$14.95

From the Publishers of **COMPUTE!** Magazine

Programmer's Reference Guide to the TI-99/4A

C. Regena

Clear explanations of BASIC TI-99/4A™ programming plus dozens of programs you can type in and run.

COMPUTE!™ Publications, Inc. abc
One of the ABC Publishing Companies

From the Publishers of **COMPUTE!** Magazine

Programmer's Reference Guide to the TI-99/4A™

C. Regena

COMPUTE!™ Publications, Inc. abc
A Subsidiary Of American Broadcasting Companies, Inc.
Greensboro, North Carolina

When You Type In Programs . . .

You may encounter braces enclosing a specified number of spaces, i.e.:

{ 4 SPACES }

In these (and only these) instances, type the appropriate number of spaces, but do *not* type the braces.

Printed in the United States of America.

ISBN 0-942386-12-4

10 9 8 7 6 5 4 3 2

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is an independent publisher of quality consumer products for the personal computer industry, and is not associated with any manufacturer of personal computers. TI-99/4 and TI-99/4A are trademarks of Texas Instruments.

Table of Contents

Preface

You have your TI-99/4 or TI-99/4A home computer — now what can you do with it? In this book I present TI BASIC programming explanations and techniques and a great variety of programs for you to key-in and RUN.

I would like to offer a special thanks to Richard Mansfield and Kathleen Martinek for their encouraging words and their confidence in me. Thanks also to Robert Lock, President of Small System Services, Inc.; Ottis Cowper, technical editor; and other members of the production staff at *COMPUTE!* Magazine and Small System Services, Inc.

I also acknowledge my husband, Chandler Whitelaw, for his patience while I was writing or programming and would answer his queries with ''Uh-huh'' or ''just one more thing here . . . '' I also appreciate my own in-house quality control department — Chery, Richard, Cindy, Bob, and Randy — for their help in testing programs and keeping me supplied with new ideas.

To: The Texas Instruments 99/4A User Community

**From: Robert Lock, Editor In Chief/Publisher,
COMPUTE! Publications, Inc.**

When C. Regena began writing a tutorial applications and programming column in COMPUTE! Magazine, we took our first steps toward an ongoing commitment to the support of the owners and users of personal computer products from Texas Instruments. We're doubly pleased to be introducing the *Programmer's Reference Guide* as the first book solely for TI from our COMPUTE! Books Division.

From "Getting Started" to "Programming Techniques," you'll find this guide an invaluable tool. The author is an experienced teacher and tutor for the range of users from beginner to most advanced. You'll find dozens of applications, tutorials, fully documented programs, and utilities designed to enhance and support your full utilization of the power of your personal computer.

Whether you use the book as simply a source of a tremendous amount of ready-to-run software, or as an equally valuable reference guide, you'll enjoy the easy-to-use format and the quality of the writing. Welcome to COMPUTE! Books.

Our thanks to the many members of our production and editorial staffs who assisted in the development of this guide.

Chapter 1

Introduction

Introduction

The Texas Instruments 99/4 home computer cost over $1000 in 1980 and included a TI color monitor. Soon the computer was sold separately from the monitor for about $600. One of its most appealing features was the plug-in command modules for programs, which made the computer very easy to use.

In late 1981, after making improvements on the TI-99/4 calculator-style keyboard, Texas Instruments introduced the new TI-99/4A console and lowered the price to under $500.

Within a few months, most dealers offered a price near $400. In September 1982, Texas Instruments changed its advertising campaign for the TI-99/4A and started offering a rebate of $100. Many stores were selling the TI-99/4A for $295, so it was possible to own this very powerful computer for about $200.

By now, more than 75 command modules are available. You can do anything from playing a video game to keeping track of a school district's accounts, simply by using the command modules.

It's Time for You to Take Command

Valuable as those pre-packaged programs are, you will enjoy your computer even more when you write your own programs with the built-in TI BASIC computer language. This book will help you design your programs, provide you some hints and techniques, and remind you of good programming habits. Best of all, there are some actual programs for you to try.

To use this book, all you need is your TI-99/4 or TI-99/4A computer and a color monitor or television set with the appropriate cables. You will probably want a cassette recorder and the dual cassette cable so that you can save the programs you write.

If you sit at the computer while you're reading this book, then you can type in each sample program as you come to it and RUN the program to see what happens. On many of the programs, you may experiment by changing some of the numbers in the commands to see how the program changes.

All the programs in this book use TI BASIC, the BASIC language that is built into the TI-99/4 or TI-99/4A console; no other memory or language cartridges are necessary.

Differences between the TI-99/4 and the TI-99/4A

This book is written for users of the TI-99/4 or TI-99/4A home computers. For simplicity's sake, I will usually use "TI-99/4A" to refer to *both* computers, since TI BASIC is the same for both.

The main differences between the TI-99/4 and the TI-99/4A consoles are:

1. The TI-99/4 has a "calculator-style" keyboard. The TI-99/4A's keyboard is more like a typewriter.

2. The placement of several keys and symbols is different. On the TI-99/4, you use SHIFT plus a key for many of the functions; on the TI-99/4A, you press the FUNCTION key plus another key for the functions.

3. The TI-99/4A has an ALPHA LOCK key and shifted and unshifted letters. The letters are actually large and small capital letters; but if you use a printer with lowercase letters the small capital letters will turn out to be lowercase letters.

4. The TI-99/4 has an Equation Calculator option that the TI-99/4A does not have. Essentially, the Equation Calculator allows you to evaluate mathematical expressions without actually writing a program.

5. The TI-99/4A has 256 fewer bytes of Random Access Memory (RAM) than the TI-99/4. If you have a program that uses almost all of the memory on a TI-99/4, it may not work on the TI-99/4A.

6. Some of the key codes returned by a CALL KEY statement are different on the two consoles.

Special Features of the TI-99/4A

I'll briefly review some of the advantages of the TI-99/4A. Later chapters will show you in some detail how you can use some of these features in your own programs. Don't worry if some of these features aren't yet clear to you. By the time you've read the appropriate chapters, everything will make sense.

Graphics and color. Probably one of the most enjoyable things to do with your computer is drawing pictures. There are 16 colors, and you can use all of them on the screen at the same time, even in high-resolution graphics. *High-resolution* means more detailed drawing. You can easily create your own high-resolution graphics characters, and you can also use text (words) anywhere on the screen at the same time you use high-resolution graphics (drawings). Many other microcomputers limit your use of text with high-resolution graphics and limit the number of colors you can use with higher resolutions.

Music. You may play up to three notes and one noise for a specified time using *one* statement. The musical tones are selected by using a number which represents a frequency of 110 Hz to 44733 Hz, which is a tone from low A on the bass clef to out of human hearing range. The tone may be between regular musical notes.

Noises. Using different combinations of musical tones and noise numbers, you can make all sorts of synthesized noises — everything from crashes and explosions to outer-space tones.

Combining sounds and graphics. "Computer choreography" is possible because, while music is played, other statements (including graphics) may be executed. You may illustrate a song, for example. If you have a game program, you may make calculations while you are making a noise.

Built-in programming language. TI BASIC is built into the main console — there's nothing extra to buy. TI BASIC is an excellent language to learn how to program; it is easy enough for a beginner, yet powerful enough for an experienced programmer because of the built-in functions.

Speech. Even though speech is not built-in, I am going to include it in this list of features. Since Texas Instruments offered a TI Speech Synthesizer free with the purchase of six command modules (for about eight months), many TI owners have a speech synthesizer. The speech synthesizer is a small box that attaches to the side of your computer console. The speech feature is relatively inexpensive and very easy to use. Plug in any of the command modules that contain speech, such as the game of *Parsec* or any of the Scott, Foresman educational command modules, and you can hear the computer speak to you. Other command modules are available for you to program your own speech.

16-bit microprocessor. The TI-99/4A uses the TI-9900 16-bit microprocessor, which offers more computing power and greater expansion and configuration flexibility than an 8-bit microprocessor. You can achieve higher numeric precision, simplified memory addressing, and impressive efficiency.

Plug-in modules. The easiest way to use the TI-99/4A is to insert a command module which contains a program. Modules are available for a variety of applications. The price depends on the amount of memory built into the module. The modules actually *add memory* to the computer.

Variable naming. In your own programming, you may use *meaningful* variable names. In many microcomputers, the BASIC language recognizes only two letters or a letter and a number for a variable name. If you have a program with the variable name BLUE and another variable name BLACK, other computers may think they are the same variable, *BL*, but the TI-99/4A knows you are using two variables. You also do not have to worry about embedded reserved words in variable names. For example, many computers would not allow the variable name AFFORD because it contains the word FOR. The TI doesn't mind.

String manipulation. TI BASIC offers powerful string operations. Your computer has two ways of interpreting the letters and numbers you enter from the keyboard. Usually, the computer assumes you are entering commands and numbers, to perform mathematical operations. However, when you tell it to, the TI-99/4A can also interpret letters, numbers, and symbols as *strings*. You would enter a list of names and addresses, for instance, as strings. It wouldn't make any sense for the computer to add up your friends' house numbers, or treat their names as numeric variables. But you might very well want to arrange those names in alphabetical order. That is just one example of a string operation that TI BASIC can perform.

It can also find out the length of a word or phrase, search for one group of letters contained within another, or cut up words or phrases into smaller segments. And just as you can use numeric variables to stand for numbers, you can use string variables to represent strings. With TI BASIC you can even use string variables in arrays.

Line editing. Programmers will enjoy the easy line editing features. Function keys allow you to change, insert, or delete characters without retyping the entire line.

Automatic line numbering. You may specify a beginning line number and an increment, and the computer will automatically number your lines for you as you are typing them in.

Automatic renumbering. After you have programmed and added or deleted statements here and there, the automatic resequencing command, RES, will automatically renumber your statements, including all statement numbers referenced by other commands.

Trace. If you use the TRACE command, TI BASIC will follow the line numbers of statements as they are being executed to help you in debugging programs. You may stop the program at any time and print out the value of any variables.

Peripherals

Unless otherwise specified, none of the programs in this book require extra equipment. However, to give you an idea of the capabilities and expandability of your TI computer, I will briefly describe peripherals you can add on to your basic console. Keep in mind that improvements and enhancements are constantly being developed and that prices fluctuate.

Software. Your computer is *hardware; software* is the programs that will make the computer do what you want it to do. The easiest way to load a program into the TI computer is to use a command module. Just plug it in.

Another way to load a program is to type (''key in'') the program each time you wish to use it. If a program is long, you'll find that it saves a lot of time to store it on a cassette or a diskette. Most ''third-party'' (not produced by Texas Instruments) software is produced on cassette or diskette. A cassette program requires a cassette recorder and the dual cassette cable. A diskette program requires a disk drive and the disk controller.

Software is available for a variety of applications, like games, education, finance, inventory, engineering, business, and music.

When you purchase software, the literature or your dealer should tell you what hardware is required. For example, business software often requires a printer and two disk drives (and thus the peripheral box, RS-232 interface, and disk controller), plus perhaps the Extended BASIC module and maybe the 32K memory expansion. Some game programs require joysticks.

Cassette recorder and cassette cable. Probably one of the first items you will need is a cassette cable to connect a cassette recorder to the computer to save your own program or to load other cassette programs for your use. Nearly any cassette recorder is acceptable; however, the TI-99/4A is more critical on volume control than the TI-99/4 is, and some brands work better than others. In general, a battery-operated recorder will not work well enough for accurate data retrieval all the time. Also, your recorder should have both a tone and volume control. Texas Instruments publishes a list of recommended cassette recorders.

The *User's Reference Guide* (page I-9 for TI-99/4A, page 15 for TI-99/4) tells how to connect the cassette cable and how to save and load data when you're using a module. The *Guide* also tells how to save and load a program you have written (pages II-40-42 for TI-99/4A and pages 68-70 for TI-99/4). Some other hints for using the cassette recorder are:

- Turn the tone control to the highest setting.
- Start with the volume about midrange.
- Type in OLD CS1 and follow the instructions printed on the screen.
- If you get the message "NO DATA FOUND," increase the volume.
- If you get the message "ERROR IN DATA," decrease the volume.

With some of the TI-99/4A consoles, a fraction of a change in volume can determine your success in reading a program. On a couple of consoles, I alternated between the two error messages at a volume setting near 2 or 3, then turned the volume up to about 8 or 9, and the program loaded with no problems.

The smallest plug of the cassette cable goes into the remote jack of the cassette recorder, so the computer can turn the recorder on and off automatically. If the recorder does not turn on and off properly, simply remove the remote plug from the jack.

You can operate the cassette recorder manually to save and load programs. For programs using the cassette recorder for data entry, you will need the remote capability. An adapter is available for the remote switch.

Two cassettes are used in some programs where you need to read and write data, such as updating files.

Speech. The TI Speech Synthesizer is a small box that attaches to the side of the computer and lets the computer speak to you. You will need a command module with built-in speech to hear the computer speak.

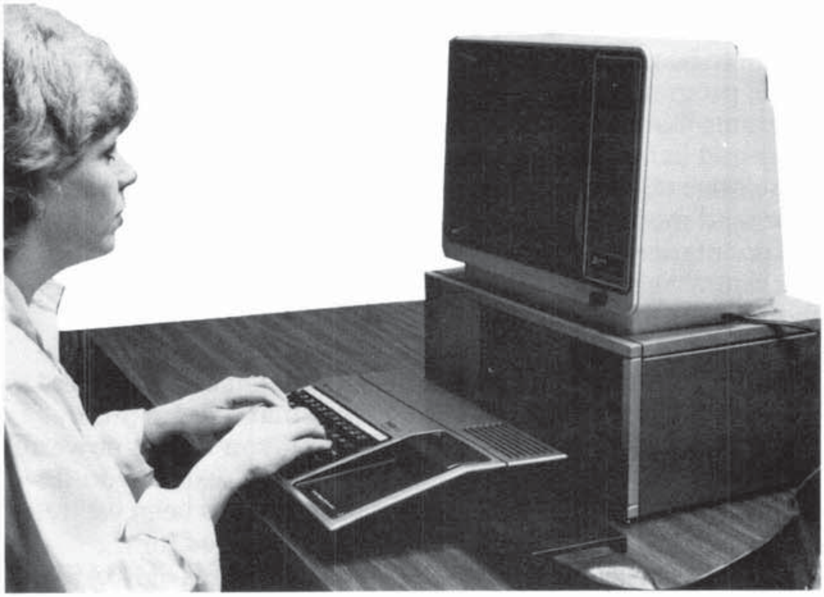
To program your own speech or to use any cassette or disk programs that use speech, you will need a module. *Speech Editor* and *Extended BASIC* are two modules that have speech capabilities with a given list of words. The *Terminal Emulator II* command module allows unlimited speech, and comes with documentation that gives you ideas and suggestions about programming speech. The easiest way is to spell something phonetically for the computer to pronounce.

Extended BASIC. TI Extended BASIC (XBASIC) is a programming language contained on a module. It comes with a programming reference card and a thick manual. No other peripherals are required to use XBASIC; if you want a powerful programming language, this may be the first "extra" you'll want to buy for your computer. If a program has been written in XBASIC, the XBASIC module must be inserted for the program to run properly. Some of the advantages of XBASIC are multi-statement lines, complex IF-THEN-ELSE logic, subroutine and MERGE capabilities, program security (save protection), excellent formatting, and moving sprites for graphics. If you like to write action games, Extended BASIC with the sprites is essential.

