CURRENT ASSETS:": PRINT
480 FOR X = 1 TO 8
490   PRINT C$(X) TAB(T);
500   INPUT C(X)
510   C(0) = C(0) + C(X)
520 NEXT X
530 GOSUB 1000   :' NEW SCREEN
540 PRINT "OTHER ASSETS:": PRINT
550 FOR X = 1 TO 12
560   PRINT A$(X) TAB(T);
570   INPUT A(X)
580   A(0) = A(0) + A(X)
590 NEXT X
600 GOSUB 1000   :' NEW SCREEN
610 PRINT "LIABILITIES:": PRINT
620 FOR X = 1 TO 7
```

(continued)

```
630   PRINT L$(X) TAB(T);
640   INPUT L(X)
650   L(0) = L(0) + L(X)
660 NEXT X
670 ' ----------
680 GOSUB 1000  :' NEW SCREEN
690 PRINT "ASSETS:"
700 FOR X = 1 TO 8
710   IF C(X) = 0 THEN 730
720   PRINT C$(X) TAB(T) C(X)
730 NEXT X
740 PRINT TAB(T) "---------"
750 PRINT "TOTAL CURRENT ASSETS" TAB(T);
760 PRINT USING "$$####,###";C(0): PRINT
770 FOR X = 1 TO 12
780   IF A(X) = 0 THEN 800
790   PRINT A$(X) TAB(T) A(X)
800 NEXT X
810 PRINT TAB(T) "---------"
820 PRINT "TOTAL OTHER ASSETS" TAB(T);
830 PRINT USING "$$####,###"; A(0)
840 PRINT TAB(T) "========="
850 PRINT "TOTAL ASSETS" TAB(T);
860 PRINT USING "$$####,###"; C(0) + A(0)
870 X$ = INKEY$: IF X$ = "" THEN 870
880 ' ----------
890 GOSUB 1000  :' NEW SCREEN
900 PRINT "LIABILITIES:"
910 FOR X = 1 TO 7
920   IF L(X) = 0 THEN 940
930   PRINT USING M$; L$(X), L(X)
940 NEXT X
950 PRINT TAB(T) "---------"
960 PRINT USING M$; "TOTAL LIABILITIES", L(0)
970 X = A(0)+C(0)-L(0)
980 PRINT USING M$; "NET WORTH", X
990 END
1000 ' SCREEN ADVANCE SUBRTN
1010 CLS: PRINT "PERSONAL NET WORTH:"
1020 PRINT
```

```
1030 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
```

(continued)

```
[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
                              =========
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
OK
```

SECTION 5
CALENDARS AND CLOCKS

Day of the Week

On which day of the week were you born? What day was it when our 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 to the one we presently use, so any day of the week given for a date before 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 ' DAY OF THE WEEK (DAYOWEEK)
110 CLS: KEY OFF
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:"
```

```
180 PRINT
190 INPUT "DATE AS MM,DD,YYYY...";M,D,Y
200 IF M = 0 THEN END
210 GOSUB 270
220 PRINT: PRINT
230 PRINT "DATE IS A " D$(B)
240 PRINT "------------------": PRINT
250 GOTO 180
260 '
270 ' FIGURE DAY OF WEEK
280 J = INT((Y - 1500) / 100)
290 A = (J * 5) + (J + 3) / 4
300 K = INT(A - FNB(A) * 7)
310 W = INT(Y / 100): V = INT(Y-W*100)
320 A = (V / 4) + V + D + T(M) + K
330 B = INT(A - FNB(A) * 7) + 1
340 IF M > 2 THEN 430
350 IF V = 0 THEN 410
360 T1 = INT(Y - FNA(Y) * 4)
370 IF T1 <> 0 THEN 430
380 IF B <> 0 THEN 400
390 B = 6
400 B = B - 1: GOTO 430
410 A = J - 1: T1 = INT(A-FNA(A)*4)
420 IF T1 = 0 THEN 380
430 IF B <> 0 THEN 450
440 B = 7
450 RETURN
460 DATA 0,3,3,6,1,4,6,2,5,0,3,5
470 DATA "SUNDAY", "MONDAY", "TUESDAY"
480 DATA "WEDNESDAY", "THURSDAY"
490 DATA "FRIDAY", "SATURDAY"
RUN
```

Sample run:

(continued)

```
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
OK
```

Perpetual Calendar

Using the next 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 more than a year or so. The program uses the same-day-finding subroutine (line 310 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 after 1582.

```
NEW
100 ' PERPETUAL CALENDAR
110 CLS: KEY OFF
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 310: 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 + 1
240 FOR X = 1 TO H
250   PRINT TAB(T);: T = T + 1
260   PRINT USING " ##"; X;
270   IF T > 25 THEN T = 0: PRINT
280 NEXT X
290 PRINT: PRINT: GOTO 170
300 '
310 ' FIRST DAY OF MONTH
320 J = INT((Y - 1500) / 100)
330 A = (J * 5) + (J + 3) / 4
340 K = INT(A - FNB(A) * 7)
350 W = INT(Y / 100): V = INT(Y-W*100)
360 A = (V / 4) + V + D + T(M) + K
370 B = INT(A - FNB(A) * 7) + 1
380 IF M > 2 THEN 470
390 IF V = 0 THEN 450
400 T1 = INT(Y - FNA(Y) * 4)
410 IF T1 <> 0 THEN 470
420 IF B  <> 0 THEN 440
430 B = 6
440 B = B - 1: GOTO 470
450 A = J - 1: T1 = INT(A - FNA(A) * 4)
460 IF T1 = 0 THEN 420
470 IF B <> 0 THEN 490
480 B = 7
490 RETURN
500 DATA  0,3,3,6,1,4,6,2,5,0,3,5
510 DATA 31,28,31,30,31,30,31,31
520 DATA 30,31,30,31
RUN
```

Sample run:

(continued)

```
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
OK
```

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? How many days did the Revolutionary War last? How many days 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 them 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 ' ELAPSED DAYS (ELAPDAYS)
110 DIM M(12)
120 FOR X = 1 TO 12
130   READ M(X)
140 NEXT X
150 '
160 ' GET DATES
170 CLS: KEY OFF
```

```
180 PRINT: PRINT "ELAPSED DAYS:": PRINT
190 INPUT "MONTH, DATE, YEAR"; MM,DD,YY
200 GOSUB 390
210 IF EC = 0 THEN 230
220 PRINT "**  ERROR  **": GOTO 190
230 D1 = J: PRINT
240 INPUT "MONTH, DATE, YEAR"; MM,DD,YY
250 GOSUB 390
260 IF EC = 0 THEN 280
270 PRINT "**  ERROR  **": GOTO 240
280 D2 = J: PRINT
290 '
300 ' COMPUTE ELAPSED DAYS
310 ED = ABS(D1 - D2)
320 ED = ED - INT(ED / (100 * 365.25))
330 ED = ED + INT(ED / (400 * 365.25))
340 PRINT ED "DAYS BETWEEN DATES"
350 PRINT: INPUT "ANOTHER (Y/N)"; X$
360 IF X$ <> "Y" THEN END
370 PRINT: PRINT: PRINT: GOTO 190
380 '
390 ' JULIAN DATE
400 EC = 0
410 IF DD > 31 THEN EC = 99: GOTO 490
420 IF MM > 12 THEN EC = 99: GOTO 490
430 X = YY * 365.25
440 J = INT(X)
450 L = X - J
460 J = J + M(MM) + DD
470 IF L <> 0 THEN 490
480 IF MM <= 2 THEN J = J - 1
490 RETURN
500 '
510 ' ELAPSED DAYS BY MONTH
520 DATA 0, 31, 59, 90, 120, 151, 181
530 DATA 212, 243, 273, 304, 334
RUN
```

(continued)

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
        OK
```

Digital Clock

The following 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 "13" to "25" in line 39Ø.

```
NEW
100 ' DIGITAL CLOCK (DCLOCK)
110 CLS: KEY OFF: PRINT: S = 0
120 PRINT "DIGITAL CLOCK:": PRINT
130 PRINT " ENTER HOUR, MINUTE"
140 INPUT " SEPARATED BY A COMMA"; H,M
150 '
160 ' FACE OF CLOCK
170 SCREEN 1
180 LINE (120,88)-(216,112),,B
190 GOSUB 440    ' DISPLAY TIME
200 '
210 ' TIMING LOOP
220 TIMER ON
230 ON TIMER(1) GOSUB 330
240 X$ = INKEY$
250 IF X$ <> "" THEN 280
260 GOTO 230
270 '
280 ' STOP THE CLOCK
290 TIMER OFF
300 SCREEN 0: CLS: END
310 ' ----------------
320 '
330 ' ADVANCE TIME ONE SECOND
340 S = S + 1
350 IF S < 60 THEN 410
360 S = 0: M = M + 1
370 IF M < 60 THEN 410
380 M = 0: H = H + 1
390 IF H < 13 THEN 410
400 H = 1
410 GOSUB 440
420 RETURN
430 '
440 ' DISPLAY THE TIME
450 T$ = ""
460 IF H < 10 THEN T$ = " "
470 H$ = STR$(H): L = LEN(H$)
```

(continued)

```
480 H$ = RIGHT$(H$, L-1)
490 T$ = T$ + H$ + ":"
500 M$ = STR$(M): L = LEN(M$)
510 M$ = RIGHT$(M$, L-1)
520 IF M < 10 THEN T$ = T$ + "0"
530 S$ = STR$(S): L = LEN(S$)
540 S$ = RIGHT$(S$, L-1)
550 T$ = T$ + M$ + ":"
560 IF S < 10 THEN T$ = T$ + "0"
570 T$ = T$ + S$
580 LOCATE 13,18
590 PRINT T$
600 RETURN
RUN
```

Ringing Digital Clock

The next program combines the accuracy and readability of a modern digital clock with the more old-fashioned notion of chiming the hour. The chime is Junior's beep tone.