Hardware. There are two main ways to add peripherals to your TI computer. The old method has each peripheral in a separate box that connects to the side of the computer or the previous peripheral. The RS-232 Interface, 32K Memory Expansion, and Disk Controller look like identical boxes. The disk drives are hooked by a cable to the disk controller or another disk drive.

The new method is to add a Peripheral Expansion Box. With it, each peripheral is a "card" that is placed in the expansion box. The expansion box is attached to the computer (or speech synthesizer) by a thick cable, and it has its own power supply, so there aren't as many power cords dangling around as in the old system. The RS-232 Interface, 32K Memory Expansion, Disk Controller, and Disk Drive are "cards" that plug into the expansion box. Other cards are planned.

RS-232 Interface. The RS-232 Interface was my first add-on, because I wanted a printer, and the Interface allows the computer to "talk" to the printer. The RS-232 Interface has two



TI-99/4A with Peripheral Expansion Box

ports so that you may be connected to a printer and a modem at the same time. The instruction book that comes with the RS-232 tells you how to operate the computer under different conditions.

Printer. You may use a number of different brands of printer with your TI-99/4A. To connect your printer, you'll need a cable to go from the RS-232 Interface to the printer. The cable should be sold where you buy the printer.

Modem. Modems allow you to use phone lines to send information from one computer to another. There are several kinds of modems and acoustic couplers that will give you access to large computer networks, data bases, or other services. You will need the RS-232 Interface and either the *Terminal Emulator I* or *Terminal Emulator II* command module.

Disk controller and disk drives. You can save and retrieve data or programs with a diskette much more quickly than with a cassette system. To connect a disk drive, you also need a TI Disk Controller. One disk controller can handle up to three disk drives. Many business applications require two disk drives. The TI-99/4A presently uses single-sided 5¼-inch soft-sectored diskettes.

Memory expansion. The TI Memory Expansion gives your computer 32K RAM. However, that memory won't do you much good unless you use a module that will access it. You cannot use it with console BASIC alone. TI Extended BASIC does not *require* the memory expansion, but it can access it. Other programming languages may require the memory expansion.

Monitor. Although the TI-99/4A may be connected to your regular television set, a color monitor gives a clearer picture.

Logo. TI Logo is a fascinating programming language designed especially to teach computer literacy to young children. TI Logo is contained in a command module, and the 32K memory expansion is required. *Logo I* can print using the TI thermal printer only. *Logo II* has RS-232 capability so you can print listings on a regular printer, and it also has music. There are several manuals and books available to help teachers implement Logo in their classrooms.

Editor/Assembler. This language requires the memory expansion, disk controller, and one disk drive. It allows you to program in the machine language of the computer's TI 9900 microprocessor.

UCSD PASCAL. This language requires the memory expansion, P-code peripheral card, disk controller, and at least one disk drive (preferably two).

Chapter 2

Getting Started

Getting Started

The best way to learn to use your computer is by using it. Most parts of this book will be more understandable if you are actually sitting at your computer, typing in the sample programs and RUNning them as you go along.

Whenever you start writing a new program, it's good to keep in mind that there are certain things that must be done *before* you can do other things. That's what this chapter is about. Along the way, I'll briefly introduce each command and concept of TI BASIC as we come to it.

What You See

The TI-99/4A keyboard is much like a typewriter keyboard. The letters are all in the same positions, and so are most of the symbols. If you aren't already a touch typist, you will gradually learn to type as you program.

When you turn on your computer, you get the *title screen*. To do your own programming, press any key; then press 1. TI BASIC is now ready, waiting for you to begin programming.

The little black square that is blinking or flashing is called a cursor. Whatever you type in at the keyboard will appear right where the cursor is; the cursor will then move one space to the right (or to the next line), waiting for you to type something else.

How to Make Things Happen

The ENTER key is probably the most important key on the keyboard. You can find it easily — it is the key with a yellow dot on the front. Simply typing commands won't make the computer do anything except put the letters and numbers you typed on the screen. Things only start happening when you press ENTER.

As soon as you press ENTER, TI BASIC tries to follow your instructions. If your instructions begin with a line number, TI BASIC stores that line as part of your program, to be carried out later, in its proper order. If there is no line number, TI BASIC will try to carry out your instructions at once.

Some BASIC Commands

A *command* is a word that tells the computer to do something. The command must be typed correctly; if a word is spelled wrong, TI BASIC won't understand the command. The TI is also very particular about spaces, and when a command requires numbers, those numbers have to be within certain ranges.

For example, type CALL CLEAR. Then press ENTER. The screen clears immediately. This is the command we use whenever we want to erase the whole screen — I often use it near the beginning of a program to get rid of words or pictures left over from the last program I ran.

Line Numbers

A program consists of a series of commands for the computer to perform. The commands are numbered with line numbers so the computer will perform them in a certain order. No matter what order you type your lines in, when you enter RUN the computer will start with the lowest line number and carry out the commands on each line in numerical order, unless one of the commands directs it to do otherwise. The main thing a programmer does is arrange commands in a certain order to get the computer to do something.

If you would like the computer to number your lines for you as you are typing a program, type NUM. Then press ENTER. The number 100 will appear on the screen. After you type in the statement for line 100 and press ENTER, the number 110 automatically appears.

You do not have to start with line 100. For example, if you want to start at 20000, type NUM 20000.

The computer automatically increments the line numbers by ten unless you specify another number. If you prefer to start at line 50 and number by fives, enter the command NUM 50,5. The first number is always the beginning line number; then, after a comma, the second number is the increment between line numbers. If you don't specify your own choices when you enter NUM, by default TI BASIC will start at line 100 and number by tens.

Why should you skip line numbers when you are programming? If you don't, you may find that you need to insert a few instructions between lines 7 and 8; you would have to renumber your program to do it. By leaving nine unused numbers between every two lines, you have plenty of room to insert lines later.

The PRINT Command

One of the most used commands in computer language is PRINT. You may print messages by typing PRINT, followed by the message *in quotation marks.* To type the quote marks on the TI-99/4A, you will need to simultaneously press the function key (FCTN, the key with the gray dot) and the letter P. In fact, all the symbols on the fronts of the keys are obtained by pressing FCTN and the appropriate key.

You can put more than one message in the same PRINT command. Just put each message in its own set of quotation marks, and separate the messages with one or more print separators — either a colon, a comma, or a semicolon. TI BASIC interprets the print separators as instructions. A colon tells the computer to go to the next line; several colons in a row make the computer skip several lines. Semicolons tell the computer to join two messages together, with no space between them at all on the screen. Commas tell the computer to tabulate before printing the next message. Remember, though, that print separators must be *outside* the messages. If they occur inside the quotation marks, the computer will assume they are part of the message and simply print a colon, comma, or semicolon on the screen.

```
100 REM PRINT
110 CALL CLEAR
120 PRINT "HI"
130 PRINT "HELLO THERE"
140 PRINT "HERE ARE EXAMPLES"
150 PRINT :::"HERE ARE THREE COLONS"
160 PRINT :"GEORGE ";"SUSAN"
170 PRINT :"DOUG","SHEILA"
180 PRINT :"ROGER":"SHERYL"
190 END
```

Did you remember to type in the space after "GEORGE" in line 160? It makes quite a difference in what you see on the screen.

REM and END

You probably noticed that in the sample program you just ran, there are two new commands: REM and END.

REM means "remark." Anything that comes after that word is ignored when the computer runs the program. Why include REM statements if they have nothing to do with the program? They're really a guide for you — or for any other programmer who looks at your program and tries to figure out what you're doing. In a simple program like the one we just wrote, it's easy to see what's going on. But when a program has a few dozen lines and the variables start coming thick and fast, a REM statement here or there can help you keep track of what those lines of code are doing. REM statements use up memory, however, so if your program starts getting too large for your computer, you can always delete a few REMs to make space.

The command END stops the computer and tells the computer that is the end of the program. I like to put END as the last statement of my programs so that you will know you have all of the lines when you are typing in these programs. Actually, you may leave off the last line, and the computer will end by itself. A similar command is STOP, which also stops the computer as if at the end of a program. I usually use STOP when I want the computer to stop within a program (such as between subroutines or different sections of a program), and END as the last line of the program.

Skipping Around with GOTO

Remember that the computer executes a program line by line, taking the lines in numerical order. One way that you can change that order, though, is with a GOTO statement. GOTO is always followed by a line number: GOTO 150. If you GOTO the first line in the program, you start the program over again. You can create loops by telling the computer to GOTO an earlier line. You can even make the computer stand completely still by telling it to GOTO the very line it is already on:

```
320 GOTO 320
```

The only way to stop the program then is to press CLEAR. You can also GOTO a later line, skipping as many program lines as you like along the way.

Type in the following program. It is very inefficient, but it illustrates how you can GOTO all over the place. Press CLEAR to stop the program.

```
100 REM GOTO
110 CALL CLEAR
120 GOTO 150
130 PRINT "SECOND"
140 GOTO 170
150 PRINT "FIRST"
160 GOTO 130
170 PRINT "THIRD"
180 GOTO 180
190 END
```

The CALL SCREEN Command

A command I often use at the beginning of a program or at the beginning of a section of a program is CALL SCREEN. The 16 colors available on the TI are numbered, and the CALL SCREEN statement allows you to specify what screen color you want. For example,

```
100 CALL SCREEN(14)
110 GOTO 110
```

RUN this program and you will see a magenta screen.

Housekeeping Commands

Some TI BASIC commands almost never appear in programs, but programmers use them often while they are *creating* programs. NEW is like a broom. When you enter NEW, it sweeps away every bit of the program that is currently in memory. You'll use it to make sure you aren't getting old program lines mixed in with the new ones. But be careful — NEW sweeps *clean*. If you want to keep whatever you've been working on before, make sure to save it on cassette or disk *before* you enter NEW.

RUN is the command that tells the TI-99/4A to start at the lowest-numbered line of the program currently in memory and begin executing the commands it finds there. Any time during your programming you can enter RUN and see how your program is working so far.

LIST is the command that lets you look at the program lines that are currently in memory. If you simply enter LIST, the TI-99/4A will display the entire program, from the first line to the last. If the program is short, it will all fit on the screen. But if the program is long, only the last few lines will stay on the screen for you to examine.

One solution is to watch carefully as your program scrolls up the screen. When the lines you want to examine are on the screen, quickly press CLEAR. This will stop the scrolling and let you look at whatever was on the screen at the moment you pressed CLEAR.

A better solution is to LIST only a portion of the program. If you wanted to look only at line 320, you would enter LIST 320. If you want to look at a range of lines, then enter LIST plus the beginning and ending line numbers, with a hyphen (minus sign) in between:

Command	Lists:
LIST	Whole program
LIST 200-300	Lines 200 through 300
LIST -150	All lines from the beginning up to and including line 150
LIST 300-	All lines from 300 to the end

If you ask for a range of lines that doesn't exist, there's no harm done — the computer just doesn't LIST anything.

Editing

Not so long ago, to try a program out you had to punch computer cards and then submit the deck to a computer center. Hours or days later you could pick up your results. Of course, sometimes there were typing (or syntax) errors or logic errors which would need correcting. The job would be resubmitted, and another day would go by before you could see the results.

Now, with home computers and terminals, the whole process of programming, correcting, and getting results is much, much faster. Within seconds you may change a number in a statement and see the results.

The TI has very easy-to-use editing capabilities built in. Either before or after you have pressed ENTER, you may correct typing errors on any line in the program. On some commands, if you have typed the statement incorrectly, and then pressed ENTER, the computer will immediately remind you that something is wrong (for example, if you spell CALL with only one L).

For simplicity, I will describe editing using the TI-99/4A. If you have the TI-99/4, be sure you have the programming overlay. You will use the SHIFT key and the appropriate key marked on the overlay.

The Editing Function Keys

Most of the editing on the TI-99/4A can be done by pressing the function key (FCTN, the key with the gray dot) with another key. You should have a narrow strip overlay that fits above the number keys. The bottom line of the overlay has a gray dot at the right. If you push the key with the gray dot (FCTN) plus the number key, the computer will do the corresponding command. For example, FCTN 4 is CLEAR. To stop a program at any time, you may press FCTN 4, CLEAR. The function keys used in editing are the arrow keys and numbers 1, 2, 3, and 4. The other numbers are used with some of the modules.

Now take a look at the arrow keys (found on letter keys E, S, D, and X). These are the same arrow keys you use to move in games; you also use them *with the FCTN key* to edit. If you want to back up as you are typing in a command, just press FCTN and the left arrow key. Type over whatever it is you want to fix; then press FCTN and the right arrow key to get back where you were. The right arrow and left arrow keys will repeat if you hold them down longer than a second.

Let's try some examples. Type in the following example program exactly as shown, *including* errors.

```
100 CALL CLEAAR
110 CALL SCREEN(14)
120 PRINT "HI"
130 GOTO 130
140 END
```

Correcting Errors

Of course you noticed that in line 100, the word *CLEAAR* is misspelled. One way to correct the error is to type line 100 over again, but you can save yourself some retyping by using the editing keys. To edit line 100, type 100; then press FCTN and the ↓ key. (You could also have typed in EDIT 100 and then pressed ENTER, but the first method is quicker.)

You'll notice that line 100 appears at the bottom of the screen with the cursor on the first character of the line. Now press FCTN and the right arrow key until you are directly over one of the extra *A's* in CLEAAR. Now press FCTN and 1 (for DELete). The word should now appear as CLEAR.

Watch out, because DELete is also a repeating key. If you hold it down too long, you'll lose more letters than you want.

When you press ENTER, the new line as corrected will replace the old line.

Revising a Program

Let's assume you don't like my magenta screen. First we need to find out which line we need to change. LIST your program by typing LIST, then pressing ENTER. The line you need to change is 110, so type 110 and then press FCTN ↓.

Use the right arrow, FCTN→, to move the cursor to the 4 in 14. Type 6 and then press ENTER. This time your screen color is 16. RUN the program again.

Suppose you don't like that color either. Press FCTN 4 (CLEAR), then type 110 and press FCTN ↓. Say you want color 6. Use FCTN→ to get to the 1. Press FCTN 1 for DELete: Then press ENTER.

Did the editing work? Enter LIST 110 to see. Line 110 should say CALL SCREEN(6).

You have probably noticed that you do not have to be at the end of the line to press ENTER. No matter where you are on the line when you press ENTER, the entire line will be stored in the program.

Now RUN the program again. This time let's edit line 120. Type 120 and press FCTN ↓. Use FCTN→ to get to the I in "HI". Stop right on top of the I and type ELLO" to replace "HI" with "HELLO." ENTER and RUN to see the change.

Inserting Characters

Let's try another function key. INSert is used to add characters to a line without having to type the whole line over. Type 120 and press FCTN ↓ to bring line 120 into editing mode. Press FCTN→ until the cursor is directly over the H.

Press FCTN 2 for INSert. Then type JIM, and notice how the rest of the line moves over. (Remember to type a space after the comma so the phrase will look right.)

When you are through inserting characters, press FCTN → and go to the second quote mark (after the O in HELLO). Now insert an exclamation mark. Be sure you use the SHIFT key and not the FCTN key when you press 1. After your line looks right, press ENTER.