This program builds upon the digital clock in the last program, and, in fact, you can load a copy of that program and make only a few changes to create this one. The changes are in lines 100, 110, 385, 400, 595, and the new lines 610–650. The listing given below is the complete program.

```
NEW
100 ' RINGING DIGITAL CLOCK (RCLOCK)
110 CLS: KEY OFF: PRINT: S = 0: R = 0
120 PRINT "DIGITAL CLOCK:": PRINT
130 PRINT " ENTER HOUR, MINUTE"
140 INPUT " SEPARATED BY A COMMA"; H,M
150 '
160 ' FACE OF CLOCK
170 SCREEN 1
180 LINE (120,88)-(216,112),,B
190 GOSUB 440    ' DISPLAY TIME
```

```
200 '
210 ' TIMING LOOP
220 TIMER ON
230 ON TIMER(1) GOSUB 330
240 X$ = INKEY$
250 IF X$ <> "" THEN 280
260 GOTO 230
270 '
280 ' STOP THE CLOCK
290 TIMER OFF
300 SCREEN 0: CLS: END
310 ' ---------------
320 '
330 ' ADVANCE TIME ONE SECOND
340 S = S + 1
350 IF S < 60 THEN 410
360 S = 0: M = M + 1
370 IF M < 60 THEN 410
380 M = 0: H = H + 1
385 R = H
390 IF H < 13 THEN 410
400 H = 1: R = H
410 GOSUB 440
420 RETURN
430 '
440 ' DISPLAY THE TIME
450 T$ = ""
460 IF H < 10 THEN T$ = " "
470 H$ = STR$(H): L = LEN(H$)
480 H$ = RIGHT$(H$, L-1)
490 T$ = T$ + H$ + ":"
500 M$ = STR$(M): L = LEN(M$)
510 M$ = RIGHT$(M$, L-1)
520 IF M < 10 THEN T$ = T$ + "0"
530 S$ = STR$(S): L = LEN(S$)
540 S$ = RIGHT$(S$, L-1)
550 T$ = T$ + M$ + ":"
560 IF S < 10 THEN T$ = T$ + "0"
570 T$ = T$ + S$
```

(continued)

```
580 LOCATE 13,18
590 PRINT T$
595 IF R <> 0 THEN GOSUB 620
600 RETURN
610 '
620 ' CHIME THE HOUR
630 BEEP
640 R = R - 1
650 RETURN
RUN
```

Tick-Tock

Like the preceding chiming clock, this program is a variation on the digital clock program. Instead of ringing the hour, however, it ticks each second. The only changes from the digital clock are the title in line 100 and a new line 595.

```
NEW
100 ' TICKING DIGITAL CLOCK (TCLOCK)
110 CLS: KEY OFF: PRINT: S = 0
120 PRINT "DIGITAL CLOCK:": PRINT
130 PRINT " ENTER HOUR, MINUTE"
140 INPUT " SEPARATED BY A COMMA"; H,M
150 '
160 ' FACE OF CLOCK
170 SCREEN 1
180 LINE (120,88)-(216,112),,B
190 GOSUB 440    ' DISPLAY TIME
200 '
210 ' TIMING LOOP
220 TIMER ON
230 ON TIMER(1) GOSUB 330
240 X$ = INKEY$
250 IF X$ <> "" THEN 280
260 GOTO 230
270 '
```

```
280 ' STOP THE CLOCK
290 TIMER OFF
300 SCREEN 0: CLS: END
310 ' ---------------
320 '
330 ' ADVANCE TIME ONE SECOND
340 S = S + 1
350 IF S < 60 THEN 410
360 S = 0: M = M + 1
370 IF M < 60 THEN 410
380 M = 0: H = H + 1
390 IF H < 13 THEN 410
400 H = 1
410 GOSUB 440
420 RETURN
430 '
440 ' DISPLAY THE TIME
450 T$ = ""
460 IF H < 10 THEN T$ = " "
470 H$ = STR$(H): L = LEN(H$)
480 H$ = RIGHT$(H$, L-1)
490 T$ = T$ + H$ + ":"
500 M$ = STR$(M): L = LEN(M$)
510 M$ = RIGHT$(M$, L-1)
520 IF M < 10 THEN T$ = T$ + "0"
530 S$ = STR$(S): L = LEN(S$)
540 S$ = RIGHT$(S$, L-1)
550 T$ = T$ + M$ + ":"
560 IF S < 10 THEN T$ = T$ + "0"
570 T$ = T$ + S$
580 LOCATE 13,18
590 PRINT T$
595 SOUND 37,1
600 RETURN
RUN
```

Countdown Alarm

Ovens, spacecraft launchers, and other things 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 absentminded 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 390. For example, if Jane says, "Call me back in 20 minutes," you can change the line to

```
390 PRINT "CALL JANE"
```

When the alarm sounds, that's the message that will appear.

```
NEW
100 ' COUNTDOWN ALARM (COUNTDN)
110 CLS: KEY OFF: PRINT
120 PRINT "COUNTDOWN ALARM:": PRINT
130 PRINT "HOURS, MINUTES, SECONDS";
140 INPUT HH, MM, SS
150 I$ = "## HOURS, ## MIN, ## SEC"
160 CLS: LOCATE 9, 13
170 PRINT "REMAINING TIME": GOSUB 310
180 '
190 ' COUNTDOWN LOOP
200 TIMER ON
210 ON TIMER(1) GOSUB 240
220 GOTO 210
230 '
240 ' REDUCE TIME ONE SECOND
250 SS = SS - 1
260 IF SS > -1 THEN 310
270 SS = 59: MM = MM - 1
280 IF MM > -1 THEN 310
290 MM = 59: HH = HH - 1
```

```
300 '
310 ' DISPLAY TIME
320 LOCATE 12, 8
330 PRINT USING I$; HH, MM, SS
340 IF HH=0 AND MM=0 AND SS=0 THEN 370
350 RETURN
360 '
370 ' TIME'S UP
380 LOCATE 15, 15
390 PRINT "TIME'S UP!"
400 BEEP
410 FOR T = 1 TO 600: NEXT T
420 X$ = INKEY$
430 IF X$ = "" THEN 400
440 '
450 ' SHUT DOWN ON KEYPRESS
460 CLS: END
RUN
```

SECTION 6

COMPUTER GRAPHICS FOR THE FUN OF IT

The IBM PC*jr* is a splendid machine for producing computer graphics. Its BASIC possesses a powerful set of graphics instructions that, although somewhat esoteric to those accustomed to standard Microsoft BASIC, nevertheless makes it relatively simple to create striking artworks, illusions of motion, bursts of color, shimmering kaleidoscopes, and other treats for the eye.

Graphics also has a serious side, of course, and that's why we've divided the subject into two sections. This section deals with graphic arts and lighthearted topics, and Section 7 presents its more businesslike face.

Virtually all of these programs build a display and "freeze" it so that you can admire the screen as long as you like. To stop the program and restore the machine to its normal state, tap the SPACE bar (actually, any key will do). The programs that produce moving displays work on the same principle and will stop anytime you press a key. Exceptions to this general rule will be noted in the text that precedes the affected program.

Most of the graphics programs are quite short, which testifies to the extraordinary amount of work done by the graphics instructions. One word of caution: Some graphics statements have several commas in a row, and it's important to include the correct number of commas or else the programs won't work. Otherwise they're easy to enter and fun to run. Have a good time. I did writing them!

Easter Egg

This program draws a brilliantly striped Easter egg against an off-white background. Besides looking pretty, it's a good tool for adjusting your monitor, since it displays most of Junior's 16 colors.

```
NEW
100 ' EASTER EGG (EGG)
110 KEY OFF: SCREEN 3: CLS
120 WINDOW (-80,50)-(80,-50)
130 CIRCLE (0,0),58,,,,2.5
140 '
150 ' BOUNDARY LINES
160 FOR Y = -30 TO 30 STEP 4
170   LINE (-30,Y)-(30,Y)
180 NEXT Y
190 '
200 ' PAINT THE BACKGROUND
210 PAINT (40,0),15,15
220 '
230 ' PAINT THE EGG
240 C = 0
250 FOR Y = 28 TO -28 STEP -4
260   PAINT (0,Y),C,15
270   C = C + 1
280 NEXT Y
290 '
300 ' FREEZE UNTIL KEYPRESS
310 IF INKEY$ = "" THEN 140
320 SCREEN 0: CLS: END
RUN
```

Bunting

Patriotism shines through this elegantly simple but colorful display. It's a red, white, and blue bunting studded with stars and surrounded by a circle of stars against a black background.

```
NEW
100 ' BUNTING
110 SCREEN 3: CLS: KEY OFF
120 WINDOW (-80,100)-(80,-100)
130 LINE (-40,30)-(40,10),12,BF
140 LINE (-40,10)-(40,-10),15,BF
150 LINE (-40,-10)-(40,-30),9,BF
160 '
170 ' CIRCLE OF STARS
180 RADIUS = 50
190 FOR ANGLE = 90 TO 450 STEP 12
200   R = ANGLE * (3.141593 / 180)
210   X = COS(R) * RADIUS
220   Y = SIN(R) * RADIUS * 5/3
230   PSET (X,Y)
240 NEXT ANGLE
250 '
260 ' STARS IN BUNTING
270 FOR X = -30 TO 30 STEP 10
280   FOR Y = 20 TO -20 STEP -40
290     PSET (X, Y)
300   NEXT Y
310 NEXT X
320 '
330 ' FREEZE DISPLAY UNTIL KEYPRESS
340 IF INKEY$ = "" THEN 340
350 SCREEN 0: CLS: END
RUN
```

Target

The display looks at first glance like an ordinary target from a shooting gallery, but if you stare at the bull's-eye for a while you can find all sorts of other things in it (including a hypnotic trance if you're not careful). The program draws a series of concentric circles, fills them alternately with red and black, and then turns off the white outlines of the circles.

```
NEW
100 ' TARGET
110 SCREEN 1: CLS: KEY OFF
120 WINDOW SCREEN (-160,110)-(159,-110)
130 COLOR 7,0
140 PALETTE 3,15: PALETTE 1,0
150 PALETTE 2,12
160 '
170 ' MAKE THE TARGET
180 FOR RADIUS = 10 TO 110 STEP 10
190   CIRCLE (0,0),RADIUS
200 NEXT RADIUS
210 '
220 ' PAINT THE BLACK RINGS
230 FOR X = 3 TO 103 STEP 20
240   PAINT (X,0),1,3
250 NEXT X
260 '
270 ' PAINT ALTERNATING RINGS RED
280 FOR X = 13 TO 93 STEP 20
290   PAINT (X,0),2,3
300 NEXT X
310 '
320 ' ERASE THE OUTLINES
330 PALETTE 3,0
340 '
350 ' FREEZE UNTIL KEYPRESS
360 X$ = INKEY$
370 IF X$ = "" THEN 360
380 SCREEN 0: CLS: END
RUN
```

Chambered Nautilus

At first you'll wonder what on Earth this program is doing, as it appears to squiggle a blob of light around in the center of a black screen. Soon, however, you'll watch the speeded-up growth of a chambered nautilus, one of nature's most beautiful designs, until it fills most of your display screen.

```
NEW
100 ' CHAMBERED NAUTILUS (NAUTILUS)
110 SCREEN 1: CLS
120 WINDOW SCREEN (-160,110)-(159,-110)
130 RADIUS = 2: ANGLE = 0: LX = 0: LY=0
140 '
150 ' SPIRAL OUT
160 RAD = ANGLE * (3.14159 / 180)
170 X = COS(RAD) * RADIUS
180 IF ABS(X) > 159 THEN 290
190 Y = SIN(RAD) * RADIUS
200 IF ABS(Y) > 110 THEN 290
210 LINE (X,Y)-(X/2, Y/2)
220 LINE (X,Y)-(LX,LY)
230 LX = X: LY = Y
240 RADIUS = RADIUS * 1.032
250 ANGLE = ANGLE + 16
260 IF ANGLE > 360 THEN ANGLE=ANGLE-360
270 GOTO 150
280 '
290 ' FREEZE UNTIL KEYPRESS
300 X$ = INKEY$
310 IF X$ = "" THEN 300
320 SCREEN 0: CLS: END
RUN
```

Spiral

Despite the almost ridiculous brevity of this program, it runs for a long time. About 6 minutes are needed for it to complete the spiral outward from the center of the screen. It's mesmerizing to watch it and even more so to stare at the focus of the finished spiral.

```
NEW
100 'SPIRAL
110 SCREEN 1: CLS
120 WINDOW (-160,-110)-(159,109)
130 RADIUS = 1: INCR = .03
```

```
140 ANGLE = 0
150 '
160 ' LOOP TO DRAW SPIRAL
170 RAD = ANGLE * (3.141593 / 180)
180   X = INT(COS(RAD) * RADIUS)
190   Y = INT(SIN(RAD) * RADIUS)
200   IF ABS(X) > 159 THEN 270
210   IF ABS(Y) > 109 THEN 270
220   PSET (X, Y)
230   ANGLE = ANGLE + 1
240   RADIUS = RADIUS + INCR
250 GOTO 160
260 '
270 ' FREEZE DISPLAY
280 IF INKEY$ = "" THEN 280
290 SCREEN 0: CLS
300 END
RUN
```