Changing Your Mind

If you are editing a line and decide you don't want to change it after all, press FCTN 4 for CLEAR and the line will stay as it was before you began editing it.

When you are typing in a program, FCTN 4 will get you off a line, and the computer will ignore that line. If it is a new line, it will not be entered as part of the program.

FCTN 3 for ERASE will erase the line you are typing. You may wish to pause here a few minutes and experiment with ERASE and CLEAR to see the difference.

To delete or get rid of a whole line, type the line number only; then press ENTER. The left arrow, right arrow, and DELete keys have the automatic repeat feature; just hold the key down for longer than one second and it will start repeating.

The INSert key needs to be pressed just once, and characters will keep being inserted as you type until you press ENTER, DELete, or one of the arrow keys.

Up and Down

When you are editing more than one line, the up arrow and down arrow keys will come in handy. Let's assume you have the following lines in your program:

```
200 CALL HCHAR(3,5,42)
210 CALL HCHAR(3,8,42)
220 CALL HCHAR(3,20,33)
```

You RUN your program and discover the graphics needs to be a line lower — the row value needs to be changed from 3 to 4 in all three lines.

Type 200 and press FCTN ↓ to begin editing line 200. Use the right arrow to go over and change the 3 to a 4.

Now, however, instead of pressing the ENTER key, press

FCTN ↓ . The very next line, line 210 in this case, will appear for editing; line 200 has also been entered. Likewise, the up arrow will give you the line just before the one on which you were working. You can scroll up and down from line to line through your whole program using the up and down arrows.

If you have pressed either the up arrow or the down arrow and find yourself on a line that does not need editing, you may press CLEAR to get out of the editing mode.

Renumbering

RES is a command that stands for *resequence*. If you have been programming, adding lines here and there, your program can get quite crowded and confusing. If you were to renumber all the lines yourself, it could take a long time, especially because you would have to find and change every *reference* to a line — every GOTO or GOSUB or THEN command that sends the computer to the line whose number you have changed.

The TI-99/4A makes it easy. Just type RES and press ENTER. As soon as the cursor reappears, your program has been resequenced, or renumbered, including all line numbers referenced in other lines. Try this sample:

```
100 REM RES SAMPLE
110 CALL CLEAR
120 GOTO 300
150 CALL SOUND(150,440,2)
200 CALL HCHAR(INT(RND*24+1),INT(RND*32+1)
    ,42,5)
210 RETURN
300 GOSUB 150
310 GOSUB 200
320 GOSUB 150
330 GOTO 300
500 END
```

First LIST the program and notice the line numbers. Now type RES and press ENTER. LIST the program again. The lines are resequenced, starting with 100 and incrementing by 10. Notice that the numbers after GOSUB and GOTO have been changed.

As with the NUM command, you may specify the starting line number and the increment. The first number after RES is the starting line number; the second is the increment.

Try RES 10. Then LIST to see the line numbers.

Try RES ,5. Then LIST.

Try RES 1,1. Then experiment with your own numbers.

Organizing the Program

Quite often when I am writing a program, I start off with the command NEW, then NUM, and then enter the preliminary statements of the program. If the program is gong to have several sections, I start the first section with line 1000; I use the command NUM 1000 to start typing in lines. Then I start the second section with line 2000, and so on. After the program is finished, I can RES so the line numbers are arranged starting with 100 and incrementing by 10.

You can also use the RES command to help you add lines. Suppose the lines are numbered in increments of 10, but you discover you need to add 15 lines between two statements. RES ,50 will spread the line numbers apart so you'll have plenty of in-between numbers to use.

Another reason I RES when I'm finished with a program is so others who look at my program can't tell where I planned poorly and had to add lines.

Initializing Variables

All that your computer understands is numbers. Even the letters that make up commands are just numbers to the computer. Fortunately, TI BASIC takes care of letting the computer know whether to treat any particular number as a command, a character, or a number. All you have to worry about are a few rules for entering your commands, characters, and numbers. I've already gone over some of the commands. Now it's time to start giving the computer some numbers and getting back results.

Numeric Operations

Arithmetic is simple. Just enter a statement like PRINT 456+5997 and your TI-99/4A will give you a quick answer. The four simple arithmetic symbols are + (add), - (subtract), * (multiply), and / (divide). You can make your problems as complex as you like: PRINT (4*(99/3))-(1111+(88/4)).

There are three ways that numbers can get into your program. The first way is the one I just used, putting the numbers directly in a program statement: PRINT 88/(14*2-6). Another way is to have a list of data in a DATA statement, and have the program read the DATA — I'll explain more about that later. A third way is to use INPUT statements, and have the computer user enter the numbers while the program is running.

Storing Numbers as Variables

But the real power of the computer is that you don't have to put the numbers directly into your programs. Instead, you can use variables. Variables are like a very long row of cupboards. When you start your program, the cupboards have no labels. So you label the first cupboard *A*, and then store a number inside it. From now on, whenever your program uses the variable *A*, TI BASIC goes to the memory location named *A* and brings back whatever number is stored there.

You can name quite a few variables in each program, and you can change the value of that variable (the number stored in that cupboard) as often as you like. When your program uses a variable name, TI BASIC will use the most recent value you assigned to it.

Types of Variables

There are two types of variables, *string* and *numeric*. The difference is that string variables are given names that end with a dollar sign, like A$, NAME$, or R55$, and whatever value is assigned to a string variable is treated as characters rather than numbers. String operations are performed on string variables; numeric operations are performed on numeric variables.

Naming Variables

There are a few simple rules in naming variables. First, variable names have to start with a letter. Second, variable names can only consist of letters and numbers. A1, B53RN, NUMBEROFPEOPLE, and WATERGATE55 are all legitimate variable names. A@, 15B, WHAT'S THIS, and #OFDOCTORS are not legal variable names.

Third, no two variables can have the same name — if two variables have the same name, the computer assumes they are the same variable. But it doesn't take much to make the names different. If even one character in the variable names is different, your TI-99/4A will always be able to tell which is which.

Fourth, your variable name can't begin with a complete command. LIST14 is not a legal variable name.

Your variable names don't have to mean anything, but it's often a good idea to use words that have some meaning. In a long program, you'll sometimes have dozens of variables at once, and it's a lot easier to keep track of what each one means when, instead of naming them A, B, C, D, and so on, you have named them SCORE1, SCORE2, TIME, SPEED, and TVCOL.

Assigning Values to Variables

In many BASIC languages, when you want to use regular statements of your program to give values to variable names, you have to use the LET statement: LET A=7. However, in TI BASIC, the word LET may be omitted, so all you need is the variable name and the value: A=7. This tells the computer to use 7 whenever your program calls for the variable A.

If you use a variable in a calculation without previously assigning a value to it, the computer automatically assumes a value of zero.

Here is a sample program.

```
100 A=7
110 B=10
120 PRINT A*B
130 END
```

If you RUN this program, the result printed would be 70.

Now suppose you had entered *this* program:

```
100 A=7
110 PRINT A*B
120 END
```

This time the value of B was never assigned. The computer assumes that B equals 0, and the result printed is zero.

The TI-99/4A sets all variables to zero each time a program is run. If you want your variables to start at zero, you won't need to define them. However, any time your program has an option to do a process more than once, you will need to make sure your variables have the right value each time the process begins again. Many of the variables will need to be re-initialized, while others will not.

Where Should Variables Be Initialized?

For example, let's say you are designing a game in which two players take turns moving a figure around a screen for a limited amount of time. At the end of the game, the players have the option of starting over.

While I won't try to show you the whole program, I can show you the basic design of it. There are three principal loops. The inner loop is the player movement loop. This one repeats hundreds of times in a game, each time the player moves his on-screen figure. The program keeps track of where the on-screen figure is with the variables *X* and *Y*. (In this book, *X* represents the *row numbers,* or vertical placement of the character; *Y* is the *column number,* or horizontal position.)

The next loop is the player-turn loop. When Player One's time is up, it is Player Two's turn; when his or her time is up, it is time for Player One again. The program keeps track of the amount of time that has passed with the variable TIME.

The outer loop is the play-again loop. When a game is over, the players are asked if they want to play again. If their answer is yes, then the whole game starts over again from the beginning, and their scores are set at zero. The program keeps track of their scores with the variables SCORE1 and SCORE2.

Figure 2-1 is a diagram of the program, including five LET statements. If the variable assignment statement is inside a loop, then each time the loop repeats that variable is initialized — set back to its original value. Otherwise, the variable keeps whatever value it received during the last time through the loop.

Notice where the values of the variables are initialized. X and Y are set to their starting position only at the beginning of a turn, *outside* the movement loop. If those values were initialized inside the movement loop, then every time the figure moved, it would immediately come right back to the starting place. TIME is also initialized at the beginning of each turn. But SCORE1 and SCORE2 are only initialized at the start

Figure 2-1. Planning Program Loops

(Initialize the program — define characters, draw the screen.)

play-again loop

(Beginning of play. The play-again loop starts here.)

```
200 SCORE1=0
210 SCORE2=0
```

player-turn loop

(Beginning of the player's turn. The player-turn loop starts here.)

```
300 TIME=0
310 X=16
320 Y=12
```

movement loop

(Check to see if the player wishes to move the figure. The player movement loop starts here.)

(Move the figure by changing the values of X and Y. Check to see if the time is up. If no, then return to the beginning of the player movement loop. If yes, then return to the beginning of the player-turn loop.)

(End of play.)

```
1000 PRINT "PLAY AGAIN? (Y/N)"
```

(If yes, return to the beginning of the play-again loop.)

```
1200 END
```

of each new game — otherwise, every time a player's turn ended, the scores would be reset to zero.

Variables in FOR-NEXT Loops

Another way variables are given values is in FOR-NEXT loops. (FOR-NEXT loops are described in more detail in Chapter 4.) Briefly, a variable is used as a counter in the loop, and each time through the loop, the counter increases or decreases by a set amount. Try this program:

```
100 FOR C=5 TO 10
110 PRINT C
120 NEXT C
130 END
```

The first time through the loop, C has the initial value of 5. When the computer executes line 120, it increments (adds 1 to) C and then checks to see if it is less than or equal to 10. If C is less than or equal to 10, the computer goes back to line 100 and begins the loop again. After the loop is finished, C has the value of 11 because it was incremented one last time after it reached the target value of 10.

This can be very powerful in programming. Add these lines to the program you just ran:

```
112 D=D+(23*C)
114 PRINT D,
116 PRINT 400-(10*C)
```

Now RUN the program. By performing various operations using the counter variable, you can produce many different effects in the same loop.

DATA and READ

Directly defining variables in statements or using FOR-NEXT loops is the easiest way to keep track of the value of a variable. That way, however, each variable needs a separate statement every time its value changes. If you need to save memory or prefer to use fewer statements, then the DATA method may be used.

A DATA statement consists of the command DATA followed by as many items of data as will fit on a line. DATA 9 is a complete, valid statement. So is DATA 9,50,DONNA SMITH,3,-580,0,0,28,1899,10,FRANK WADE. There can be as many DATA statements as you like in a program, and you can put them anywhere in the program you want. The computer doesn't do anything with them until it comes to a READ statement.

The READ statement is always followed by as least one variable name. READ A is valid. So is READ A,B,C$,D,E,F,G,H,I,J,K$. (Notice that when the DATA is a string, it must be READ into a string variable.) When the computer encounters the first READ statement in a program, it starts at the beginning of the program and looks for the first DATA statement. The first variable after the READ statement is assigned the value of the first item of data after the DATA statement. The second variable after READ is given the value of the second item of data, and so on. (READ and DATA are described in more detail in Chapter 6.)

Following this paragraph are two programs assigning the same values to the same variables. The one on the left assigns the values in separate statements. The one on the right uses READ and DATA. The DATA statement happens to come after the READ statement. It could have come before; it makes no difference.

```
100 A=7
110 B=10
120 C=6
130 D=2
140 E=5
150 PRINT A*B+C/D+E
```

```
100 READ A,B,C,D,E
110 PRINT A*B+C/D+E
120 DATA 7,10,6,2,5
```

User INPUT

One more way to get information into the computer is to let the user of the program INPUT the data. Here is an example:

```
100 CALL CLEAR
110 INPUT "ENTER A NUMBER ":A
120 PRINT "YOUR NUMBER SQUARED IS ";A*A
130 END
```

In line 110, the computer waits for the user to type something in and then press the ENTER key. The variable A is assigned the value of whatever number the user enters.

Line 110 shows only one way of using the INPUT statement. Another way is to use a PRINT statement and an INPUT statement:

```
110 PRINT "ENTER A NUMBER"
115 INPUT A
```

If your message to the user, or *prompt*, is part of the INPUT statement, then it must come immediately after the command INPUT. After the message, type a colon and the variable name.

User Errors

What happens if the user doesn't follow your instructions, and enters a letter or symbol, or a number too large for your program to handle? If the user INPUTs a letter or symbol, the computer gives him or her a chance to try again. If the user INPUTs a number too large to handle, the program *crashes* — it stops abruptly and won't go on. So it's a good idea to test the value entered and make sure it is within reasonable limits for your particular program *before* you actually try to do anything with the INPUT variable. For example:

```
100 CALL CLEAR
110 PRINT "ENTER A NUMBER"::"FROM 1 TO 100
    ."::
120 INPUT N
130 IF N>=1 THEN 160
140 PRINT :"SORRY, TRY AGAIN.":::
150 GOTO 110
160 IF N<=100 THEN 190
170 PRINT :"SORRY, MUST BE LESS THAN 100"
180 GOTO 120
190 PRINT ::"YOUR NUMBER SQUARED IS";N*N
200 END
```

The symbol < means "less than." The symbol > means "greater than." Combined with the equal sign (=), these symbols mean "less than or equal to" or "greater than or equal to."

Testing with the IF-THEN Command

The IF-THEN command is very powerful. It tests to see if a particular condition is true or not. If it is false, then the computer goes on to the next line of the program. But if the condition is true the computer goes somewhere else.

In line 130, the command IF was followed by the test condition. Is it true that the value of variable N is greater than, or equal to, 1? If not, the computer will go on to line 140 — the user entered a number that the program cannot use. If N *is* greater than or equal to 1, however, the program branches to the line number that follows the command THEN. If the statement following IF is true, the program will always go to the line number following THEN.

CALL KEY for User-proof Input

One way to avoid user errors is to give the user fewer choices. If you ask a yes or no question and use the INPUT method of getting the user's answer, what is to stop the user from typing YEP or AFFIRMATIVE or OF COURSE NOT? A better way is to give him only two choices, each consisting of only one letter. Then use the CALL KEY statement to find out what letter the user chose.

The CALL KEY statement is always followed by a zero and two variables, all in parentheses: CALL KEY(0,K,S). All that concerns us now is the first variable — in this case, K. After the CALL KEY statement is executed, the value of that variable will be the numeric code for the last key that the user pressed.

```
800 PRINT "PRESS Y FOR YES, N FOR NO"
810 CALL KEY(0,K,S)
820 IF K=89 THEN 200          (Y WAS PRESSED)
830 IF K<>78 THEN 810         (ANY KEY OTHER
                               THAN N)
840     (PROGRAM CONTINUES FOR N)
```

If the user presses Y, the program branches to line 200 for the YES procedure. If the user presses N, the program continues to line 840 for the NO option. Any other key pressed sends the computer back to line 810, to ask for another key. The program will not continue until the user presses either Y or N.

Menus

The CALL KEY method isn't limited to "yes/no" situations. You may have a menu of options on the screen. Label each option with a letter or number; then use a CALL KEY statement to read which key was pressed and branch appropriately. If the user presses a key that is not one of the choices, you can send the computer back to ask for a new key.

In the following example, the user must press 1, 2, 3, or 4. If any other key is pressed, the value of K is less than the ASCII code 49 (the code for the character "1") or greater than 52 (the code for the character "4"), and the computer will return to the CALL KEY statement in line 120.

```
100 CALL CLEAR
110 PRINT "CHOOSE 1, 2, 3, OR 4"
120 CALL KEY(0,K,S)
130 IF (K<49)+(K>52) THEN 120
140 ON K-48 GOTO 1000,2000,3000,4000
1000 PRINT "1"
1010 GOTO 120
2000 PRINT "2"
2010 GOTO 120
3000 PRINT "3"
3010 GOTO 120
4000 PRINT "4"
4010 GOTO 120
5000 END
```

The ON Command

In line 140, we used the ON command. ON is similar to IF-THEN, because it can cause the program to branch to another line. This time, though, instead of having only one possible branch, there can be many.

Instead of testing the statement that follows ON to see if it is true or false, TI BASIC finds out the *numerical value* of the expression. In line 140, the expression after ON was K-48. The variable K held the value of the ASCII code for the key the user pressed; because of the test in line 130, you know that K is a number from 49 to 52. Now we subtract 48, so that the value of the expression is a number from 1 to 4.

Now comes the multiple branching. If the value of the expression is 1, the program will branch to the first line number following the GOTO command. If the value is 2, the program branches to the second line number, and so on. If our menu had 10 options, we could have specified ten line numbers after the GOTO statement.

However, you must be careful when using ON statements to make sure that the expression tested by ON has a value no less than one and no greater than the number of line numbers specified after the GOTO command. The program will crash if

ON finds a value for which there is no corresponding line number after GOTO.

Initializing Arrays with DIM

Earlier in this chapter we noted that variables all need to have different names so the computer can tell them apart. There is an exception. A variable *array* is a group of variables with the same name. However, the computer can tell them apart because the variable name is followed immediately by a number in parentheses, called a *subscript*. Arrays are discussed in more detail in Chapter 6; what matters now is how you initialize a variable array.

Since each variable in an array takes up space in memory (whether you are using it or not), it is important to make sure that each variable array has as many individual subscripted variables as you need — and no more. The DIM ("dimension") statement does this. If you use a subscripted variable, like A(7), *before* DIMensioning the array, TI BASIC will automatically DIMension that variable array as if you had entered the statement DIM A(10). That will allow you to use subscripted variables from A(0) to A(10). If you tried to use a variable like A(11), however, you would crash your program.

So it is usually a good practice to use a DIM statement early in the program for all the variable arrays your program will use. The DIM statement *should* come right at the beginning of the program, so that in editing you don't accidentally add a line that uses a subscripted variable before the DIM that creates it.

A DIM statement can create several arrays, separated by commas, and each array can provide for up to three subscripts, like this:

```
100 DIM A(7),B(2,30),C(5,2,7),D(3)
```

Subscripts may be zero, and the computer automatically reserves a spot for the variable with zero subscripts. The statement DIM A(7) actually creates *eight* subscripted variables, from A(0) to A(7). If you need to save memory and prefer to start the subscripts numbering with 1, use the following procedure:

```
100 OPTION BASE 1
110 DIM A(7),B(2,30)
```

Defining Functions

Another statement that needs to be near the beginning of the program is a DEF (''define function'') statement. It is used to DEFine a *function,* or series of commands, that will be used often in the program.

Here is a segment of a program that could be used to check homework. Assume the algebra teacher wanted you to evaluate $F(X) = X^3 + 2X^2 + X / 2$ for various values of X. The program is:

```
100 CALL CLEAR
110 DEF F(X)=X^3+2*X^2+X/2
120 INPUT "ENTER VALUE FOR X: ":Q
130 PRINT :"ANSWER =";F(Q):::
140 GOTO 120
150 END
```

The symbol ^ means ''to the power of''; X^2 means ''X to the power of 2,'' or ''X squared.''

RUN this program and INPUT values of 3, 7, 4, or any other numbers.

Function Variables

In this program the name of the function is F. In line 110, the X in parentheses immediately after F is a variable name used inside the function. That is, when the computer carries out this function, it will use whatever value it finds inside the parentheses after the function name as the value of the variable X. When the function was carried out in line 130, the value inside the parentheses was the value of the variable Q, which got its value from the INPUT statement in line 120.

Also, the variable X in this program is used only within the function. It has no value outside the function. You can even use X as a variable elsewhere in the program, and function F will have no effect on it. To test this, add the following lines to the program:

```
90 X=1000
135 PRINT X
```

Now RUN the program. You can see that no matter what value

X has inside the function, it has no effect on the rest of the program.

Functions don't have to have variables associated with them. They can also be used as often as you like within a program. Try this program:

```
100 CALL CLEAR
110 DEF R=INT(RND*10)+1
120 CALL HCHAR(R,R,47+R,R)
130 GOTO 120
140 END
```

R is first defined as a random number from 1 to 10. In the CALL HCHAR statement, a random number from 1 to 10 is placed a random number of times on the screen starting at a random row and random column.

Using Random Numbers

One other statement that needs to be used before a related statement is RANDOMIZE. Random numbers are used many times in computer applications. TI BASIC uses the RND function to specify a random number. Try this program:

```
100 CALL CLEAR
110 FOR I=1 TO 10
120 PRINT INT(100*RND)+1
130 NEXT I
140 END
```

This program will print ten random numbers from 1 to 100. RUN the program and note the results.

Now RUN the program again. And again. You'll notice that the same sequence of numbers is printed each time.

It could be very handy in debugging a program to know exactly what sequence of numbers will appear. However, in most situations you really want random numbers — different every time.

To get it truly random, use RANDOMIZE. Add this line to the program:

```
115 RANDOMIZE
```

Now try the program several times; the results will be different.

Sometimes it works to place one RANDOMIZE statement near the beginning of the program, but not always. It's probably best to use RANDOMIZE just before you use a statement involving the RND function.

Defining Graphics Characters

Chapter 5 of *Beginner's BASIC,* the manual that comes with the TI-99/4 or TI-99/4A, teaches you how to place graphics characters on the screen, how to define your own graphics characters, and how to set colors for your graphics. Let's look at some additional graphics concepts.

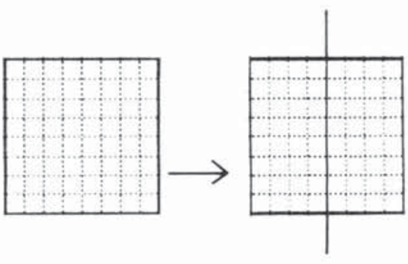
Defining graphics characters is part of getting started because usually you will want to define colors and characters before you place them on the screen. Chapter 3 of this book will give you more ideas about graphics and colors, along with some programs using graphics for you to try.

How the Screen Is Organized

The TI-99/4A divides the television into squares. These are arranged in 24 rows and 32 columns.

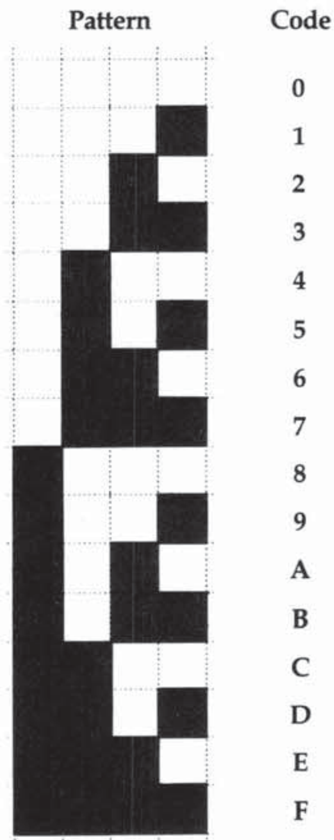
Each of those squares is further divided into 64 tiny dots arranged in eight rows and eight columns. Each dot in that 8 by 8 square can be turned on or off — colored in or not. The arrangement of colored and not-colored dots gives the shape of each character.

To define a character of your own, think of that square as if it were divided into half. Each half is four dots wide.



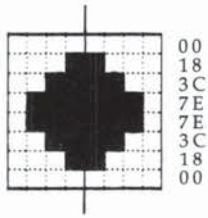
Figuring Out the Character Code

Each four-dot row can have any one of sixteen possible combinations of colored and not-colored dots. Each possible combination has a code which you can use to tell your computer what arrangement of dots you want in the character you're defining. The following chart shows each combination of dots and the corresponding codes.



To design your character, figure out which dots on your eight-by-eight grid should be colored or filled in. Then, using the code chart, figure out what the code for each four-dot segment is. Then arrange the code for all the segments in order, starting in the upper left-hand corner of the character and proceeding just as you do when you read a book — left to right, then down to the next line, then left to right again. There will be sixteen segments in each finished character code.

For a ball, the pattern might be



The finished code is "00183C7E7E3C1800."

The String Method

Once you get the code in proper order, there are several ways to tell the computer how you want to define the character. The easiest way is using one statement to define each character.

```
120 CALL CHAR(128,"FF818181818181FF")
130 CALL CHAR(129,"FFFFFFFFFFFFFFFF")
```

If your character definition ends with zeroes, you may omit them. The computer assumes that if you use fewer than sixteen codes to define a character, all the rest of the character will be blank. For the ball shown above, the definition could be CALL CHAR(130,"00183C7E7E3C18"), leaving off the two *final* zeroes, but not the two *beginning* zeroes.

The String-Variable Method

Another method of defining characters is to assign the code to a string variable first and then use the CALL CHAR statement:

```
120 A$="FFFFFFFFFFFFFFFF"
130 CALL CHAR(128,A$)
140 CALL CHAR(136,A$)
```

The DATA Method

One more method of defining characters is to use DATA statements. Here are two examples:

```
100 CALL CLEAR
110 FOR C=1 TO 10
120 READ C1,C$
130 CALL CHAR(C1,C$)
140 NEXT C
150 DATA 96,000000FFFF,97,2070D08809050602
    ,102,0808080F0F,104,080808FFFF
160 DATA 110,0070888F8F887,111,06137CFFFF7
    C1306,117,07182040438080 8
170 DATA 118,3018848232818101,120,FF,136,0
```

Notice that in line 120 the program reads the character number, then the code string, and assigns them to the variables C1 and C$. If you are defining a series of character numbers in order, use the counter variable to specify the character number, like this:

```
100 CALL CLEAR
110 FOR C=96 TO 105
120 READ C$
130 CALL CHAR(C,C$)
140 NEXT C
150 DATA FF,FFFF,FFFFFF,FFFFFFFF,FF8181818
    18181FF,FFFFFFFFFFFFFFFF
160 DATA 0808080F0F,080808FFFF,080808F8F8,
    FFE
```

A Character Definer Program

The following program allows you to design a graphics character without resorting to paper and pencil. You will see a large square which has been divided into sixty-four smaller squares, representing the eight-by-eight character grid. Use the arrow keys to move the cursor. Press F if you want the square filled in and the space bar if you don't want the square filled in. Press ENTER when you are finished with your square. The computer will calculate the pattern of on and off dots and will print the code values. Then an actual-size character will be placed on the screen so you can see what your character looks like. The definition is then repeated in string form so you may write it down and use it in your own program.

After the character is defined, you have the option of modifying it, defining a new character, or ending the program.

If you choose to modify, the character you just drew will reappear. You may alter any squares you wish.

If you choose the new-character option, a blank square appears.

How the Program Works

Naturally, some characters were defined in order to create the screen display in this program. Character 97, *a*, is re-defined as an open square, □, and Character 98, *b*, is defined as a filled square, ■ (lines 200-210). When the 8×8 grid is drawn on the screen, it is done by printing the string "aaaaaaaa" eight times (lines 420-440).

The character codes as they will appear on the screen are READ in as DATA (lines 120-170). The string array H$(0,1) through H$(15,1) holds the sixteen patterns of blank ("a") or filled-in ("b") squares. The string array H$(0,2) through H$(15,2) holds the corresponding code number or letter as you would use it in *your* programs later.

The flashing cursor is red so that you can tell where you are on the pattern you are designing (lines 180-190).

CALL GCHAR(X,Y,C) determines what character number C is at row X and column Y (line 480).

Lines	
120-170	READ in from DATA statements the pattern and corresponding hex code.
180-190	Define red cursor as Character 128.
200-210	Define "a" as a blank square and "b" as a filled square.
220-270	Clear the screen and print the instructions.
280	For the first run of the program and for Option 2, to design a new character, branch to line 420.
290-410	For the option to modify the previously designed character, evaluate the character definition code numbers one at a time and print the corresponding patterns on the 8×8 grid.
420-440	Print new 8×8 grid to begin character designing.
450-460	Assign starting values to the cursor position variables, X (horizontal) and Y (vertical).

470	Beep a tone to indicate user may move.
480	Determine the cursor character and put it at X and Y.
490-520	Blink the cursor over square while waiting for user to press a key.
530-750	If a key is pressed, branch appropriately. If an arrow key is pressed, move the cursor in the correct direction, making sure of the boundaries first. If the space bar is pressed (K=32), print a blank square. If ''F'' is pressed (K=70), print a filled square.
760	If ENTER was pressed (K=13), beep a short tone.
770-950	For eight rows, determine the character pattern of the first four squares and print the corresponding character code; then find the pattern of the second four squares and print the corresponding code. D$ collects the codes for the string definition.
960-970	Draw the actual size character, as defined by the user, at row 20, column 20.
980	Print the code string that defines the character.
990-1040	Print the options and branch appropriately.
1050-1100	Subroutine to compare user's pattern with preassigned patterns to determine corresponding hex code.
1110-1120	End.