Polygon

Use this program to dazzle your friends with the speed of a computer. It asks you how many sides you want the polygon to have. Type any positive number greater than two. The computer pauses for a second or so while it "solves" the polygon, and then it draws it in the twinkle of a cursor. Each angle of the polygon is joined to the center by a line; there is also a perimeter that defines the actual shape. The computer labels the drawing to tell you how many sides the figure has. Tap a key to make it go away.

```
NEW
100 ' POLYGON
110 CLS: KEY OFF
120 PRINT: PRINT "POLYGON:": PRINT
130 INPUT "HOW MANY SIDES"; SIDES
140 '
```

(continued)

```
150 ' BUILD ARRAY OF CORNER COORDS
160 DIM C (SIDES, 2)
170 ANGLE = 270: RADIUS = 80
180 INCR = 360 / SIDES
190 FOR RAY = 1 TO SIDES
200   RAD = ANGLE * (3.141593 / 180)
210   C(RAY,1) = COS(RAD) * RADIUS
220   C(RAY,2) = SIN(RAD) * RADIUS
230   ANGLE = ANGLE + INCR
240 NEXT RAY
250 C(0,1) = C(SIDES,1)
260 C(0,2) = C(SIDES,2)
270 '
280 ' SET UP SCREEN
290 SCREEN 1: CLS
300 WINDOW SCREEN (-160,110)-(159,-110)
310 PRINT TAB(13);
320 PRINT SIDES "SIDED FIGURE"
330 '
340 ' PLOT THE FIGURE
350 FOR S = 1 TO SIDES
360   SX = C(S,1): SY = C(S,2)
370   DX = C(S-1,1): DY = C(S-1,2)
380   LINE (SX,SY)-(DX,DY)
390   LINE (SX,SY)-(0,0)
400 NEXT S
410 '
420 ' FREEZE UNTIL SIGNAL
430 X$ = INKEY$
440 IF X$ = "" THEN 430
450 SCREEN 0: CLS
460 END
RUN
```

Continuous Series of Polygons

This program makes a great conversation piece at a party. Just start it and let it run, and the folks will gather 'round to watch it cycle tirelessly through all the polygons from 5 to 20 sides and back down again.

The program embellishes on the preceding one, but there is one difference in the figures it produces. Here, all the angles are joined, filling the inside of each polygon with a maze of lines. They create interesting illusions (a ball of jewels, a space station from *Star Wars*, a string sculpture with glass beads, etc.). When your guests' interests begin to flag, ask them what they see in the figures. And be prepared for some strange visions in the long conversation that will surely take place.

You can stop the program by pressing any key at any time. The actual halt comes the next time the program decides to change the figure.

```
NEW
100 ' POLYGON DEMO (MANYGONS)
110 CLS: KEY OFF
120 SIDES = 5: MOST = 20: SV = 1
130 DIM C(MOST, 2)
140 '
150 ' BUILD ARRAY OF CORNER COORDS
160 ANGLE = 270: RADIUS = 80
170 INCR = 360 / SIDES
180 FOR RAY = 1 TO SIDES
190   RAD = ANGLE * (3.141593 / 180)
200   C(RAY,1) = COS(RAD) * RADIUS
210   C(RAY,2) = SIN(RAD) * RADIUS
220   ANGLE = ANGLE + INCR
230 NEXT RAY
240 '
250 ' SET UP SCREEN
260 SCREEN 1: CLS
270 WINDOW SCREEN (-160,110)-(159,-110)
280 PRINT TAB(13);
290 PRINT SIDES "SIDED FIGURE"
300 '
310 ' PLOT THE FIGURE
320 FOR S = 1 TO SIDES
330   SX = C(S,1): SY = C(S,2)
```

(continued)

```
340   FOR V = S+1 TO SIDES
350     DX = C(V,1): DY = C(V,2)
360     LINE (SX,SY)-(DX,DY)
370   NEXT V
380 NEXT S
390 '
400 ' CHECK FOR SIGNAL TO END
410 X$ = INKEY$
420 IF X$ = "" THEN 450
430 SCREEN 0: CLS: END
440 '
450 'REPEAT FOR NEXT POLYGON
460 FOR DELAY = 1 TO 500: NEXT DELAY
470 SIDES = SIDES + SV
480 IF SIDES = MOST THEN SV = -1
490 IF SIDES = 5 THEN SV = 1
500 GOTO 150
RUN
```

Patchwork Quilt

This program produces a rectangular quilt of four-by-four multicolored patches. Some patches have an up-and-down pattern, some a horizontal division, and others split on the diagonal. The quilt is brilliantly colored.

Because it relies heavily on randomness to produce both the sequences of patch patterns and the colors, you can run this program a million times and never get the same quilt twice. The basic layout will always be the same, but the combination of patches and colors will always be different.

```
NEW
100 ' PATCHWORK DESIGN (PATCHES)
110 PX = 0: PY = 0
120 DEF FNR(X)=INT(RND(1)*14)+1
130 SCREEN 3: CLS: RANDOMIZE TIMER
140 '
150 ' BUILD THE QUILT WITH VIEWPORTS
160 FOR PORT = 0 TO 15
```

```
170   CX = PX + 39: CY = PY + 49
180   CA = FNR(0)
190   VIEW (PX,PY)-(CX,CY),FNR(0),15
200 'SLICE VIEWPORT
210   V=FNR(0): M3=V MOD 3: M2=V MOD 2
220   IF M3=0 THEN GOSUB 460: GOTO 250
230   IF M2=0 THEN GOSUB 370: GOTO 250
240   GOSUB 410
250   PAINT ( 2,25),FNR(0),15
260   PAINT (37,35),FNR(0),15
270 'ADVANCE VIEWPORT LOCATION
280   PX = PX + 40
290   IF PX < 159 THEN 310
300   PX = 0: PY = PY + 50
310 NEXT PORT
320 '
330 ' FREEZE UNTIL KEYPRESS
340 IF INKEY$ = "" THEN 340
350 SCREEN 0: CLS: END
360 '
370 ' SLICE PORT HORIZONTALLY
380 LINE (0,25)-(39,25),15
390 RETURN
400 '
410 ' SLICE PORT VERTICALLY
420 LINE (13,0)-(13,49),15
430 LINE (27,0)-(27,49),15
440 RETURN
450 '
460 ' SLICE PORT DIAGONALLY
470 IF V > 7 THEN 500
480 LINE ( 0, 0)-(39,49),15
490 GOTO 510
500 LINE ( 0,49)-(39, 0),15
510 RETURN
RUN
```

Crazy Quilt

This program is the same general idea as the last one, except that it builds its pattern from randomly placed overlapping geometric figures and then fills the spaces between lines with assorted colors. Occasionally, it changes the same area's color several times, and usually it leaves some areas uncolored. The result is a wild profusion of shapes and colors that delights fans of abstract art and pleases the eye of anyone who likes colorful and somehow orderly disarray. The probability of this program ever repeating a pattern is one in several hundred trillion.

```
NEW
100 ' CRAZY QUILT (CRAZYQLT)
110 SCREEN 3: CLS: KEY OFF
120 WINDOW (0,199)-(159,0)
130 RANDOMIZE TIMER
140 DEF FNI(X) = INT(RND(1) * 200)
150 '
160 ' QUILT PIECES
170 FOR P = 1 TO 25
180   Q = FNI(1)
190   IF (Q MOD 2) = 1 THEN 210
200   GOSUB 380: GOTO 220
210   GOSUB 430
220 NEXT P
230 '
240 ' COLOR THE QUILT
250 FOR P = 1 TO 200
260   Y = FNI(1)
270   X = FNI(1)
280   IF X > 159 THEN 270
290   C = INT(FNI(1) / 14)
300   PAINT (X,Y),C,15
310 NEXT P
320 '
330 ' FREEZE UNTIL KEYPRESS
340 IF INKEY$ = "" THEN 340
350 SCREEN 0: CLS: END
360 ' --------------------
370 '
```

```
380 ' CIRCLE PATTERN
390 X = FNI(1): Y = FNI(1): R = FNI(1)
400 CIRCLE (X,Y),R
410 RETURN
420 '
430 ' RECTANGULAR PATTERN
440 X1 = FNI(1): Y1 = FNI(1)
450 X2 = FNI(1): Y2 = FNI(1)
460 LINE (X1,Y1)-(X2,Y2),,B
470 RETURN
RUN
```

Perspective Puzzle

This is the kind of *trompe l'oeil* (joke on the eye) that made Max Escher a famous artist. It's a whole screen full of interlocking cubes. The question is, are you closer to their upper-left-hand corners and looking down, or are you closer to their lower-right-hand corners and looking up? It can be either way, you see, and there is no correct answer. It's a puzzle of perspective.

And just to further trick the eye, about every 2 seconds one of the colors in the puzzle changes. That can have the startling effect of turning the entire display inside out (or at least your perspective of it).

```
NEW
100 ' CUBES
110 RANDOMIZE TIMER
120 SCREEN 1: COLOR 7,0: CLS: KEY OFF
130 PALETTE 3,15: PALETTE 1,0
140 WINDOW (0,0)-(319,199)
150 '
160 BX = 0: BY = 307: X=BX: Y=BY
170 GOSUB 330
180 X = X + 36: Y = Y - 12
190 IF X < 319 THEN 170
200 BX = BX - 24: BY = BY - 24
```

(continued)

```
210 X = BX: Y = BY
220 IF Y > 0 THEN 170
230 '
240 ' CYCLE COLORS UNTIL KEYPRESS
250 IF INKEY$ = "" THEN 270
260 SCREEN 0: CLS: END
270 FOR T = 1 TO 1000: NEXT T
280 A = INT(RND(1) * 4)
290 C = INT(RND(0) * 16)
300 PALETTE A,C
310 GOTO 240
320 '
330 ' DRAW A CUBE AT X,Y
340 PSET (X,Y)
350 DRAW "R24 F12 L24 H12"
360 DRAW "BF3 BR3 P1,3 BL3 BH3"
370 DRAW "C3 D24 F12 U24"
380 DRAW "BL3 P2,3 BR3"
390 DRAW "C3 D24 R24 U24"
400 DRAW "BL4 BU2 P1,3 BD2 BR4 C3"
410 RETURN
RUN
```

Kaleidoscope

Here's another party program that runs forever (unless you press a key), producing an ever-changing kaleidoscopic sunburst of colors on the screen. It's not as challenging to the imagination as the continuous series of polygon program, but the shimmering patterns of colors are a joy to behold. It's recommended to counteract the blahs of a rainy gray day.

```
NEW
100 ' KALEIDOSCOPE (KALEIDO)
110 SCREEN 3: CLS: KEY OFF
120 WINDOW (-80,100)-(80,-100)
130 RADIUS = 100
140 FOR ANGLE = 0 TO 360 STEP 20
150   R = ANGLE * (3.141593 / 180)
```

```
160    X = COS(R) * RADIUS
170    Y = SIN(R) * RADIUS * 5/3
180    LINE (0,0)-(X,Y)
190 NEXT ANGLE
200 CIRCLE (0,0),65
210 '
220 ' FILL WITH COLORS
230 C = 1
240 FOR RADIUS = 30 TO 70 STEP 30
250    FOR ANGLE = 10 TO 350 STEP 20
260       R = ANGLE * (3.141593 / 180)
270       X = COS(R) * RADIUS
280       Y = SIN(R) * RADIUS * 5/3
290       PAINT (X,Y),C,15
300       C = C + 1
310       IF C > 14 THEN C = 1
320    NEXT ANGLE
330 NEXT RADIUS
340 '
350 ' SHIMMER
360 DEF FNC(X) = INT(RND(1) * 14) + 1
370 A = FNC(1): C = FNC(1)
380    PALETTE A,C
390 FOR T = 1 TO 300: NEXT T
400 IF INKEY$ = "" THEN 370
410 SCREEN 0: CLS: END
RUN
```

Simple Animation

This program introduces apparent motion into the display by rotating a hoop around a vertical axis. It creates the illusion of animation in the same manner as a movie film, with a series of still frames carrying progressive stages of the action. That accounts for the visible stepping of the hoop as it spins.