Program 2-1. Defining Characters

```
110 REM DEFINING CHARACTERS
120 DIM H$(15,2)
130 FOR I=0 TO 15
140 READ H$(I,1),H$(I,2)
150 NEXT I
160 DATA aaaa,0,aaab,1,aaba,2,aabb,3,abaa,4
    ,abab,5,abba,6,abbb,7,baaa,8,baab,9
170 DATA baba,A,babb,B,bbaa,C,bbab,D,bbba,E
    ,bbbb,F
180 CALL COLOR(13,9,1)
190 CALL CHAR(128,"FFFFFFFFFFFFFFFF")
```

```
200 CALL CHAR(97,"FF818181818181FF")
210 CALL CHAR(98,"FFFFFFFFFFFFFFFF")
220 CALL CLEAR
230 PRINT "DEFINE A GRAPHICS CHARACTER"
240 PRINT :"PRESS F TO FILL THE SQUARE"
250 PRINT "PRESS SPACE TO CLEAR SQUARE"
260 PRINT "PRESS ARROW KEYS TO MOVE"
270 PRINT :"PRESS ENTER WHEN FINISHED": : :
280 IF (K=50)+(K=0)THEN 420
290 FOR I=1 TO 15 STEP 2
300 FOR L=0 TO 15
310 IF SEG$(D$,I,1)=H$(L,2)THEN 330
320 NEXT L
330 C$=H$(L,1)
340 PRINT "{3 SPACES}";C$;
350 FOR L=0 TO 15
360 IF SEG$(D$,I+1,1)=H$(L,2)THEN 380
370 NEXT L
380 C$=H$(L,1)
390 PRINT C$
400 NEXT I
410 GOTO 450
420 FOR I=1 TO 8
430 PRINT "{3 SPACES}aaaaaaaa"
440 NEXT I
450 X=16
460 Y=6
470 CALL SOUND(150,1397,2)
480 CALL GCHAR(X,Y,C)
490 CALL KEY(0,K,S)
500 CALL HCHAR(X,Y,128)
510 CALL HCHAR(X,Y,C)
520 IF S<0 THEN 490
530 IF K=13 THEN 760
540 IF K=70 THEN 740
550 IF K=32 THEN 720
560 IF K<>68 THEN 600
570 IF Y=13 THEN 470
580 Y=Y+1
590 GOTO 480
600 IF K<>88 THEN 640
```

```
610 IF X=23 THEN 470
620 X=X+1
630 GOTO 480
640 IF K<>83 THEN 680
650 IF Y=6 THEN 470
660 Y=Y-1
670 GOTO 480
680 IF K<>69 THEN 490
690 IF X=16 THEN 470
700 X=X-1
710 GOTO 480
720 CALL HCHAR(X,Y,97)
730 GOTO 470
740 CALL HCHAR(X,Y,98)
750 GOTO 470
760 CALL SOUND(150,440,2)
770 D$=""
780 FOR I=1 TO 8
790 C$=""
800 FOR J=6 TO 9
810 CALL GCHAR(I+15,J,C)
820 C$=C$&CHR$(C)
830 NEXT J
840 GOSUB 1050
850 CALL HCHAR(I+15,16,ASC(D1$))
860 D$=D$&D1$
870 C$=""
880 FOR J=10 TO 13
890 CALL GCHAR(I+15,J,C)
900 C$=C$&CHR$(C)
910 NEXT J
920 GOSUB 1050
930 CALL HCHAR(I+15,17,ASC(D1$))
940 D$=D$&D1$
950 NEXT I
960 CALL CHAR(136,D$)
970 CALL HCHAR(20,20,136)
980 PRINT :"DEFINITION = ";D$
990 PRINT : :"PRESS 1 TO MODIFY"
1000 PRINT "{6 SPACES}2 TO START OVER"
1010 PRINT "{6 SPACES}3 TO END PROGRAM";
```

```
1020 CALL KEY(0,K,S)
1030 IF (K=49)+(K=50)THEN 220
1040 IF K=51 THEN 1110 ELSE 1020
1050 FOR L=0 TO 15
1060 IF C$=H$(L,1)THEN 1090
1070 NEXT L
1080 L=L-1
1090 D1$=H$(L,2)
1100 RETURN
1110 PRINT : :
1120 END
```

Chapter 3

Graphics and Sound

Graphics and Sound

Planning Graphics

The screen display for the TI computer is a rectangle of 24 rows and 32 columns. PRINTed characters are in the middle 28 columns (columns 3 through 30), but graphics characters may be placed in all 32 columns. (Some television sets may cut off the outer columns of a 32-column display.)

Designing the Screen

To plan graphics, I use a sheet of graph paper with the rows and columns numbered (Figure 3-1). The numbers start with "1," not "0." I sketch the basic screen with colored pencils.

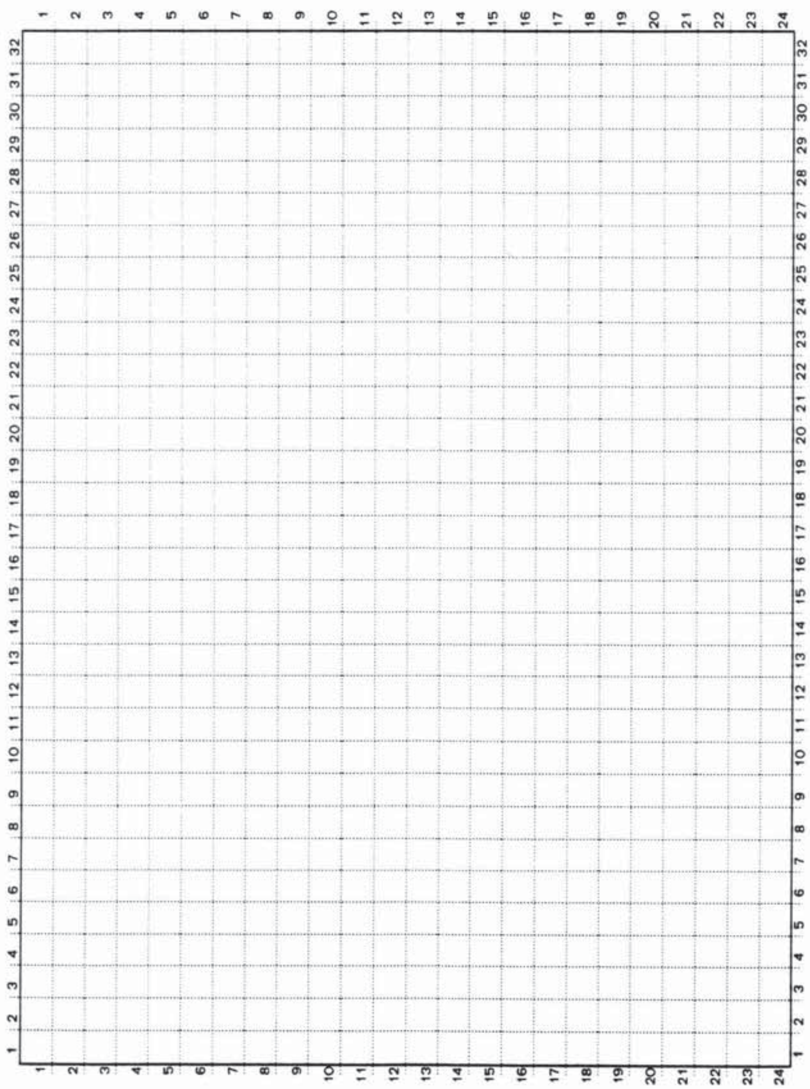
Each square represents one character. It's a good idea to use full squares of color as much as possible. The more odd shapes you use, the more special graphics characters you'll need to define. That will use up memory and slow down the program.

Designing the Characters

Each square of the 24×32 rectangular screen can be thought of as an 8×8 character grid (Figure 3-2). Take part of your basic screen design and draw it in more detail on this high resolution graph paper.As you draw, you'll begin to see where you need to create new characters to express vital details. Try to define as few characters as possible, even if it means your drawing is less than perfect in unimportant areas.

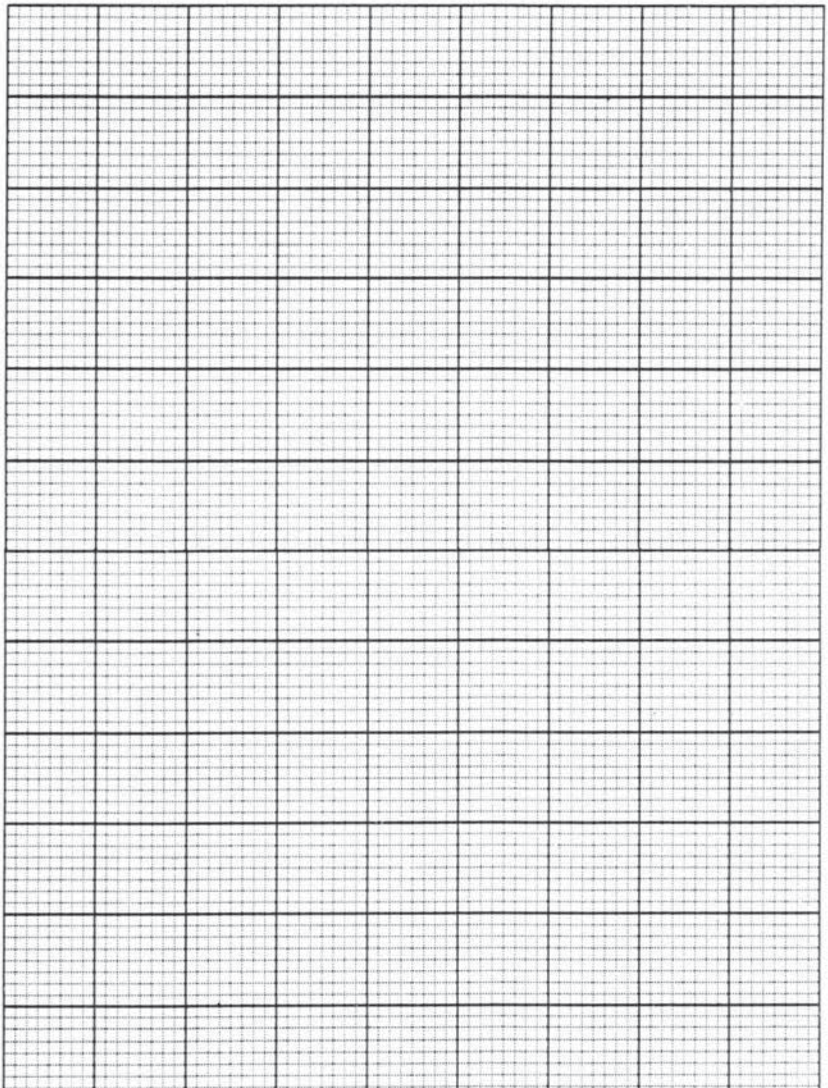
You may be able to use one defined character in several places. For example, in a map of the United States, the same character can be used as part of the slope of the coast of California, part of the border along the Rio Grande of Texas, and part of the southern coast of Florida.

Figure 3-1. Screen Display — a Rectangle of 24 Rows and 32 Columns



Permission granted to photocopy this page.

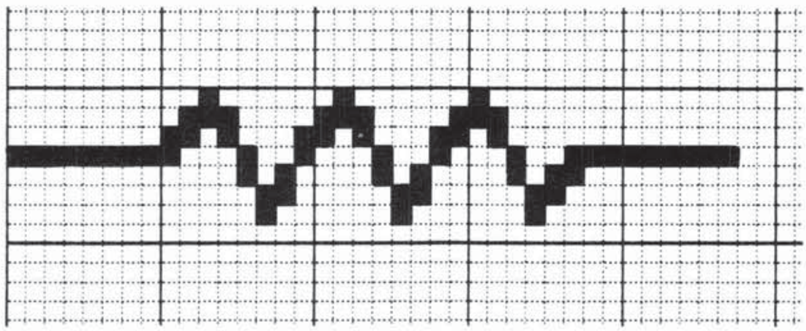
Figure 3-2. High Resolution Graph Paper — Each 8x8 Grid Is One Character



Permission granted to photocopy this page.

Economy in Character Design

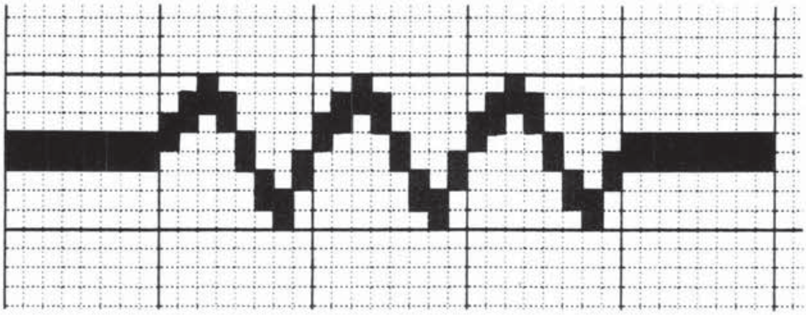
In an electric circuit analysis program, I needed to draw a resistor that looked like this: . I might have done it this way:



The program to produce that drawing takes five different character definitions:

```
100 CALL CLEAR
110 CALL CHAR(128,"000000FF")
120 CALL CHAR(129,"207051890A0604")
130 CALL CHAR(130,"40E1A312140C08")
140 CALL CHAR(131,"80C040272C181")
150 CALL CHAR(132,"000000FC")
160 CALL HCHAR(12,12,128)
170 CALL HCHAR(12,13,129)
180 CALL HCHAR(12,14,130)
190 CALL HCHAR(12,15,131)
200 CALL HCHAR(12,16,132)
210 END
```

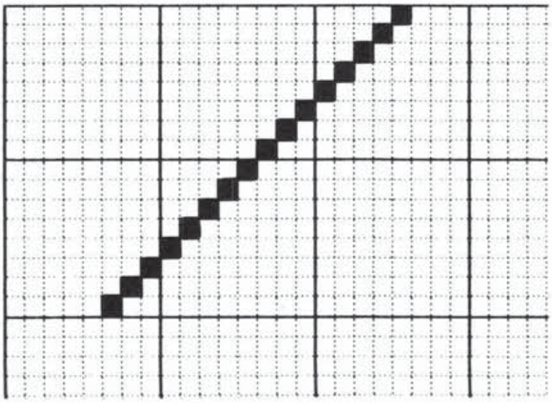
But a more efficient way is:



And the program must define only two characters:

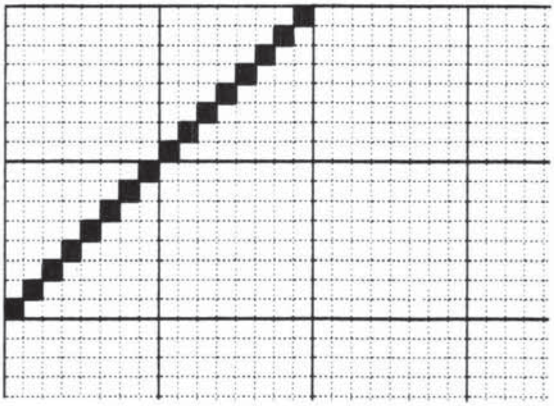
```
100 CALL CLEAR
110 CALL CHAR(128,"000000FFFF")
120 CALL CHAR(129,"2070D08809050602")
130 CALL HCHAR(12,12,128,5)
140 CALL HCHAR(12,13,129,3)
150 END
```

If you have a diagonal line, go through corners of squares to economize on graphics. For example, this method requires two character definitions and places four characters on the screen.



```
100 CALL CLEAR
110 CALL CHAR(128,"081020408")
120 CALL CHAR(129,"000000000010204")
130 CALL HCHAR(12,17,128)
140 CALL HCHAR(12,16,129)
150 CALL HCHAR(13,16,128)
160 CALL HCHAR(13,15,129)
170 END
```

The following method produces the same line, but needs only one character definition and places only two characters on the screen.



```
100 CALL CLEAR
110 CALL CHAR(128,"010204081020408")
120 CALL HCHAR(12,17,128)
130 CALL HCHAR(13,16,128)
140 END
```

Keep Track of Color Sets

Each color set can have up to eight different characters. If you want a magenta hat and a magenta purse, you should design one color set so that only eight characters will be enough to draw both the hat and the purse.

You can, of course, assign the same color to two different sets of eight characters — but not only does that reduce the total number of colors you can place on the screen, it also requires a separate color set definition for *each* set, even if they are the same color. For example, suppose you have designed a green tree that requires nine special characters. Can you redraw the tree so you'll need only eight characters? Then you'd need to define only one color set, rather than two.

Putting Your Characters on the Screen

To display your graphics on the screen, you may use CALL HCHAR, CALL VCHAR, PRINT, or DISPLAY statements. CALL HCHAR is used to draw a horizontal row that repeats the same character. CALL VCHAR is used to draw a vertical column repeating the same character.

Both statements work the same way if you are placing a single character on the screen. Three numbers in parentheses follow the CALL statement. The first number is the row number, the second is the column number, and the third number is the character number. But if you are placing a row or column on the screen, a fourth number is added within the parentheses, telling TI BASIC how many times you want the character repeated. Then, when the statement is executed, the computer starts at the row and column you specified and then repeats the character, either downward (CALL VCHAR) or to the right (CALL HCHAR).

Try to take a good look at the drawing you have designed. If there are places where the same character (for instance, a solid square of color) is repeated several times in a row or column, you can save quite a few program lines by figuring out what arrangement of horizontal or vertical rows you can put on the screen with the fewest single HCHAR and VCHAR calls.

The DISPLAY and PRINT statements give identical results when you are printing something on the screen. Using PRINT will draw something faster than using HCHAR and VCHAR, if there are a lot of characters and very few horizontal and vertical repetitions, and if you don't mind having the screen scroll.

Before using PRINT, redefine the characters you'll need. Be sure *not* to change a character that you will need to use elsewhere, unchanged. If you are going to PRINT the word *MISSISSIPPI,* don't redefine *S.*

In the following example, the characters from 96 through 126 are graphically defined. These are the lowercase letters and a few seldom-used symbols. (See Appendix.) When you use these letters and symbols in the PRINT statement, the listing will show the original letters and symbols; but when you RUN the program, the characters are redefined.

Drawing a Horse

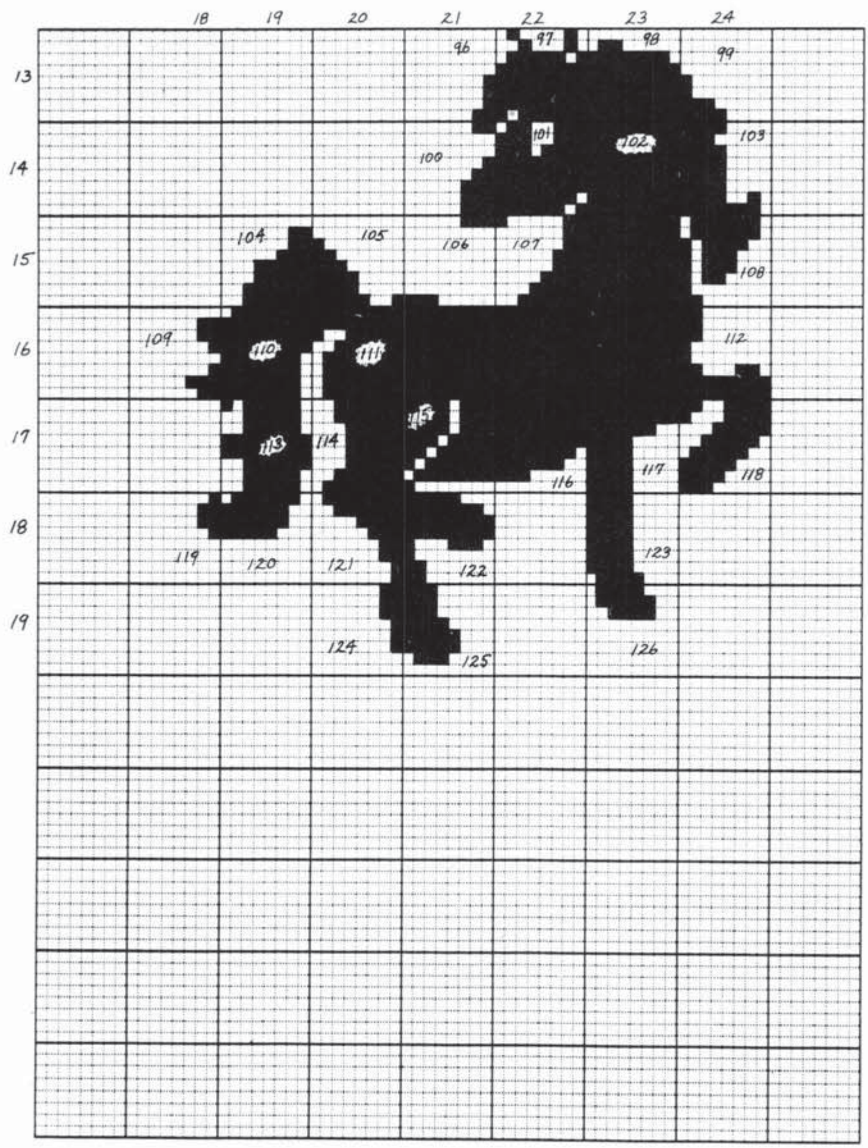
Figure 3-3 is a picture of a horse. Method 1 of drawing the horse uses PRINT statements; the horse appears as the lines on the screen scroll upward. Method 2 uses CALL HCHAR to place each character on the screen.

In Program 3-1, line 110 clears the screen. Lines 120-150 define graphics characters from character number 96 through 126, using definitions in the DATA statements of lines 160-210. Line 220 labels the two drawings. Lines 230-250 use PRINT statements to draw the horse. Lines 260-290 draw the horse on the screen again, a character at a time, READing the row, column, and character number from DATA in lines 300-340. Line 350 keeps the picture on the screen until you press CLEAR. Notice that when you stop the program all characters return to their original definition.

Program 3-1. Horse

```
100 REM HORSE
110 CALL CLEAR
120 FOR C=96 TO 126
130 READ C$
140 CALL CHAR(C,C$)
150 NEXT C
160 DATA 0000000001010103,42227DFFFFFFFFBF,
    0060FEFFFFFFFFFF,000000008080E0F,0300
    000103070707
170 DATA 67E7EFFFFFFFFEFD,FFFFFFFFFFFFFFFF,
    F0E0F0F0F0F2FE,000303071F1F3F3F,000
    080C0E0F0F9
180 DATA 07000000000000E,830303070 70F1F3F,7
    E7EBCB8B8B080C,0003030100010703,7FFFF
    FFFFEFEFEFE
190 DATA FFFF9F3F7F7F7F7F,C0C0C08080C6FFFF,
    BE3E3EFFFF3F3E3E,3F3F1F1F1F1F3F7F,F7F
    7F7EFDFBF8
200 DATA FFFFFFFFFEFCE,FFFFFCF0F0F0F0F,CF8F
    1F3E7CF8F0E,01030301,7EFCFCF8,7F3F0F0
    70303010l
```

Figure 3-3. Graphically Defined Horse



```
210 DATA F8FEFFFF8E80C0C,F0F0F0F0F0F078,0
    30303010101,E0E0E0F0F8F87,787C3C
220 PRINT "METHOD 1:";TAB(15);"METHOD 2:"
230 PRINT : : :"{3 SPACES}`abc":"
    {3 SPACES}defg":" hijkfl"
240 PRINT "mnofffp":" qrstuv"
250 PRINT "wxyz {":"  ¦} ~": : : : :
260 FOR I=1 TO 35
270 READ X,Y,G
280 CALL HCHAR(X,Y,G)
290 NEXT I
300 DATA 13,21,96,13,22,97,13,23,98,13,24,9
    9,14,21,100,14,22,101,14,23,102,14,24,
    103
310 DATA 15,19,104,15,20,105,15,21,106,15,2
    2,107,15,23,102,15,24,108,16,18,109,16,
    19,110
320 DATA 16,20,111,16,21,102,16,22,102,16,2
    3,102,16,24,112,17,19,113,17,20,114,17,
    21,115
330 DATA 17,22,116,17,23,117,17,24,118,18,1
    8,119,18,19,120,18,20,121,18,21,122,18,
    23,123
340 DATA 19,20,124,19,21,125,19,23,126
350 GOTO 350
360 END
```

Remember that the computer performs each statement in turn by number. Plan your graphics so the picture appears in the right order. You will usually want to define the colors before the characters are drawn. You may wish to change the colors at a certain place in the process of drawing. I drew the horse from the head down. You may prefer to draw the head first, then the forebody, the legs, the rest of the body, and finally the tail. You can tell the computer exactly which character must be drawn before another.

Colors

With your TI you may use all 16 colors at any time, even in high resolution graphics. To see all the colors, try this program:

```
100 FOR COLOR=1 TO 16
110 CALL CLEAR
120 CALL SCREEN(COLOR)
130 PRINT "COLOR NUMBER";COLOR
140 CALL SOUND(1000,9999,30)
150 NEXT COLOR
160 CALL CLEAR
170 END
```

Change the 1000 in line 140 if you want to see the colors for a different length of time.

Each color has a number, and these same numbers are used in any statements requiring a color number. Color 1 is transparent. If you have a transparent graphics character, it will be the same color as the existing screen color. However, if you specify CALL SCREEN(1), the screen will be black. Color number 2 is black; and since printing is also black, you will not see a "COLOR NUMBER" message for black in the above program. For the first second of this program, your screen will be black for color 1, and the next second the screen will be black for color 2.

Enter 155 GOTO 100 if you want to keep cycling through the colors — then press CLEAR to stop your program. You may need to adjust your television or monitor to get the proper colors.

Another program to see the colors is:

```
100 CALL CLEAR
110 FOR COL=1 TO 16
120 CALL COLOR(COL,COL,COL)
130 CALL VCHAR(1,COL*2-1,32+8*(COL-1),48)
140 NEXT COL
150 GOTO 150
160 END
```

The colors may vary depending on the screen color, the adjacent colors, and the character shapes. Notice in this program how the sky darkens as more stars appear.

```
100 CALL CLEAR
110 CALL SCREEN(2)
120 CALL COLOR(2,16,1)
```

```
130 CALL HCHAR(INT(RND*24+1),INT(RND*32+1)
    ,42)
140 GOTO 130
150 END
```

Press CLEAR to stop the program.

Setting the Foreground and Background Colors

Each graphics character you define may have a foreground color and a background color. This is done with the statement

CALL COLOR(set,foreground,background)

Keep in mind that if you specify the color to be number 1, it will be the screen color. To get an idea of what the combinations of screen color, foreground color, and background color look like, run this program:

Program 3-2. Color Combinations

```
100 REM COLOR COMBINATIONS
110 DIM C$(16)
120 DATA TRANSP,BLACK,MED GREEN,LT GREEN
130 DATA DARK BLUE,LIGHT BLUE,DARK RED
140 DATA CYAN,MED RED,LIGHT RED,DARK YELLOW
150 DATA LT YELLOW,DARK GREEN,MAGENTA,GRAY,
    WHITE
160 FOR I=1 TO 16
170 READ C$(I)
180 NEXT I
190 CALL CLEAR
200 CALL CHAR(96,"FFFFFFFFFFFFFFFF")
210 CALL CHAR(92,"3C4299A1A199423C")
220 PRINT TAB(6);"COLOR COMBINATIONS"
230 PRINT : : : : : : : :
240 CALL CHAR(97,"FF0055AA55AA00FF")
250 CALL CHAR(98,"0")
260 PRINT "YOU MAY CHOOSE A COLOR"
270 PRINT "NUMBER FROM 1 TO 16."
280 PRINT :"FIRST CHOOSE A SCREEN COLOR"
290 PRINT "THEN A FOREGROUND"
300 PRINT "THEN A BACKGROUND."
310 PRINT : : : :"PRESS ANY KEY TO START."
320 CALL KEY(0,K,S)
```

```
330 IF S<1 THEN 320
340 CALL CLEAR
350 PRINT " 1 TRANSPRNT{3 SPACES}9 MEDIUM R
    ED"
360 FOR I=2 TO 8
370 PRINT I;C$(I);TAB(14);I+8;C$(I+8)
380 NEXT I
390 PRINT : :
400 INPUT "SCREEN COLOR: ":SC
410 CH=SC
420 GOSUB 740
430 IF R=1 THEN 400
440 INPUT "FOREGROUND COLOR: ":F
450 CH=F
460 GOSUB 740
470 IF R=1 THEN 440
480 INPUT "BACKGROUND COLOR: ":B
490 CH=B
500 GOSUB 740
510 IF R=1 THEN 480
520 CALL CLEAR
530 CALL SCREEN(SC)
540 FOR I=1 TO 8
550 CALL COLOR(I,2,16)
560 NEXT I
570 CALL COLOR(9,F,B)
580 CALL HCHAR(7,1,96,198)
590 CALL HCHAR(13,1,97,198)
600 CALL HCHAR(19,1,98,198)
610 PRINT :"SCREEN COLOR";SC;C$(SC)
620 PRINT "FOREGROUND  ";F;C$(F)
630 PRINT "BACKGROUND  ";B;C$(B)
640 PRINT :"PRESS C TO CHANGE; N TO END"
650 CALL KEY(0,K,S)
660 IF K=78 THEN 810
670 IF K<>67 THEN 650
680 CALL CLEAR
690 CALL SCREEN(8)
700 FOR I=1 TO 8
710 CALL COLOR(I,2,1)
720 NEXT I
```

```
730 GOTO 350
740 R=0
750 IF (CH>0)+(CH<17)=-2 THEN 800
760 CALL SOUND(150,131,2,-1,2)
770 PRINT :"SORRY, COLOR NUMBER MUST BE"
780 PRINT "FROM 1 TO 16.  TRY AGAIN.": :
790 R=1
800 RETURN
810 CALL CLEAR
820 END
```

Each character number is assigned to a color set, and there are eight characters per set. The Appendix includes a character chart with the eight-character sets marked off for easy reference. In the stars program above, I used color set 2 because the asterisk, character number 42, is in color set 2.

If you do not define colors in a set, the characters will automatically be black on a transparent background. When you use a CALL COLOR statement, *all* characters in that set will be the color you specified. If there are already characters on the screen, their color will change as soon as the CALL COLOR statement in the program is executed.

Flash and Twinkle

It's possible to make objects flash by using CALL COLOR statements. If you want stars to twinkle, you can use this technique:

```
100 CALL CLEAR
110 CALL SCREEN(5)
120 CALL COLOR(2,16,1)
130 FOR I=1 TO 15
140 CALL HCHAR(INT(RND*24+1),INT(RND*32+1)
    ,42)
150 NEXT I
160 CALL COLOR(2,11,1)
170 CALL COLOR(2,16,1)
180 GOTO 160
190 END
```

Using color 1, you can draw something invisible, then make it appear all at once with another CALL COLOR statement:

```
100 CALL CLEAR
110 CALL COLOR(6,1,1)
120 CALL VCHAR(8,10,72,7)
130 CALL VCHAR(8,14,72,7)
140 CALL HCHAR(11,11,72,3)
150 CALL VCHAR(8,18,73,7)
160 CALL VCHAR(8,23,73,5)
170 CALL HCHAR(14,23,73)
180 CALL COLOR(6,9,9)
190 GOTO 190
200 END
```

Press CLEAR to stop the program.

All the characters in any one set will be the same color. To get varied colors, you need to use characters from different color sets. You will need to plan so that your characters will be in the right color sets. (The **Choreography** section of this chapter illustrates planning and using color with music.)