The program uses two screen images within the computer memory. While one is visible on the display, the program clears the other and builds the image, then switches the visible frames.

You don't see the actual swap but see instead the result as the hoop appears to rotate a little farther. This continues for as long as the program runs. Like most of the others, this one stops when you press a key.

As you watch the hoop, see if you can figure out which way it's turning (left to right or right to left?).

```
NEW
100 ' HOOP
110 CLEAR ,,,34816!
120 V = 1: A = 30: R = 60
130 DIM ASPECT(A)
140 '
150 ' LOAD ARRAY FOR ELLIPSES
160 FOR P = 0 TO A
170   X = (P * (90 /A)) * .0174533
180   Y = R / ((COS(X) * R) + .01)
190   ASPECT(P) = Y
200 NEXT P
210 '
220 ' SIDE VIEW TO END VIEW
230 SCREEN 1: CLS: COLOR 1,1
240 FOR P = 0 TO A
250   GOSUB 360 :REM SPIN THE HOOP
260   GOSUB 480 :REM CHECK FOR STOP
270 NEXT P
280 '
290 ' END VIEW TO SIDE VIEW
300 FOR P = A-1 TO 1 STEP -1
310   GOSUB 360 :REM SPIN AGAIN
320   GOSUB 480 :REM CHECK FOR STOP
330 NEXT P
340 GOTO 240
350 '
360 ' SUBRTN TO ADVANCE ROTATION
370 CLS
380 CIRCLE (160,100),R,2,,,ASPECT(P)
390 LINE (160,30)-(160,169),1
400 IF V = 0 THEN 440
410 V = 0
420 SCREEN 1,,1,0: COLOR 1,1
```

```
430 GOTO 460
440 V = 1
450 SCREEN 1,,0,1: COLOR 1,1
460 RETURN
470 '
480 ' CHECK FOR KBD STOP SIGNAL
490 X$ = INKEY$
500 IF X$ = "" THEN RETURN
510 SCREEN 0: CLS
520 END
RUN
```

Zooming In and Out

One of the neatest things about computer art is that you can create a living, moving "reality" that exists only in your imagination. Putting it another way, the objects that appear on the screen don't necessarily have to look or behave like anything familiar. This is true, of course, in any art form, but artists who work in static media, such as painters and sculptors, can only suggest motion, whereas a dynamic medium such as a display screen can actually produce it.

This program does precisely that. The *objet d'art* is very simple. It zooms, shrinking and growing in a manner all its own. Although I invented it, I don't know what it is and I don't think it matters; perhaps after watching it awhile you'll discover that it resembles something in your reality.

```
NEW
100 ' ZOOM
110 SCREEN 1: CLS
120 WINDOW (-160,110)-(160,-110)
130 COLOR 0,0: PALETTE 3,15
140 S=20: F=50: I=1: GOSUB 180
150 S=49: F=21: I=-1: GOSUB 180
160 GOTO 140
170 '
```

(continued)

```
180 ' SUBRTN TO ZOOM DISPLAY
190 FOR R = S TO F STEP I
200   LINE (-R,0)-(R,0)
210   LINE (0,-R*.9)-(0,R*.9)
220   CIRCLE (0,0),R
230   IF INKEY$ <> "" THEN 280
240   CIRCLE (0,0),R-I,0
250 NEXT R
260 RETURN
270 '
280 ' PAINT THE FIGURE
290 PAINT (-5, 5),1,3
300 PAINT ( 5,-5),1,3
310 PAINT ( 5, 5),2,3
320 PAINT (-5,-5),2,3
330 '
340 ' FREEZE UNTIL KEYPRESS
350 IF INKEY$ = "" THEN 350
360 SCREEN 0: CLS: END
RUN
```

SECTION 7
GETTING DOWN TO BUSINESS WITH GRAPHICS

The business and professional worlds are only beginning to discover the potential of computer graphics. Since business applications tend to be highly specialized—in contrast to the general kinds of programs appearing in this book—this chapter probably won't do much to advance the cause of graphics among the more serious-minded. On the other hand, here are three graphics utility programs that are useful "as is" and give a glimpse of the potential in representing data pictorially.

Equation Plotting

This is a self-scaling plot program that uses the full screen to graph the curve of any equation whose x is variable and whose y is the unknown.

The program asks you to supply the range of x. It then scales the screen to fit the range of x and the approximate range of the resulting y. If the axes are within the plot area, it draws them. It also furnishes dots within the plot area to show the intersections of the reference points; that is, the first dot below and to the right of the axis intersection is at $x = 1$, $y = -1$, the first dot above and right of the Ø point is $x + 1$, $y = 1$, and so on, according to the standard Cartesian convention. The plot itself appears as a curve (normally) consisting of dots. Finally, the program reports the range of y to three decimal places across the bottom of the display.

This program listing furnishes an equation for demonstration purposes. Run it with x in the range from –2 to 5 and it will produce a double curve that sweeps up from the lower-left-hand corner, bends down toward the lower-right-hand corner, then shoots up off the upper-right-hand corner of the screen, with y ranging from –19.000 to 7.049.

Naturally, you'll want to plug your own equations into this program and not always run the one supplied here. To do this, enter the calculations as BASIC instructions starting at line 930. You may use as many lines as you need. The program provides a RETURN at line 999 (since the equation operates as a subroutine), so you don't need to worry about program control or structure. Your calculations are subject to the following rules in this program:

1. The controlling variable must have the name X.
2. The computations must resolve to a variable named Y.
3. BASIC programming notation must be used.
4. The first (or only) line of the calculation must be line 930, and subsequent lines must have numbers in the range 931–998.

These rules might force you to rearrange and/or rewrite an equation, substituting X and Y for other variable names used in the original expression. Suppose, for example, you have the equation

$$b = 2a^2 - 5a - 3$$

In this case, a is the controlling variable and b is the unknown. Thus, by substitution according to the rules, it becomes

$$y = 2x^2 - 5x - 3$$

Converted into BASIC notation the instruction reads

```
930 Y = (2 * X^2) - (5 * X) - 3
```

which is exactly the line you need to type in to incorporate it into the program.

```
NEW
100 ' XY EQUATION PLOT (PLOTXY)
110 ' SELF-SCALING PLOT PROGRAM
120 CLS: KEY OFF
130 PRINT: PRINT "EQUATION PLOT"
140 PRINT: PRINT
150 INPUT "LOWEST X..."; LX
160 INPUT "HIGHEST X.."; HX
170 '
180 ' PROGRAM DRIVER
190 GOSUB 340   :' DEVELOP Y SCALE
200 GOSUB 440   :' SCALE PLOT AREA
210 GOSUB 520   :' MARK X AXIS
220 GOSUB 580   :' MARK Y AXIS
230 GOSUB 680   :' X INCR FOR PLOT
240 GOSUB 740   :' PLOT THE GRAPH
250 GOSUB 830   :' DISPLAY RANGE OF Y
260 '
270 ' FREEZE DISPLAY
280 IF INKEY$ = "" THEN 280
290 SCREEN 0: END
300 '--------------------------------
310 '
320 ' SUBROUTINES
330 '
340 ' DETERMINE APPROX Y LIMITS
350 IX = (HX - LX) / 10
360 LY = 999999!: HY = -LY
370 FOR X = LX TO HX STEP IX
380   GOSUB 920
390   IF Y < LY THEN LY = Y
400   IF Y > HY THEN HY = Y
410 NEXT X
420 RETURN
430 '
440 ' SCALE PLOT
450 SCREEN 1
460 XL = LX * 1.05: XR = HX * 1.05
```

(continued)

```
470 YT = HY * 1.05: YB = LY * 1.1
480 WINDOW (XL,YT)-(XR,YB)
490 RETURN
500 '
510 '
520 ' DRAW X AXIS
530 IF LY > 0 THEN 560
540 IF HY < 0 THEN 560
550 LINE (LX,0)-(HX,0)
560 RETURN
570 '
580 ' DRAW Y AXIS
590 IF LX > 0 THEN 620
600 IF HX < 0 THEN 620
610 LINE (0,HY)-(0,LY)
620 FOR X = INT(LX) TO HX
630   FOR Y = LY TO HY
640     PSET (X,Y)
650   NEXT Y
660 NEXT X: RETURN
670 '
680 ' COMPUTE X STEP VALUE
690 L  = PMAP(LX,0): R = PMAP(HX,0)
700 D  = R - L
710 IX = (HX - LX) / D
720 RETURN
730 '
740 ' PLOT EQUATION
750 FOR X = LX TO HX STEP IX
760   GOSUB 920
770   PSET (X,Y)
780   IF Y > HY THEN HY = Y
790   IF Y < LY THEN LY = Y
800 NEXT X
810 RETURN
820 '
830 ' DISPLAY RANGE OF Y
840 LOCATE 25,1
850 I1$ = "RANGE OF Y: "
860 I2$ = "###.### TO ###.###"
```

```
870 I$  = I1$ + I2$
880 PRINT USING I$; LY, HY;
890 LOCATE 1,1
900 RETURN
910 '
920 ' EQUATION
930 Y = X^3 - (5 * X^2) - X + 7
999 RETURN
RUN
```

Bar Charts

This listing introduces the concept, familiar to seasoned programmers but new to others, of the *library subroutine*. It is a canned routine that is not a complete program, but one that you can incorporate into your programs simply by building it in, without having to figure out and write the instructions yourself and without even having to understand how it works.

This particular library routine produces a bar chart. Your program must pass it certain information that it uses to draw a professional-looking graph in up to four colors on the screen of the computer. You load certain variables with the information and then call one of its three subroutines with a GOSUB instruction. The library routine takes care of the actual graphing. This relieves you of the tedium of figuring out how to make the machine do graphics functions.

The first step is to type the BARCHART subprogram shown in the listing and save it on disk with the command

```
SAVE "BARCHART",A
```

Once this is done, you can create programs that use BARCHART by first loading the routine with the command.

```
LOAD "BARCHART"
```

and then developing your program in lines 1–998, following the guidelines discussed below. When you save your program, BARCHART becomes a part of it as though you had written the instructions yourself. An example will be given in the next program listing ("Sales chart").

Guidelines: BARCHART provides three services for constructing bar graphs:

GOSUB 1ØØØ sets up the screen for the chart.
GOSUB 13ØØ draws a bar.
GOSUB 14ØØ freezes the display until you press a key.

The first two services require that you load certain variables with control information before calling them, as follows.

Screen setup:

The screen setup is always the first step in building a bar graph since it establishes the operating environment, scale, and layout of the screen according to the data you plan to display. After loading the variables, execute GOSUB 1ØØØ and BARCHART will automatically adjust the screen as you specify.

XS	=	the X scale, that is, the number of horizontal reference points across the screen, assuming that the left edge of the screen is point Ø.
YS	=	the Y scale, that is, the number of vertical reference points assuming that Ø is at the bottom of the screen and the points are numbered upward.
BG	=	the number of the color for the screen background.
A1	=	attribute No. 1 color.
A2	=	attribute No. 2 color.
A3	=	attribute No. 3 color.