Solid Squares

There are several ways to get a solid square of color for a character. One way is to assign the same color to both foreground and background in the CALL COLOR statement. In the program above, the characters in set 6 are first defined to be transparent on transparent, or invisible. To make the design appear, the color set is assigned a red foreground and a red background; all characters in the set then become red squares, regardless of the on-off patterns of the characters. Only when you break the program can you see what the actual characters are.

Another way to get a solid colored square is to define the character as "0," or as completely filled: "FFFFFFFFFFFFFFFF". The "Kinder-Art" program (Program 3-3) illustrates this technique. I gave a group of children graph paper marked off in 24 rows and 32 columns, representing the computer screen. The children were instructed to draw a design, people, animals, buildings, or whatever they wanted to by coloring in the squares.

Kinder-Art redefines printable characters as solid colors. Each drawing is converted a line at a time into a string of characters representing the colors. The strings are stored in DATA statements. Depending on the complexity of the artwork, from fifteen to twenty drawings can be put into one program without exceeding available memory.

With the Speech Synthesizer and *Terminal Emulator II* command module, Kinder-Art greets the students by name. (Be sure to try out the pronunciations ahead of time.) If you want to try this program without speech, delete lines 170, 1030, 1090, and 1160.

RESTORE with DATA

When I used this program, I had one child at a time come up and type in his or her name. The computer would then search the array of names to find a match and RESTORE the proper DATA. The computer says "HELLO," followed by the child's name, prints the child's picture, repeats the child's name, and declares, "THIS IS YOUR PICTURE." To continue the cycle for the next child, press ENTER and type in the next name. To end the program, type END instead of a child's name.

When you use DATA statements, the computer usually READs the DATA in order. This is fine for work that is always done in the same order, like defining character sets and drawing screens. But in this case, you don't want to have to bring the students to the computer in any particular order. RESTORE, followed by a line number, tells the computer that the next READ statement should begin with the first item in the DATA statement at that line number. This gives you random rather than sequential access to your data.

Repeating Procedures Keeps a Program Brief

Since each drawing is PRINTed a line at a time, a general procedure can be used for all drawings — no need to figure out individual HCHARs and VCHARs for each. The first character of each color set is defined as "0," and the second character is "FFFFFFFFFFFFFFFF," to give solid colors. Orange and brown are simulated with mixtures. Table 3-1 gives the character and the square of color it represents.

Table 3-1. Characters and Colors for Kinder-Art Program

Character	Color	Command
(space)	Cyan	CALL COLOR(1,3,8)
!	Green	
(	Red	CALL COLOR(2,12,7)
)	Light Yellow	
*	Orange	
0	Dark Yellow	CALL COLOR(3,14,11)
1	Magenta	
2	Brown	
8	Blue	CALL COLOR(4,16,5)
9	White	
@	Black	Color set 5 is already black

Only 23 lines of the drawing are PRINTed, so one line remains to print the child's name.

Recognizing Strings in an Array

The list of all the names in the class is READ in as an array A$(I). The DATA for the drawings is in the same order as the children's names in the A$(I) array. When a child's name is entered, the program compares it with each name in A$(I) to determine what position the DATA is in, so the program can branch to the appropriate RESTORE statement. I did not RESequence the line numbers in this program, so you could more easily see how to add DATA.

I included only two of the pictures to illustrate how to arrange the DATA. You will, of course, draw your own pictures and change the names when you use this program.

In the DATA statements, remember that blank lines are included as "" and that leading spaces require quote marks: " @@ 889", not "@@ 889"

What's Happening in "Kinder-Art"

Lines	
150	Dimension array for number of names.
160	Clear screen.
170	Open speech output device.
180-260	Define characters and colors for graphics.
270-310	READ array of names.
320-350	Clear screen and receive child's name.

360-400	Compare input name with array of possible names and determine position of name.
410-680	RESTORE appropriate DATA depending on name.
1000-1030	Clear screen, PRINT child's name on screen, and greet child by name with Speech Synthesizer.
1040-1070	Draw child's picture.
1080-1090	PRINT child's name on screen. Say child's name and "This is your picture."
1100-1110	Wait until ENTER is pressed before continuing.
1150-1170	If END is entered, clear screen, CLOSE speech device, and stop program.
2000-2050	DATA for Bob's picture.
2100-2170	DATA for Cindy's picture.
2200-	User adds more DATA here.

Program 3-3. Kinder-Art

```
100 CALL CLEAR
110 CALL CHAR(92,"3C4299A1A199423C")
120 PRINT TAB(8);"KINDER-ART": : : : : : :
    : : : :
130 FOR I=1 TO 600
140 NEXT I
150 DIM A$(14)
160 CALL CLEAR
170 OPEN #1:"SPEECH",OUTPUT
180 FOR I=32 TO 56 STEP 8
190 CALL CHAR(I,"0")
200 CALL CHAR(I+1,"FFFFFFFFFFFFFFFF")
210 NEXT I
220 CALL CHAR(64,"FFFFFFFFFFFFFFFF")
222 CALL CHAR(50,"5BB55BB55BB55BB5")
224 CALL CHAR(42,"AA55AA55AA55AA55")
230 CALL COLOR(1,3,8)
240 CALL COLOR(2,12,7)
250 CALL COLOR(3,14,11)
260 CALL COLOR(4,16,5)
270 RESTORE 310
280 FOR I=1 TO 14
```

```
290 READ A$(I)
300 NEXT I
310 DATA BOB,CINDY,CHERY,RICHARD,RANDY,LENA
    ,ANDY,AURA,GRANT,KELLY,JENNIE,ANGELA,
    BRYAN,LEWIS
320 CALL CLEAR
330 PRINT "TYPE YOUR NAME": : :
340 INPUT NAME$
350 IF NAME$="END" THEN 1150
360 FOR I=1 TO 14
370 IF NAME$=A$(I)THEN 410
380 NEXT I
390 PRINT : :"DID YOU TYPE YOUR NAME": :"CO
    RRECTLY?": :"DO IT AGAIN PLEASE.": : :
400 GOTO 340
410 ON I GOTO 420,440,460,480,500,520,540,5
    60,580,600,620,640,660,680
420 RESTORE 2000
430 GOTO 1000
440 RESTORE 2100
450 GOTO 1000
460 RESTORE 2200
470 GOTO 1000
480 RESTORE 2300
490 GOTO 1000
500 RESTORE 2400
510 GOTO 1000
520 RESTORE 2500
530 GOTO 1000
540 RESTORE 2600
550 GOTO 1000
560 RESTORE 2700
570 GOTO 1000
580 RESTORE 2800
590 GOTO 1000
600 RESTORE 2900
610 GOTO 1000
620 RESTORE 3000
630 GOTO 1000
640 RESTORE 3100
650 GOTO 1000
```

```
660 RESTORE 3200
670 GOTO 1000
680 RESTORE 3300
1000 CALL CLEAR
1010 READ N$
1020 PRINT NAME$
1030 PRINT #1:"^HELLO.";N$
1040 FOR I=1 TO 23
1050 READ D$
1060 PRINT D$
1070 NEXT I
1080 PRINT NAME$;
1090 PRINT #1:N$;". ^THIS IS YOUR PICTURE."
1100 CALL KEY(Ø,K,S)
1110 IF K<>13 THEN 1100 ELSE 320
1150 CALL CLEAR
1160 CLOSE #1
1170 STOP
2000 DATA ^BOB,"{4 SPACES}(((((((((((((((("
     ,"{4 SPACES}(*****(((******(","{4 SPAC
     ES}(*****(((******(",(((((((((((((((((
     ((((((("
2010 DATA (((((((((((((((((((((((,(8888(↓↓
     ↓↓(((↓↓↓↓(88888(,(8888(↓↓↓↓(((↓↓↓↓(88
     888(,(8888(↓↓↓↓(((↓↓↓↓(88888(
2020 DATA (8888(↓↓↓↓(((↓↓↓↓(88888(,(8888(↓↓
     ↓↓(((↓↓↓↓(88888(,(8888(↓↓↓↓(((↓↓↓↓(88
     888(,(8888(↓↓↓↓(((↓↓↓↓(88888(
2030 DATA (8888(↓↓↓↓(((↓↓↓↓(88888(,(8888(↓↓
     ↓↓(((↓↓↓↓(88888(,(8888(↓↓↓↓(((↓↓↓↓(88
     888(
2040 DATA (88((((((((((((((((((88(,(88(1111
     11(((1111111(88(,(88(111111(((1111111
     (88(,(88(111111(((1111111(88(
2050 DATA (88(111111(((1111111(88(,((((((((
     (((((((((((((,(*********(((*******
     ***(,(*********(((**********(
2100 DATA ^SINNDY,999111119999999999999911
     111,9991{4 SPACES}1@@@@999@@@@1
     {4 SPACES}1,9991↓↓↓ 1999@@@@9991 ↓↓↓ 1
```

```
2110 DATA 9991 ¡11¡ 199@@@@991 ¡11¡ 1,9991
     ¡1(1¡ 11@@@@@11 ¡1(1¡ 1,9991 ¡11¡ 1
     1@@@@@1 1 ¡11¡ 1
2120 DATA 99991 ¡¡ 1  1@@@@@1  1 ¡¡ 19,9999
     91  1 ¡ 1@@@@@1 ¡ 1  199,9999911 ¡¡
     1@@@@@1 ¡¡ 11999
2130 DATA 9999991 ¡1¡ 1@@@@@1 ¡1¡ 1999,9999
     91 ¡11¡ 1@@@@@1 ¡11¡ 199,99991 ¡1(1¡
     1@@@@@1 ¡1(1¡ 19
2140 DATA 9991 ¡1((1¡ 1@@@@@1 ¡1((1¡ 1,9991
     ¡1((1¡ 1@@@@@1 ¡1((1¡ 1,9991 ¡1((1¡
     1@@@@@1 ¡1((1¡ 1
2150 DATA 99991 ¡1(1¡ 1@@@@@1 ¡1(1¡ 19,9999
     91 ¡11¡ 1@@@@@1 ¡11¡ 199,999991 ¡¡¡¡
     1@@@@@1 ¡¡¡¡ 199
2160 DATA 999991 ¡   ¡ 1@@@@@1 ¡   ¡ 199,9999
     991 11 11@@@@@11 11 1999,999999919919
     9@@@@@9919919999
2170 DATA 9999999999999999999999999999,9999
     999999999999999999999999
2180 END
```

Music

Music is produced by using a CALL SOUND statement. In the parentheses following the CALL SOUND statement, the first number (parameter) is the duration of the sound in milliseconds. It can be any integer from 1 to 4250.

The second parameter is the frequency (pitch) of the tone you want to hear (for example, the note A is 440). The frequency may be from 110, low A on the bass clef, to 44733, which is out of human hearing range. You may specify any number between these limits: you are not limited just to the notes of a scale.

The third parameter is the loudness, and can range from 0 (loudest) through 30 (softest). Of course, the volume also depends on the loudness setting of your monitor or television set.

You may specify up to three musical tones, with a loudness for each, in one statement; thus you may play a three-note chord.

Setting Sound Durations

I like to specify the tempo (speed) before I use any CALL SOUND statements by assigning values to variables and using them as the duration parameter in each CALL SOUND statement. This method makes it possible to change the tempo of the whole tune by changing only the variable assignment rather than each CALL SOUND statement. For example:

```
100 T=350
110 CALL SOUND(T,440,2)
120 CALL SOUND(T/2,554,2)
130 CALL SOUND(T/2,659,2)
140 CALL SOUND(2*T,880,2)
150 END
```

Line 100 sets the variable T to represent a duration of 350 milliseconds. If I want T to be a quarter note, T/2 is an eighth note and 2*T is a half note. Line 110 plays a quarter note, lines 120 and 130 play eighth notes, and line 140 plays a half note. To change the tempo of the arpeggio here, change line 100 to have a different value for T.

This program allows the user to INPUT a number for the duration. The number entered will determine how fast or how slow the computer plays the tune.

Program 3-4. Musical Tempo Demonstration

```
100 CALL CLEAR
110 PRINT TAB(8);"** MUSIC **"
120 CALL CHAR(64,"3C4299A1A1994237")
130 PRINT : : :
140 PRINT : : :"THIS COMPUTER CAN PLAY"
150 PRINT :"UP TO THREE TONES AT A TIME"
160 PRINT :"USING ONE STATEMENT."
170 PRINT : :"HERE IS A SAMPLE TUNE."
180 PRINT :"YOU MAY ENTER A NUMBER"
190 PRINT :"FROM 6 TO 1062 FOR DURATION."
200 PRINT :"FOR EXAMPLE, DURATION = 450": :
    :
210 INPUT "DURATION = ":T
220 IF (T>=6)+(T<=1062)=-2 THEN 250
```

```
230 PRINT :"SORRY, 6<D<1062": :
240 GOTO 210
250 CALL CLEAR
260 PRINT "DURATION =";T
270 CALL SOUND(T,392,1,330,6,131,9)
280 CALL SOUND(T/3,262,2,165,8)
290 CALL SOUND(T/6,330,2,165,8)
300 CALL SOUND(T/6,330,2,196,8)
310 CALL SOUND(T/3,392,2,196,8)
320 CALL SOUND(T,349,2,196,7,123,9)
330 CALL SOUND(T,349,1,196,6,123,8)
340 CALL SOUND(T/2,330,2,165,7,131,9)
350 CALL SOUND(T/2,294,2,165,7,131,9)
360 CALL SOUND(T/2,262,2,165,7,131,9)
370 CALL SOUND(T/2,294,2,165,7,131,9)
380 CALL SOUND(2*T,330,2,165,6,131,8)
390 CALL SOUND(T,440,1,220,6,175,6)
400 CALL SOUND(T*3/4,440,2,175,6)
410 CALL SOUND(T/4,440,3)
420 CALL SOUND(T,523,1,220,6,175,8)
430 CALL SOUND(T,523,2,220,7,175,9)
440 CALL SOUND(T/2,659,1,208,7,165,9)
450 CALL SOUND(T/2,587,1,208,7,165,9)
460 CALL SOUND(T/2,523,1,208,7,165,9)
470 CALL SOUND(T/2,587,1,208,7,165,9)
480 CALL SOUND(T*2,659,0,165,6,131,8)
490 CALL SOUND(T/3,784,0,165,6,131,8)
500 CALL SOUND(T/3,659,1)
510 CALL SOUND(T/3,523,1)
520 CALL SOUND(T/3,392,1,165,6,131,8)
530 CALL SOUND(T/3,330,1)
540 CALL SOUND(T/3,262,1)
550 CALL SOUND(T*2,330,0,196,6,131,8)
560 CALL SOUND(T/4,294,1,175,6,123,8)
570 CALL SOUND(T*4,262,2,165,6,131,8)
580 PRINT : : :"TRY AGAIN? (Y/N)": :
590 CALL KEY(0,K,S)
600 IF K=78 THEN 620
610 IF K=89 THEN 210 ELSE 590
620 CALL CLEAR
630 END
```

Setting Up Pitch Values

At the beginning of the program, you may wish to assign to variables the frequencies or pitches you want to use in the music in your program. If you use the letter names of the notes as the corresponding variable names, the values will be much easier to remember.

```
100 T=400
110 C=262
120 D=294
130 E=330
140 F=349
150 G=392
160 CALL SOUND(T,E,1)
170 CALL SOUND(T,D,2)
180 CALL SOUND(T,C,2)
190 CALL SOUND(T,D,2)
200 CALL SOUND(T,E,2)
210 CALL SOUND(T,E,1)
220 CALL SOUND(T*2,E,0)
230 END
```

If you read music, you can translate any written music to the computer, though only three notes will play at any one time. You can also use a frequency chart (Figure 3-3).