(*Attributes* are numbers that refer to colors. BARCHART uses the medium resolution graphics mode, which permits up to three colors in addition to the background. The IBM color coding scheme assigns specification number 6 to the color brown, so the instruction

```
A1 = 6
```

says that "hereafter, when I refer to attribute No. 1, I mean brown." These assignments remain in effect for as long as the program continues to run.)

YR = the number of vertical units between horizontal reference lines across the chart. These lines indicate the scale of the bars, so if the maximum bar height on the graph is 1Ø, specify

YS = 1Ø: YR = 1

and BARGRAPH will draw a base line and 1Ø reference lines evenly spaced. This lets you see the magnitude each bar represents. If you don't want reference lines on the graph, assign Ø to YR.

H$ = screen heading. This is a line of text across the top of the screen telling what the graph represents. The text must be less than 40 characters in length.

Plotting a bar

Each time you want to draw a bar, you must set up the following variables and then execute GOSUB 13ØØ.

B1 = the location of the bar's left edge, measured in "XS" units from the left side of the screen.
BW = the width of the bar in "XS" units.
BH = the height of the bar in "YS" units.

Note: Normally "B1" and "BW" are greater than Ø and less than "XS," and "BH" is a value between Ø and "YS." BARCHART does not check to make sure these variables fall within the view of the screen, however, and it does not report an error if they fall outside it. The result of a range error may be a bar partially or wholly outside the viewing window. It is your responsibility to ensure that the screen boundaries are respected.

BA = the attribute number of the bar. Suppose, for example, that in the setup you specified

A1 = 6

assigning brown to attribute No. 1. If you want a brown bar, before GOSUB 13ØØ specify

BA = 1

The value of "BA" must always be in the range Ø–3, where Ø is the screen background color and 1–3 refer to the attributes whose colors were established in the setup phase.

Freezing the display

Code GOSUB 1400 to hold the display until you press a key. No setup is required. This subroutine places the display in a state of suspended animation until a key is operated, and then it clears the screen and returns to the program. If you want the program to stop at this point, write

```
GOSUB 1400: END
```

since the subroutine itself does not automatically halt execution.

All of this might seem to be rather complicated, but it actually substantially reduces the effort of writing bar chart programs. The next listing gives a sample of programming using BARCHART.

```
NEW
999 ' SUBRTN BARCHART
1000 ' SET UP SCREEN FOR GRAPH
1010 ' PARAMETERS:
1020 '    XS      X SCALE (0 - XS)
1030 '    YS      Y SCALE (0 - YS)
1040 '    BG      BACKGROUND COLOR
1050 '    A1      ATTR #1 COLOR
1060 '    A2      ATTR #2 COLOR
1070 '    A3      ATTR #3 COLOR
1080 '    YR      INTERVAL BTWN Y REF'S
1090 '    H$      CHART HEADING
1100 '
1110 SCREEN 1: CLS: COLOR BG, 0
1120 PALETTE 1, A1
1130 PALETTE 2, A2
1140 PALETTE 3, A3
1150 LH = LEN(H$)
1160 LH = INT((40 - LH) / 2) + 1
1170 LOCATE 1, LH: PRINT H$
1180 LX = 0 - (XS * .1)
1190 RX = XS + (XS * .1)
1200 TY = YS + (YS * .1)
1210 BY = 0  - (YS * .1)
1220 WINDOW (LX,TY)-(RX,BY)
```

```
1230 LINE (0,YS)-(0,0)
1240 IF YR = 0 THEN 1280
1250 FOR Y = 0 TO YS STEP YR
1260   LINE (0,Y)-(XS,Y)
1270 NEXT Y
1280 RETURN
1290 '-------------------
1300 ' PLOT A BAR
1310 ' PARAMETERS:
1320 '   B1      BAR LEFT EDGE
1330 '   BW      BAR WIDTH
1340 '   BH      BAR HEIGHT
1350 '   BA      BAR ATTRIBUTE
1360 B1 = BL + BW
1370 LINE (BL,BH)-(B1,0),BA,BF
1380 RETURN
1390 '-------------------
1400 '
1410 ' FREEZE CHART UNTIL KEYPRESS
1420 IF INKEY$ = "" THEN 1420
1430 SCREEN 0: CLS
1440 RETURN
1450 '-------------------
SAVE "BARCHART",A
```

A Sample Bar Chart Program

This program illustrates how to use the BARCHART library routine by creating a program that reports projected sales and actual sales over four quarters for a hypothetical organization.

The first step in writing a program that will use a library routine is to bring the routine into memory from disk, so we begin with the command

```
LOAD "BARCHART"
```

This makes BARCHART the basis for any program we write, and the program itself is an addition to the already existing lines of BARCHART. The only thing we must be careful to do, aside from using BARCHART correctly, is to make sure our new program lines don't conflict with those of BARCHART. Consequently, our program must avoid using line numbers in the range 999–1450. Also, because BASIC executes line numbers in numeric sequence, the new part of the program has to begin with line numbers less than 999. (If 1–998 aren't enough lines, we can have a GOTO that jumps to additional program lines above 1450, although that's seldom necessary and won't be done here.)

The display will show projected versus actual sales for each of four quarters. Projected sales will be a green bar, actual sales red. Each quarter will have a space, a green bar, another space, then a red bar, and three bar-widths will separate it from the next quarter to the right. Stated another way, the patterns will consist of space–bar–space–bar–space (5 bar-widths × 4 clusters), plus an extra space between each cluster, for a total of 23 units of width. Consequently, "XS" = 23.

The maximum sales volume per quarter in this organization is $100,000, so the vertical scale of the chart is "YS" = 100,000. We'll mark the graph every $10,000, giving "YR" = 10,000. This comprises line 110 of the program.

The chart will have a blue background ("BG" = 1), with bars in green ("A1" = 2) and red ("A2" = 4). Lettering in IBM BASIC normally appears in the highest-numbered attribute, so to have soft white text on the screen we'll assign "A3" = 7. Line 120 includes these setup instructions.

Finally, the text of the graph heading will read "SALES PROJ VS ACTUAL," which appears in line 130 and completes the information needed before calling the setup subroutine in line 140.

The sales volumes themselves appear in DATA statements at lines 310 and 330 (although they could be anywhere in the program listing). These volumes are the bar heights on a scale of 0–100,000, so the loops at 170–200 and 230–270 read them as "BH." Line 160 establishes the bar width "BW" as one unit. For the first loop, the bar color attribute "BA" = 1 for green, and in the second, "BA" = 2 for red. In this way, the loops superimpose the data bars on the graph, projections first, then actuals.

Line 280 freezes the display with a call to 1400, and when you press a key, END stops the program with the screen cleared.

This is, of course, only one approach to structuring a program that uses the BARCHART services. It is not necessary to use DATA statements; we could have coded the data directly into the program instructions (usually a poor idea, however). Also, the rationale in having the screen freeze and restore in a subroutine is that you could have a program build several different screens for presentation in the manner of a flip-chart, where each time you press a key the program advances to the next chart. The idea here has been merely to show a working example of a program that uses BARCHART.

After you've typed and run the program, save it with

```
SAVE "CHARTSAM" ,A
```

BARCHART has now been incorporated into CHARTSAM and saved as a part of the new program, so you won't need to reLOAD BARCHART again to run CHARTSAM.

```
LOAD "BARCHART"
100 ' SALES CHART SAMPLE (CHARTSAM)
110 XS = 23: YS = 100000: YR = 10000
120 BG = 1: A1 = 2: A2 = 4: A3 = 7
130 H$ = "SALES PROJ VS ACTUAL"
140 GOSUB 1000  'SET UP SCREEN
150 ' PROJ SALES
160 BW = 1: BA = 1
170 FOR BL = 1 TO 19 STEP 6
180   READ BH
190   GOSUB 1300         'PLOT BAR
200 NEXT BL
210 '
220 ' ACTUAL SALES
230 BA = 2
240 FOR BL = 3 TO 22 STEP 6
250   READ BH
260   GOSUB 1300         'PLOT BAR
270 NEXT BL
280 GOSUB 1400: END
290 '
300 ' PROJ SALES DATA
```

(continued)

```
310 DATA 60000, 45000, 69000, 88000
320 ' ACTUAL SALES DATA
330 DATA 49000, 57000, 71000, 94000
RUN
```

Pie Charts

Another popular graphing model is the pie chart, in which portions of a whole are represented as slices of a pie. The size of each slice depends on its percentage of the whole. For example, if you surveyed a large group of people to determine the months in which they were born, a pie chart representing the results would have 12 slices of approximately equal size, since about the same number of babies are born each month of the year. A pie chart, then, is a means of reducing statistical percentages to a visual image, which is easier to understand than a complex table of numbers.

Like the BARCHART routine, PIECHART given here is a library aid that is not a complete program, but a canned routine that you can draw upon as a basis for programs that produce pie charts. It contains three services:

1. GOSUB 1000 sets up the screen for a pie chart.
2. GOSUB 1200 cuts a slice in the pie.
3. GOSUB 1400 freezes the display until a keypress and then restores it to normal operation.

The setup steps for PIECHART are simpler than those for BARCHART, since a pie chart merely depicts portions of a whole and thus requires less flexibility.

The only variable you need to load to set up the screen is "H$," which must contain the heading text for the display. Because PIECHART uses low-resolution graphics, which supports 20 characters per line of text, the heading must have fewer than that number of characters. Once you have loaded "H$," GOSUB 1000 and a white circle appears on the screen with the heading above it. The program can now begin charting percentages.

To cut a slice, load the variable "PCT" with the percentage to be represented (30% is specified as PCT = 30, *not* 0.3). Having assigned the percentage to PCT, GOSUB 1200 and the subroutine will slice a piece of the appropriate size from the pie and paint it with a color. Each slice has a different color up to 14, and then the colors begin to repeat. It is up to you to make sure that the total of the percentages passed to PIECHART is less than 100, since the routine does not check for reasonableness of the data. Each time you GOSUB 1200, the new slice begins where the last one left off.

Your program must not use variables named "TP" and "SC," which are reserved for management of the pie by the subroutine. If you accidentally alter these variables, the pie chart will come out wrong and the program might crash.

The library routine follows, and the next program shows an example of using it.

```
NEW
1000 'SUBRTN PIECHART
1010 ' SET UP SCREEN FOR PIE CHART
1020 ' PARM:  H$    SCREEN HEADER
1030 SCREEN 3:CLS:KEY OFF:RADIUS = 70
1040 WINDOW (-80,110)-(80,-90)
1050 CIRCLE (0,0),RADIUS
1060 PAINT (0,0),7,15
1070 LINE ((RADIUS * 3/5),0) - (0,0)
1080 LH = LEN(H$)
1090 LH = INT((21 - LH) / 2) + 1
1100 LOCATE 1,LH: PRINT H$
1110 RETURN
1120 '
1200 ' CUT A SLICE FROM THE PIE
1210 ' PARM:  PCT   % OF PIE TO SLICE
1220 ' RESERVED VARIABLES:
1230 '   TP      TOTAL % USED
1240 '   SC      NEXT SLICE COLOR
1250 LTP = TP: TP = TP + PCT/100
1260 DEF FNR(X)=(360 * X)*(3.141593/180)
1270 DEF FNX(X)=COS(R) * X * 3/5
1280 DEF FNY(X)=SIN(R) * X
1290 ' SLICE LINE
```

(continued)

```
1300 R=FNR(TP): X=FNX(RADIUS): Y=FNY(RADIUS)
1310 LINE (X,Y)-(0,0)
1320 ' PAINT THE SLICE
1330 XP=(TP+LTP)/2: PX=RADIUS - 7
1340 R=FNR(XP) :X=FNX(PX) :Y=FNY(PX)
1350 SC = SC + 1
1360 IF SC > 14 THEN SC = 1
1370 IF SC = 7 THEN SC = 8
1380 PAINT (X,Y),SC,15
1390 RETURN
1400 '
1410 ' FREEZE UNTIL KEYPRESS
1420 IF INKEY$ = "" THEN 1420
1430 SCREEN 0: CLS
1440 RETURN
SAVE "PIECHART",A
```

A Sample Pie Chart Program

This program will demonstrate a working example of programming with the PIECHART library routine.

Let's suppose that we've acquired information about the racial makeup of a town and we want to show it with a pie chart. The raw numbers are unimportant because we've already converted them to percentages, with the following results:

European	71.9%
Black	16.1%
Jewish	5.3%
Hispanic	3.8%
Oriental	2.2%
Other	0.7%

The first thing we have to do is bring the PIECHART routine into memory with the command

LOAD "PIECHART"

It now becomes the basis for our program, which builds upon it just as we built upon BARCHART earlier.

Now we can begin the program by assigning the title to "H$" in line 11Ø and calling the screen setup subroutine in line 12Ø.

The population data will be included in a DATA statement (line 25Ø). The list of items will end with Ø, a value that we would never put on a pie chart. Consequently, the loop in lines 15Ø–18Ø reads each item and, if it's not Ø, calls the "slicing" subroutine at line 12ØØ. When it *does* read a Ø, it jumps out of the loop to line 21Ø, freezes the display until someone presses a key, and then the program ends.

This is a very simple program, which serves to emphasize how easy it is to create pie charts using the library routine.

Save the program using the command

```
SAVE "PIESAM" ,A
```

Anytime you run this program in the future, you can type

```
RUN "PIESAM"
```

without having first to load PIECHART. This is because a copy of PIECHART has become part of the new PIESAM program.

```
LOAD "PIECHART"
100 ' PIECHART SAMPLE (PIESAM)
110 H$ = "POPULATION"
120 GOSUB 1000  ' SET UP SCREEN
130 '
140 ' POPULATION PIE CHART
150 READ PCT
160 IF PCT = 0 THEN 200
170 GOSUB 1200  ' SLICE OF PIE
180 GOTO 150
190 '
200 ' END OF RUN
210 GOSUB 1410  ' FREEZE DISPLAY
220 END
230 '
240 ' POPULATION DATA
250 DATA 71.9,16.1,5.3,3.8,2.2,.7,0
RUN
```

SECTION 8 COMPUTER GAMES

When UNIVAC produced the first commercial computers back in the early 1950s, computer pioneers immediately began talking on the radio and to newpapers about playing tic-tac-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 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.

```
NEW
100 ' ODDS AND RISK IN DICE (ODDS)
110 CLS: KEY OFF: 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 340
270 P = NC / 36
280 BET = INT((P * POT) * 100) / 100
290 PRINT "ODDS ARE" NC "IN 36"
300 I$ = "PROBABILITY = #.#######"
310 PRINT USING I$; P
320 I$ = "SAFE BET LIMIT IS $$##.##"
330 PRINT USING I$; BET
```

(continued)

```
340 PRINT: PRINT: NC = 0
350 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 = 0.05555556
        SAFE BET LIMIT IS    $2.77


        HOW MUCH IS THE POT? 0
        OK
```

Docking a Spacecraft

Space fantasies are almost synonymous with arcade-style computer games, so this is our contribution to the genre. To play it, you need a joystick attached to the "A" port (the connector closest to the right side of the machine in the rear).

Here is the situation: You are flying a spacecraft—the green square in the upper-left-hand corner of the screen—to a rendezvous with one of our space stations. That's the circle, which appears at a different location each time you play the game. Your objective is to reach the space station, signal it to open its door, and maneuver inside to safety.

The problem is that the Bad Guys have thrown a barrier around this region of space (the white border) and filled it with the explosive particles that look like stars. If you touch any particle or the barrier with your spacecraft, a terrible explosion occurs and you lose the game. Also, the space station is extremely fragile, and if you touch any part of it with your craft the same fate will befall you.

Oh, and there's one other problem, too. The space station is round and you have no way of knowing where the door is. Our people keep it closed so that particles can't enter, and they won't open it until your craft is nearly touching the station and you press the red button atop your joystick. If you're too far away, they'll ignore your signal, so you have to maneuver alongside, open the door, and the maneuver around the particles to get inside to safety. If you succeed, your reward is . . . Well, try it and find out for yourself.

Rules of play. You may move the spacecraft in any direction in order to maneuver and avoid obstacles. If the craft touches any object, including the border and the space station, you lose. The game is won by moving the spacecraft inside the space station without it touching anything. When the game ends, you can signal to play another by pressing the red button on top of the joystick housing or by typing Y in response to the question on the screen. If you don't want to play again, type N.

You can increase or decrease the difficulty of the game by changing the value of "NP" in line 15Ø. This governs the number of randomly placed particles that obstruct your progress. The higher the number, the harder the game; 3ØØ is a moderate level of difficulty, 2ØØ is easy, and 6ØØ is almost impossible.

Good luck, Astronaut.

```
NEW
100 ' DOCKING A SPACECRAFT (DOCKING)
110 ' USES JOYSTICK A
120 STRIG(0) ON: KEY OFF
130 ON STRIG(0) GOSUB 1470
140 TRUE = -1: WON = NOT TRUE
150 RANDOMIZE TIMER: NP = 300
160 DEF FNR(X)=INT((RND(1) * X) + 1)
170 '
```

(continued)

```
180 ' SET UP FOR RUN
190 JX = STICK(0)   'JOYSTICK NEUTRAL
200 JY = STICK(1)   '  POSITION
210 GOSUB 360       'OBSTACLE COURSE
220 GOSUB 610       'BUILD PLAYER
230 '
240 ' GAME LOOP
250 GOSUB 680       'STICK DIRECTION
260 IF DX=0 AND DY=0 THEN 250
270 GOSUB 760       'NEW POSITION
280 GOSUB 880       'COLLISION CHECK
290 GOSUB 1280      'MOVE THE PLAYER
300 GOSUB 1360      'WON THE GAME?
310 GOTO 240        'IF NOT, REPEAT
320 '------------------------------
330 '
340 '  SUBROUTINES
350 '
360 ' BUILD THE OBSTACLE COURSE
370 SCREEN 1,0: CLS: PALETTE 3,7
380 WINDOW SCREEN (-10,-10)-(329,209)
390 LINE (0,0)-(319,199),,B
400 FOR P = 0 TO NP
410   X = FNR(320): Y = FNR(200)
420   PSET (X,Y)
430 NEXT P
440 '
450 ' SPACE STATION
460 AX = FNR(300)
470 IF AX < 30 THEN 460
480 AY = FNR(170)
490 IF AY < 30 THEN 480
500 FOR R = 1 TO 9
510   CIRCLE (AX,AY),R,0
520 NEXT R
530 CIRCLE (AX,AY),10,3
540 D1 = (RND(1) * 2) * 3.14159
550 IF D1 < 0 THEN 540
560 D2 = D1 + 1.2
570 IF D2 > 6.2831 THEN 540
```

```
580 RETURN
590 ' --------------------------
600 '
610 ' BUILD "THE PLAYER"
620 CX = 5: LX = CX + 6
630 CY = 5: LY = CY + 6
640 LINE (CX,CY)-(LX,LY),1,BF
650 RETURN
660 ' --------------------------
670 '
680 ' GET JOYSTICK DIRECTION
690 X = STICK(0) - JX: DX = 0
700 Y = STICK(1) - JY: DY = 0
710 IF ABS(X)>10 THEN DX = SGN(X)
720 IF ABS(Y)>10 THEN DY = SGN(Y)
730 RETURN
740 ' --------------------------
750 '
760 ' POSITION AHEAD OF PLAYER
770 '  (BASED ON JOYSTICK DIRECTION)
780 IF DX =  0 THEN VX = CX: GOTO 810
790 IF DX =  1 THEN VX = CX + 7
800 IF DX = -1 THEN VX = CX - 1
810 VY = CY + DY: HX = CX + DX
820 IF DY =  0 THEN HY = CY: GOTO 850
830 IF DY =  1 THEN HY = CY + 7
840 IF DY = -1 THEN HY = CY - 1
850 RETURN
860 ' --------------------------
870 '
880 ' CHECK FOR COLLISION
890 IF DX = 0 THEN 930
900 FOR Y = VY TO VY+6
910   IF POINT(VX,Y)=3 THEN 990
920 NEXT Y
930 IF DY = 0 THEN 970
940 FOR X = HX TO HX+6
950   IF POINT(X,HY)=3 THEN 990
960 NEXT X
```

(continued)

```
970 RETURN     'NO COLLISION
980 '
990 ' THE GAME IS LOST
1000 SOUND ON: BEEP OFF
1010 FOR C = 1 TO 10
1020   COLOR 4
1030   NOISE 6,15,10
1040   FOR T = 1 TO 50: NEXT T
1050   NOISE 6,15,10
1060   COLOR 15
1070   FOR T = 1 TO 50: NEXT T
1080 NEXT C
1090 SOUND OFF: BEEP ON
1100 '
1110 ' RESTORE SCREEN
1120 ON STRIG(0) GOSUB 1540
1130 SCREEN 0: CLS: PRINT: PRINT: PRINT
1140 IF NOT WON THEN 1160
1150 PRINT "YOU WON!": GOTO 1170
1160 PRINT "SORRY, YOU LOST"
1170 PRINT: PRINT
1180 PRINT "WANT TO PLAY AGAIN?"
1190 PRINT " IF NO, TYPE N"
1200 PRINT " IF YES, TYPE Y OR ELSE"
1210 PRINT " PRESS THE JOYSTICK BUTTON"
1220 X$ = INKEY$: IF X$="" THEN 1220
1230 IF X$ = "Y" THEN GOTO 100
1240 PRINT "N"
1250 END
1260 ' ----------------------------
1270 '
1280 ' ADVANCE THE PLAYER PER JOYSTICK
1290 NX = CX + DX: NY = CY + DY
1300 LINE (CX,CY)-(CX+6,CY+6),0,BF
1310 LINE (NX,NY)-(NX+6,NY+6),1,BF
1320 CX = NX: CY = NY
1330 RETURN
1340 ' -------------------------------
1350 '
1360 ' CHECK IF INSIDE SPACE STATION
```

```
1370 IF ABS(CX-AX) > 5 THEN 1390
1380 IF ABS(CY-AY) < 5 THEN 1410
1390 RETURN     'IF NOT DOCKED
1400 '
1410 ' THE GAME IS WON
1420 PLAY "O2 T200 L8 G>C EG4 EG2"
1430 WON = TRUE
1440 GOTO 1110
1450 ' ---------------------------
1460 '
1470 ' OPEN STATION DOOR IF NEARBY
1480 IF ABS(CX-AX) > 35 THEN 1510
1490 IF ABS(CY-AY) > 35 THEN 1510
1500 CIRCLE (AX,AY),10,0,D1,D2
1510 RETURN
1520 ' ---------------------------
1530 '
1540 ' BUTTON PRESSED TO REPEAT GAME
1550 X$ = "Y"
1560 RETURN 100
1570 ' ---------------------------
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 from –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 decimals, the program rounds the answer).

Play this game as follows:

1. Start the program by typing RUN.
2. The computer displays the instructions.

3. Press the ENTER key (or any other except S) to begin the game.
4. The computer poses a problem and asks you to type the answer. It then tells you if you were right or, if you gave the wrong answer, what you should have typed.
5. Press ENTER 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 6Ø%, and the grade rises by one letter for each 1Ø% above that (6Ø% and above is a D, 7Ø% 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 time out 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:

```
140 DEF FNR(X) = 10 - INT(RND(0))
```

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

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

You can control the number of decimal places to which the answers are rounded by changing the value of "NP" in line 11Ø: "NP = 3" means three places, "NP = 4" means four places, and so on. Finally, to change the grading system so that it corresponds with yours, replace the "cut points" in lines 1Ø1Ø–1Ø4Ø. For example, if you give an A for 95% or better, change the 9Ø in line 96Ø to 95.

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

```
NEW
100 ' MATH DRILL (MATHDRIL)
110 NP = 3: RV = 10^NP
120 RANDOMIZE TIMER
130 DEF FNA(X) = FIX(X * RV) / RV
140 DEF FNR(X) = 100-INT(RND(1) * 200)
150 CLS: KEY OFF
160 PRINT "MATH DRILL:": PRINT
170 PRINT "COMPUTER GIVES A PROBLEM.";
180 PRINT "  YOU TYPE THE"
190 PRINT "ANSWER. IF YOU ARE RIGHT,";
200 PRINT " THE COMPUTER"
210 PRINT "TELLS YOU SO, AND IF YOU";
220 PRINT " GIVE THE WRONG"
230 PRINT "ANSWER, THE COMPUTER TELLS";
240 PRINT " YOU WHAT THE"
250 PRINT "CORRECT ANSWER IS."
260 PRINT " AFTER EACH EXERCISE, TAP";
270 PRINT " 'ENTER' TO"
280 PRINT "CONTINUE OR";
290 PRINT " THE LETTER 'S' TO STOP."
300 PRINT "THE COMPUTER WILL THEN ";
310 PRINT "GRADE YOUR PER-"
320 PRINT "FORMANCE."
330 PRINT " DECIMALS ARE CARRIED OUT ";
340 PRINT "TO" NP "PLACES."
350 GOTO 610
360 ' -------------------------------
370 '
380 ' CONTINUE/STOP QUERY
390 PRINT "S TO STOP, 'ENTER' FOR NEXT"
400 X$=INKEY$: IF X$ = "" THEN 400
410 IF X$ = "S" THEN 920
420 RETURN
430 ' -------------------------------
440 '
450 ' SELECT OPERATION
460 M = INT(RND(1) * 5)
470 IF M < 1 THEN 460
```

(continued)

```
480 RETURN
490 ' ---------------------------------
500 '
510 ' COMPUTATION
520 N1 = FNR(0): N2 = FNR(0)
530 ON M GOTO 540, 550, 560, 570
540 A = N1 + N2: GOTO 580
550 A = N1 - N2: GOTO 580
560 A = N1 * N2: GOTO 580
570 A = FNA(N1 / N2)
580 RETURN
590 ' ---------------------------------
600 '
610 ' ***  MAIN PROGRAM  ***
620 C = 0: I = 0
630 PRINT:PRINT "TAP 'ENTER' TO BEGIN"
640 GOSUB 400
650 '
660 ' MAIN LOOP
670 CLS: LOCATE 1,14
680 PRINT "MATH DRILL:"
690 PRINT "  RIGHT:" C; TAB(25);
700 PRINT "WRONG:" I
710 PRINT: PRINT
720 PRINT "PROBLEM:"
730 GOSUB 450: GOSUB 510
740 PRINT TAB(5) N1;
750 ON M GOTO 760, 770, 780, 790
760 PRINT " + ";: GOTO 800
770 PRINT " - ";: GOTO 800
780 PRINT " x ";: GOTO 800
790 PRINT " / ";
800 PRINT N2: PRINT: PRINT
810 INPUT "ANSWER"; Y: PRINT
820 Y = FNA(Y)
830 IF Y <> A THEN 860
840 PRINT "THAT'S RIGHT!"
850 C = C + 1: GOTO 880
860 PRINT "SORRY, THE ANSWER IS" A
870 I = I + 1
```

```
880 PRINT: PRINT: GOSUB 380
890 GOTO 660
900 ' ------------------------------
910 '
920 ' GRADE PERFORMANCE
930 CLS: LOCATE 2,14
940 PRINT "YOUR SCORE:"
950 PRINT: PRINT: PRINT
960 PRINT "   NUMBER RIGHT:   " C
970 PRINT "   NUMBER WRONG:   " I
980 T = C + I: A = FNA(C / T * 100)
990 PRINT "   PERCENT RIGHT: " A
1000 G$ = "F"
1010 IF A >= 60 THEN G$ = "D"
1020 IF A >= 70 THEN G$ = "C"
1030 IF A >= 80 THEN G$ = "B"
1040 IF A >= 90 THEN G$ = "A"
1050 PRINT
1060 PRINT "   YOUR GRADE IS " G$
1070 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 10 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 ' SHOOTING GALLERY (GALLERY)
110 RANDOMIZE TIMER
120 SOUND ON: BEEP ON: KEY OFF
130 SCREEN 1: CLS
140 '
150 ' SET UP GALLERY
160 WINDOW (-5,-5)-(84,205)
170 COLOR 9,1
180 PALETTE 1,6
190 LINE (60,98)-(79,102),1,BF
200 LINE (-1,0)-(79,199),3,B
210 LINE (-2,0)-(-2,199)
220 PAINT (0,200),2,3
230 '
240 ' CONTROL VALUES
250 H = 0: M = 0     :' HITS, MISSES
260 BR = 10          :' BULLETS REMAINING
270 GOSUB 1380       :' DISPLAY SCORE
280 DEF FNR(X) = INT(RND(1) * 45) + 4
290 '
300 ' *** MAIN PROGRAM ***
310 BX = 200: BI = 0: FIRED = 0
320 GOSUB 650       :' START TARGET
330 '
340 ' ACTION LOOP
350 IF FIRED THEN 370
360 GOSUB 980       :' CHECK IF FIRING
370 GOSUB 760       :' ADVANCE TARGET
380 GOSUB 870       :' AT WALL YET?
390 GOSUB 1080       :' BULLET IMPACT?
400 GOSUB 1310       :' ADVANCE BULLET
410 GOTO 340
420 '
430 ' ACTION FINISHED
440 IF BR <= 0 THEN 480
450 GOSUB 1380       :' DISPLAY SCORE
460 GOTO 300         :' REPEAT FOR NEXT
470 '
```

```
480 ' END OF GAME
490 SCREEN 0: CLS
500 LOCATE 3,14: PRINT "GAME'S OVER"
510 LOCATE 6,14: PRINT "HITS     " H
520 LOCATE 7,14: PRINT "MISSES   " M
530 W$ = "TERRIFIC"
540 IF H < 8 THEN W$ = "GOOD"
550 IF H < 5 THEN W$ = "FAIR"
560 IF H < 3 THEN W$ = "LOUSY"
570 LOCATE 9,11: PRINT "YOU'RE A ";
580 PRINT W$ " SHOT"
590 LOCATE 11,10
600 INPUT "ANOTHER GAME (Y/N)"; X$
610 IF X$ = "Y" THEN 100
620 CLS: END
630 '-------------------------------
640 '
650 ' START A TARGET
660 X1 = FNR(0): X2 = X1 + 2
670 IF (X1 MOD 2) = 1 THEN 710
680 TY = 191: MI = -2
690 LINE (X1,TY)-(X2,TY+7),1,BF
700 GOTO 730
710 TY =   8: MI =  2
720 LINE (X1,TY)-(X2,TY-7),1,BF
730 RETURN
740 '-------------------------------
750 '
760 ' ADVANCE TARGET
770 IF MI < 0 THEN 800
780 Y1 = TY-5: Y2 = TY-7: Y3 = TY-6
790 GOTO 810
800 Y1 = TY+5: Y2 = TY+7: Y3 = TY+6
810 TY = TY + MI
820 LINE (X1,TY)-(X2,Y1),1,BF
830 LINE (X1,Y2)-(X2,Y3),0,BF
840 RETURN
850 '------------------------------
860 '
```

(continued)

```
870 ' CHECK IF TARGET HAS CROSSED
880 IF MI > 0 THEN 900
890 IF TY < 2 THEN 930 ELSE 910
900 IF TY > 197 THEN 930
910 RETURN          :' NOT AT WALL YET
920 '
930 ' TARGET IS AT THE WALL
940 LINE (X1,TY)-(X2,Y2),0,BF
950 RETURN 430      :' ACTION FINISHED
960 '-------------------------------
970 '
980 ' CHECK TRIGGER, FIRE IF PUSHED
990 IF INKEY$ = "" THEN 1050
1000 ' FIRE THE BULLET
1010 BX = 58: BI = -2
1020 BR = BR - 1: FIRED = -1
1030 PSET (BX,100)
1040 NOISE 6,15,1
1050 RETURN
1060 '-------------------------------
1070 '
1080 ' CHECK IF BULLET HAS STRUCK
1090 A = POINT(BX-1,100)
1100 IF A = 1 THEN GOSUB 1140
1110 IF A = 3 THEN GOSUB 1220
1120 RETURN
1130 '
1140 ' HIT TARGET
1150 H = H + 1      :' SCORE A HIT
1160 NOISE 3,15,4
1170 GOSUB 1380
1180 PRESET (BX,100)
1190 LINE (X1,TY)-(X2,Y2),0,BF
1200 RETURN 430     :' DISPLAY SCORE
1210 '
1220 ' HIT THE WALL
1230 M = M + 1      :' SCORE A MISS
1240 NOISE 3,15,4
1250 GOSUB 1380
1260 PRESET (BX,100)
```

```
1270 BX = 200: BI = 0
1280 RETURN            :' RESUME
1290 '--------------------------------
1300 '
1310 ' ADVANCE THE BULLET
1320 BX = BX + BI
1330 PSET (BX,100)
1340 PRESET (BX-BI,100)
1350 RETURN
1360 '--------------------------------
1370 '
1380 ' DISPLAY THE SCORE
1390 LOCATE 2,30
1400 PRINT "HITS   " H
1410 LOCATE 3,30
1420 PRINT "MISSES " M
1430 RETURN
1440 '--------------------------------
RUN
```

SECTION 9
SOUND EFFECTS AND COMPUTER MUSIC

The IBM PC*jr* has a three-voice synthesizer capable of making sounds ranging from strange effects to quite sophisticated music in three-part harmony. What we'll give you in this section is but a sampling of its capabilities. Yet some—the Bach invention in particular—are quite remarkable and demonstrate the artistic heights to which you can rise.

Few computers offer such wide-ranging sound capabilities (most offer none at all), with the result that software has heretofore been mostly silent. Not any longer; let your imagination take off after you try these programs and hear for yourself what you can do to enliven your own programs.

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 mere 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.

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 ' EMERGENCY! (EMRGNCY)
110 SOUND ON
120 FOR T = 200 TO 500 STEP 5
130   SOUND T, .5
140 NEXT T
150 FOR T = 500 TO 1000 STEP 3
160   SOUND T, .5
170 NEXT T
180 FOR T = 1000 TO 500 STEP -3
190   SOUND T, .5
200 NEXT T
210 IF INKEY$ = "" THEN 150
220 FOR T = 500 TO 37 STEP -3
230   SOUND T, .5
240 NEXT T
250 SOUND OFF
RUN
```

Sounds for Late at Night

The fact that I wrote this program at about 2 o'clock in the morning might have a bearing on its name. Nevertheless, it'll send shivers down your spine at any hour. Some of the sounds resemble the weird sizzling and the "Russian woodpecker" familiar to radio hams, and others seem to come from Deep Space and the macabre imaginations of people who make sci-fi horror films. Let your own imagination run wild.

On a more practical note, this program runs through eight sound effects and keeps repeating them. If you tap any key, it will stop when it has completed the set of eight.

```
NEW
100 ' SOUNDS FOR LATE AT NIGHT (CREEPY)
110 FOR N = 0 TO 7
```

(continued)

```
120    SOUND ON
130    FOR V = 0 TO 15
140      NOISE N,V,10
150    NEXT V
160    FOR T = 1 TO 100: NEXT T
170    FOR V = 15 TO 0 STEP -1
180      NOISE N,V,10
190    NEXT V
200    FOR T = 1 TO 8000: NEXT T
210 NEXT N
220 IF INKEY$ = "" THEN 110
230 SOUND OFF: END
RUN
```

R2D2

Everybody loves R2D2, that cute little robot from the *Star Wars* films. We can't understand what he says, but that doesn't matter; 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." After you start the program, R2D2 "talks" in phrases of varying lengths until you tap a key, and then he'll finish what he's saying before he falls silent.

```
NEW
100 ' R2D2
110 RANDOMIZE TIMER: SOUND ON
120 DEF FNR(X)=INT(RND(1) * X) + 1
130 '
140 ' TALK TO ME
150 N = FNR(32): A$ = "T200 L16 ML"
155 IF N < 5 THEN 150
160 FOR L = 1 TO N
170   V = FNR(84)
180   A$ = A$ + "N" + STR$(V)
190 NEXT L
```

```
200 PLAY A$
210 '
220 ' PAUSE FOR BREATH
230 FOR T = 1 TO 1000: NEXT T
240 IF INKEY$ = "" THEN 140
250 END
RUN
```

Nursery Song

One of the great strengths of the PC*jr*'s synthesizer is the musical "sublanguage," which enables you to transcribe music directly from a score or from experimentation on a keyboard instrument such as a piano. The musical notation is not difficult to master if you read music (and not bad even if you don't). The secret is in the creative use of DATA statements to contain all the music; a simple program with minor variations can then play anything from "Mary Had a Little Lamb" all the way up to classics and baroque. The next four programs demonstrate just that.

```
NEW
100 ' MARY HAD A... (LILAMB)
110 SOUND ON
120 READ A$: PLAY A$
130 PLAY ON
140 ON PLAY(1) GOSUB 170
150 GOTO 140
160 '
170 ' MEASURE OF MUSIC
180 READ A$
190 IF A$ = "99" THEN 240
200 PLAY A$
210 RETURN
220 '
230 ' STOP THE MUSIC
240 END
250 '
260 ' THE SCORE FOLLOWS
```

(continued)

```
270 DATA "O2 T150 L4 MS EDCD"
280 DATA "EEE2", "DDD2"
300 DATA "EGG2", "EDCD"
310 DATA "EEEE", "DDEDC1"
320 DATA "99"
RUN
```

Bugle Call

Aficionados 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 that the same program can be used over and over to make music; the only differences between this and the previous program are line 100 and the DATA statements that contain the actual musical notation. Both of these programs use only one of the synthesizer's three voices.

```
NEW
100 ' SPORT OF KINGS (BUGLE)
110 SOUND ON
120 READ A$: PLAY A$
130 PLAY ON
140 ON PLAY(1) GOSUB 170
150 GOTO 140
160 '
170 ' MEASURE OF MUSIC
180 READ A$
190 IF A$ = "99" THEN 240
200 PLAY A$
210 RETURN
220 '
230 ' STOP THE MUSIC
240 END
250 '
260 ' THE SCORE FOLLOWS
270 DATA "O2 T150 L8 MS G>"
280 DATA "CE G8 L12 GGG"
300 DATA "E8 EEE L8 CEC <G2"
310 DATA "G>CEG L12 GGG"
```

```
320 DATA "L8 GEC<G L12 GGG >C1"
330 DATA "99"
RUN
```

Who Says Classical Music is Stuffy?

Enough of this elementary stuff! Let's make some real music, as written by one of the greatest composers of all time.

A lot of people seem to have the idea that Johann Sebastian Bach was a longhair who wrote music to put concert goers to sleep. A lot of people are wrong. Bach was an enormously versatile man who did indeed write some very serious music, but who was also the Duke Ellington or John Lennon of his time. In fact, many of his compositions are the basis for jazzy pieces by people such as Barry Manilow, and the unadulterated Bach stands up well with much of today's hard rock and catchy rhythms.

Take this piece as an example. It's the first section of Bach's Two-Part Invention No. 8, originally written for the harpsichord in the early 1700s. The term "two-part" doesn't mean two sections, but instead that only two notes are ever played at the same time. Listening to it, you'll probably have difficulty believing that, but it's true. This piece uses two of the synthesizer's three voices, and so the program differs slightly in the "play a measure" subroutine from the two programs that precede it.

There are a lot of DATA statements, for the simple reason that the Invention is crammed with 16th notes and, by dint of using two voices, it takes twice as many notes to make a piece. Still, it's well worth the keying just to hear the result.

```
NEW
100 ' BACH INV #8 (BACH8)
110 DEF FNC(X$) = INT((40-LEN(X$))/2)+1
120 CLS: KEY OFF
130 SCREEN 1: COLOR 1
140 LINE (64,56)-(248,112),,B
150 X$ = "TWO-PART INVENTION #8"
```

(continued)

```
160 LOCATE 9,FNC(X$): PRINT X$
170 LOCATE 11,19: PRINT "BY"
180 X$ = "JOHANN SEBASTIAN BACH"
190 LOCATE 13, FNC(X$): PRINT X$
200 '
210 PLAY ON: SOUND ON
220 GOSUB 260
230 ON PLAY(2) GOSUB 260
240 GOTO 230
250 '
260 ' MEASURE OF MUSIC
270 READ T$
280 IF T$ = "99" THEN 320
290 READ B$
300 PLAY B$,T$
310 RETURN
320 ' STOP THE MUSIC
330 PLAY OFF
340 FOR T = 1 TO 1000: NEXT T
350 SCREEN 0: CLS: END
360 '
370 DATA "O2 V15 T120 MS P8 L8 FAF>C<F"
380 DATA "O1 V12 T120 MS L8"
390 DATA ">F L16 EDCDC<B-AB-AG"
400 DATA "P8      F A F  >C <F"
410 DATA "L8F       A>C<A>   F  C"
420 DATA " >F L16 ED CDC<B- AB-AG"
430 DATA "L16 A>C<B->C"
440 DATA "L8  F   A"
450 DATA "<A>C<B->C <A>C<B->C"
460 DATA ">C  <A    >F   C L16"
470 DATA "<FAGA FAGA FAGA"
480 DATA "F>C<B->C<A>C<B->C<A>C<B->C<"
490 DATA "DFEF DFEF DFEF"
500 DATA "FAGA FAGA FAGA"
510 DATA "L8<B G >D<B> F D"
520 DATA "    DFEF DFEF DFEF"
530 DATA "L16 GAGF EFED CDC<B-"
540 DATA "L8 <B G >C<G >E C   L16"
550 DATA "A8>DC <B>C<BA  G A GF"
```

```
560 DATA "FG FE  D E DC <B>C<BA"
570 DATA "EF E D C8 >C<B>C8< E8"
580 DATA "G8>C<B AB  A G FG  FE"
590 DATA "L8 F >C <E >C <D   B"
600 DATA "   DE DC GF EF G8 <G8"
610 DATA ">C4"
620 DATA ">>C4
630 DATA "99"
RUN
```

Patriot

It seems appropriate to end this section and this book with an all-out salvo, sort of like the ending of the Fourth of July fireworks show. So here it is: computer graphics and three-part harmony, together with a patriotic salute to the country that fostered the technology that made this fantastic computer available to—and affordable by—you.

Hint: Load the BUNTING program from disk and start typing this program at line 330. Except for line 100, it's all the same down to that point.

```
NEW
100 ' PATRIOT
110 SCREEN 3: CLS: KEY OFF
120 WINDOW (-80,100)-(80,-100)
130 LINE (-40,30)-(40,10),12,BF
140 LINE (-40,10)-(40,-10),15,BF
150 LINE (-40,-10)-(40,-30),9,BF
160 '
170 ' CIRCLE OF STARS
180 RADIUS = 50
190 FOR ANGLE = 90 TO 450 STEP 12
200   R = ANGLE * (3.141593 / 180)
210   X = COS(R) * RADIUS
220   Y = SIN(R) * RADIUS * 5/3
```

(continued)

```
230   PSET (X,Y)
240 NEXT ANGLE
250 '
260 ' STARS IN BUNTING
270 FOR X = -30 TO 30 STEP 10
280   FOR Y = 20 TO -20 STEP -40
290     PSET (X, Y)
300   NEXT Y
310 NEXT X
320 '
330 ' MY COUNTRY 'TIS OF THEE
340 SOUND ON: PLAY ON
350 READ S$, A$, B$
360 PLAY B$,A$,S$
370 ON PLAY(2) GOSUB 400
380 GOTO 370
390 '
400 ' MUSIC IN 3 PARTS
410 READ S$
420 IF S$ = "99" THEN 470
430 READ A$, B$
440 PLAY B$, A$, S$
450 RETURN
460 '
470 ' END OF RUN
480 FOR T = 1 TO 2000: NEXT T
490 SCREEN 0: CLS: END
500 '---------------------------------
510 '
520 ' MUSICAL SCORE
530 DATA "O2 ML T90 V15 L4 F F G"
540 DATA "O1 MN T90 V12 L4 C>D D"
550 DATA "O1 MN T90 V12 L4 F D<B-"
560 '
570 DATA " E. F8 G  A  A  B-"
580 DATA " C. D8 E  F  F  G"
590 DATA ">C. C8 C  F  D <B-"
600 '
610 DATA " A. G8 F  G  F  E F4.G8A8B-8"
620 DATA " F. E8 F  D  C  C C2."
630 DATA ">C. C8 D <B->C  C F2."
```

```
640 '
650 DATA ">C  C  C  C.<B-8A"
660 DATA " A  A  A  A. G8 F"
670 DATA " F  A >C <F. F8 F"
680 '
690 DATA " B- B- B- B-.A8 G"
700 DATA " G  G  G  G. F8 E"
710 DATA " C  E  G  C. C8 C"
720 '
730 DATA " A L8 B-A G F L4"
740 DATA " F L8 D C<B-A>L4"
750 DATA " F     F    F"
760 '
770 DATA " A. B-8 >C D8 <B-8A G    F2."
780 DATA " F. F8   F F8  G8 F E<ML A2."
790 DATA " F. G8   A B-8>D8 C<C ML F2."
800 DATA "99"
RUN
```

SOME OF THE THINGS YOU CAN DO WITH KENT PORTER'S PROGRAMS— AND YOUR IBM® PC*jr*

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

ALPHABETIZE LONG LISTS

PLAY YOUR OWN VIDEO GAMES

TURN YOUR COMPUTER INTO A DIGITAL TIMEPIECE AND CALENDAR

USE THE COLOR CAPACITY OF THE IBM PC*jr*

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 IBM® PC*jr*

82088

0 71162 00695

ISBN 0-451-82088-6