Figure 3-3. Frequencies of the Musical Scale

Melody and Accompaniment

It's convenient to use the first frequency and volume as the melody tone, then the second and third frequencies and volumes as the accompaniment tones. This is just so you can keep track of which number is the melody; the order doesn't matter to the computer.

If you start to run out of memory in a piece, you can go back to the CALL SOUND statements and delete accompaniment tones. It's easy to find them if the first frequency and volume are always the melody. Also, you might want to use a lower volume setting for the low notes in order to emphasize the melody note.

One method of accompaniment is to use two notes of the three basic chords of the key in which the music is written. For example, if a song is written in the key of C Major (no sharps and no flats), the basic chords are C, F, and G_7. When you play Middle C as the melody note, two notes of the C major triad may be chosen for accompaniment — perhaps E and G. The program statement is:

```
CALL SOUND(400,262,2,196,6,165,8)
```

Translating Published Music

Rather than compose your own music, you may prefer to work from a copy of published music. The top note is usually the melody note. Any two notes written directly under that note may be chosen for the accompaniment in your CALL SOUND statement.

If you have two successive CALL SOUND statements which specify the same frequency and volume, the notes may sound like one long note rather than two separate notes. To make the notes sound distinct, just change the volume number for one of the notes. To make a bass note sound tied while two melody notes are played, keep the frequency and the volume the same in both statements.

Here is the written music for the Musical Tempo Demonstration (Program 3-4) so you can see how I translated the music for the computer version.

What You Can Do with Music

There are all kinds of applications using the computer's music feature. Writing music on the TI-99/4A is fun because you can immediately hear any changes you want to implement as you are composing. You can put some music on the computer and have a sing-along.

Or perhaps you are trying to learn a piece that has a difficult rhythm. Program it onto the computer, play it at a slow tempo, then gradually increase the tempo as you practice the piece along with the computer.

You might also wish to program an accompaniment on the computer, then play along with a melody instrument like the clarinet or violin.

Musical tones also work well in interactive programs. For example, in an educational program you might use a happy musical interlude for a correct answer.

Learning Musical Notation

''Name the Notes'' and ''Music Steps and Chords'' are educational programs for music students. A piano or organ teacher can use them before or after a student's regularly scheduled lesson as enrichment, drill, or as a different approach to teaching. Music students could use these programs at home for additional practice. A programmer who doesn't read music may be interested in using the first program to learn enough about reading notes to incorporate printed music in his or her own programs.

''Name the Note'' is a tutorial program designed for the beginning music student. The first option, Keyboard, shows the letter names of the keys on a piano or organ keyboard and then presents a drill of ten keys chosen at random. A question mark appears on a key, and the student must press the correct letter name. When the correct letter is pressed, the name of the key appears and the tone is played.

The second and third options are ''Treble Clef'' and ''Bass Clef.'' These two sections display the appropriate staff and clef, and present words and phrases to help the student remember the letter names of the notes. A drill of ten notes is then presented.

How "Name the Note" Works

Lines

110-120	Option Base 1 sets the lowest numbers in the array to 1 rather than 0. N is an array that keeps track of the ASCII code of the letter name of a note and the note's frequency.

130-220	Define graphics characters and colors.
230-320	Print title screen with options; branch appropriately after student presses number of option.
330-530	Subroutine to draw piano keyboard (may be entered at line 380).
540-580	Subroutine to PRINT "Press Enter" and wait for student to respond.
590-630	Subroutine to play arpeggio for right answer.
640-710	Subroutine to PRINT "Try Again" and wait for student to respond.
720-770	Subroutine to define graphics characters.
780-820	Subroutine to draw treble clef and staff.
830-860	Subroutine to draw bass clef and staff.
870-920	Subroutine to draw graphics characters that put phrases on staff.
930-1090	Instruction for learning the names of the keys.
1100-1370	Drill for keyboard.
1380-1450	Define graphics characters for treble clef.
1460-1600	Present instruction for learning the names of the treble notes.
1610-2020	Drill for treble clef.
2030-2070	Define graphics characters for bass clef.
2080-2210	Present instruction for learning the names of the bass notes.
2220-2260	Drill for bass clef.
2270	End.

Program 3-5. Name the Note

```
100 REM{3 SPACES}NOTES
110 OPTION BASE 1
120 DIM N(11,2)
130 CALL CHAR(96,"000000FF")
140 L$="````````````````````````"
150 CALL CHAR(64,"3C4299A1A199423C")
160 RESTORE 170
170 DATA 000000000000F30C,000000000000F00C03,0
    10202040404020 1,7088083040400004,80404
    020202040 8,8060100F,010608F
180 FOR C=144 TO 150
```

```
190 READ C$
200 CALL CHAR(C,C$)
210 NEXT C
220 CALL COLOR(15,7,1)
230 CALL CLEAR
240 CALL COLOR(4,2,1)
250 CALL SCREEN(8)
260 PRINT " N A M E{3 SPACES}T H E
    {3 SPACES}N O T E"::::"{6 SPACES}CHOOS
    E:"::::"{6 SPACES}1 KEYBOARD"
    ::"{6 SPACES}2 TREBLE CLEF"
270 PRINT :"{6 SPACES}3 BASS CLEF"::"
    {6 SPACES}4 END PROGRAM"::::::
280 CALL KEY(Ø,K,S)
290 IF (K<49)+(K>52)THEN 280
300 CALL CLEAR
310 F=K-48
320 ON F GOTO 930,1380,2030,2270
330 CALL CHAR(152,"Ø")
340 CALL CHAR(153,"FFFFFFFFFFFFFFFF")
350 CALL CHAR(154,"Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1")
360 CALL CHAR(155,"8Ø8Ø8Ø8Ø8Ø8Ø8Ø8")
370 CALL COLOR(16,2,16)
380 CALL HCHAR(1,1,152,480)
390 RESTORE 400
400 DATA 3,6,12,15,18,24,27,9,21,30
410 FOR C=1 TO 7
420 READ J
430 CALL VCHAR(1,J,153,12)
440 CALL VCHAR(1,J+1,153,12)
450 CALL VCHAR(13,J,154,3)
460 CALL VCHAR(13,J+1,155,3)
470 NEXT C
480 FOR C=1 TO 3
490 READ J
500 CALL VCHAR(1,J,154,15)
510 CALL VCHAR(1,J+1,155,15)
520 NEXT C
530 RETURN
540 PRINT TAB(16);"PRESS <ENTER>";
550 CALL KEY(Ø,K,S)
```

```
560 IF K<>13 THEN 550
570 CALL HCHAR(24,18,32,13)
580 RETURN
590 CALL SOUND(150,262,2)
600 CALL SOUND(150,330,2)
610 CALL SOUND(150,392,2)
620 CALL SOUND(300,523,2)
630 RETURN
640 FOR C=1 TO 15
650 CALL HCHAR(24,C+12,ASC(SEG$("TRY AGAIN
    (Y/N)",C,1)))
660 NEXT C
670 CALL KEY(0,K,S)
680 IF K=78 THEN 230
690 IF K<>89 THEN 670
700 CALL HCHAR(24,13,32,15)
710 RETURN
720 READ C
730 FOR J=97 TO C
740 READ A$
750 CALL CHAR(J,A$)
760 NEXT J
770 RETURN
780 CALL CLEAR
790 PRINT TAB(9);"TREBLE CLEF":::
800 PRINT "{3 SPACES}a":"{3 SPACES}b":"```c
    ``"&L$:"  de":"``fg``"&L$:" h i":"`jklm
    `"&L$
810 PRINT " nop q":"`rst`t"&L$:" u v w":"``
    xyz`"&L$:"{3 SPACES}{":"{3 SPACES}{":"
    {3 SPACES}¦"
820 RETURN
830 CALL CLEAR
840 PRINT TAB(10);"BASS CLEF":::
850 PRINT ::"``abc`"&L$:" d{3 SPACES}e k":"
    `f```g"&L$:"{5 SPACES}h k":"`````i"&L$:
    "{4 SPACES}h":"````i`"&L$:"{3 SPACES}j"
    :"```````"&L$:::
860 RETURN
870 READ C,J
880 FOR I=C TO J
890 READ K,G
```

```
900 CALL HCHAR(K,I,G)
910 NEXT I
920 RETURN
930 PRINT "A KEYBOARD HAS GROUPS OF"::"TWO
    BLACK KEYS AND GROUPS"::"OF THREE BLA
    CK KEYS.":::
940 GOSUB 330
950 GOSUB 540
960 PRINT "THE NAMES OF THE KEYS ARE"::"THE
     FIRST 7 LETTERS."::
970 RESTORE 980
980 DATA 67,68,69,70,71,65,66,67,68,69,70
990 FOR J=2 TO 32 STEP 3
1000 READ G
1010 CALL HCHAR(9,J,G)
1020 NEXT J
1030 GOSUB 540
1040 CALL CLEAR
1050 PRINT "YOU MAY REMEMBER THAT JUST"::"L
     EFT OF THE TWO BLACK KEYS"::"IS THE K
     EY CALLED 'C'.":::
1060 GOSUB 380
1070 CALL HCHAR(14,2,67)
1080 CALL HCHAR(14,23,67)
1090 GOSUB 540
1100 CALL CLEAR
1110 CALL SCREEN(12)
1120 PRINT TAB(8);"NAME THE KEY"::::
1130 GOSUB 380
1140 RESTORE 1150
1150 DATA 67,262,68,294,69,330,70,349,71,39
     2,65,440,66,494,67,523,68,587,69,659,
     70,698
1160 FOR C=1 TO 11
1170 READ N(C,1),N(C,2)
1180 NEXT C
1190 FOR T=1 TO 10
1200 RANDOMIZE
1210 X=INT(RND*11+1)
1220 J=3*X-1
1230 CALL HCHAR(14,J,63)
1240 CALL KEY(0,K,S)
```

```
1250 CALL COLOR(4,16,16)
1260 CALL COLOR(4,7,16)
1270 IF S<1 THEN 1240
1280 IF K=N(X,1)THEN 1310
1290 CALL SOUND(500,-8,2)
1300 GOTO 1240
1310 CALL HCHAR(14,J,K)
1320 CALL SOUND(600,N(X,2),2)
1330 CALL SOUND(1,N(X,2),30)
1340 CALL HCHAR(14,J,152)
1350 NEXT T
1360 GOSUB 640
1370 GOTO 1190
1380 PRINT TAB(9);"TREBLE CLEF":::
1390 RESTORE 1400
1400 DATA 124,0000384482828282,828282828282
     8282,848488FF889090A,0000000000010102
     ,A0A0C0C0C040404,040810FF2040808
1410 DATA 404040FF2020202,0102040408101 01,2
     0202020202020202,202040FF4040404,000000
     FF0304081,101010FF1010101
1420 DATA 000000FF18040201,808080808080808,
     1020202020404 04,0808080808080808,C020
     201010080808,808080FF4040402
1430 DATA 404040FF2020100C,040404FF04040404
     ,2020101008040403,0202020202010101,08
     0808101020408,804038FF
1440 DATA 010101FF01010101,030C30FF,0101010
     101010101,010101011111 10E
1450 GOSUB 720
1460 GOSUB 800
1470 PRINT :"THINK OF THE WORD 'FACE' FOR":
     :"THE NOTE NAMES ON SPACES."::
1480 CALL HCHAR(14,14,70)
1490 CALL HCHAR(12,17,65)
1500 CALL HCHAR(10,20,67)
1510 CALL HCHAR(8,23,69)
1520 GOSUB 540
1530 GOSUB 780
1540 PRINT :"MEMORIZE THIS PHRASE TO HELP":
     :"LEARN LINE NOTES E G B D F."::
1550 RESTORE 1560
```

```
1560 DATA 9,28,15,69,15,86,15,69,15,82,15,8
     9,13,71,13,79,13,79,13,68,11,66,11,79
     ,11,89,9,68,9,79
1570 DATA 9,69,9,83,7,70,7,73,7,78,7,69
1580 GOSUB 870
1590 GOSUB 540
1600 GOSUB 780
1610 RESTORE 1620
1620 DATA 70,698,69,659,68,587,67,523,66,49
     4,65,440,71,392,70,349,69,330
1630 FOR C=1 TO 9
1640 READ N(C,1),N(C,2)
1650 NEXT C
1660 PRINT TAB(8);"NAME THE NOTE"::::
1670 FOR T=1 TO 10
1680 RANDOMIZE
1690 X=INT(9*RND+1)
1700 J=5+X
1710 CALL HCHAR(J,20,144)
1720 CALL HCHAR(J,21,145)
1730 CALL HCHAR(J+1,19,146)
1740 CALL HCHAR(J+1,22,148)
1750 CALL HCHAR(J+2,20,149)
1760 CALL HCHAR(J+2,21,150)
1770 CALL HCHAR(J+1,21,147)
1780 CALL SOUND(150,1397,4)
1790 CALL KEY(0,K,S)
1800 IF S<1 THEN 1790
1810 IF K=N(X,1)THEN 1840
1820 CALL SOUND(200,-5,4)
1830 GOTO 1790
1840 CALL HCHAR(J+1,21,N(X,1))
1850 IF F=2 THEN 1900
1860 FOR I=N(X,2)TO N(X,2)+48 STEP 12
1870 CALL SOUND(150,I,2)
1880 NEXT I
1890 GOTO 1910
1900 CALL SOUND(500,N(X,2),2)
1910 CALL SOUND(1,N(X,2),30)
1920 IF X/2=INT(X/2)THEN 1970
1930 CALL HCHAR(J,20,32,2)
```

```
1940 CALL HCHAR(J+1,19,96,4)
1950 CALL HCHAR(J+2,20,32,2)
1960 GOTO 2000
1970 CALL HCHAR(J,20,96,2)
1980 CALL HCHAR(J+1,19,32,4)
1990 CALL HCHAR(J+2,20,96,2)
2000 NEXT T
2010 GOSUB 640
2020 GOTO 1670
2030 PRINT TAB(10);"BASS CLEF":::
2040 RESTORE 2050
2050 DATA 107,000000FF0F10608,000000FFFF,00
     0000FF80700C03,0102041810202 04,804020
     2010080804,40583CFF3C18
2060 DATA 040202FF01010101,0101020204040408
     ,080810FF2040808,0101020408,000E1F1F1
     F0E
2070 GOSUB 720
2080 GOSUB 850
2090 PRINT :"LEARN THIS PHRASE FOR THE"::"N
     OTES ON SPACES, A C E G."::
2100 RESTORE 2110
2110 DATA 13,30,15,65,15,76,15,76,15,32,13,
     67,13,79,13,87,13,83,13,32,11,69,11,6
     5,11,84,11,32
2120 DATA 9,71,9,82,9,65,9,83,9,83
2130 GOSUB 870
2140 GOSUB 540
2150 GOSUB 830
2160 PRINT :"THIS PHRASE HELPS YOU KNOW"::"
     THE LINE NOTES, G B D F A."::
2170 RESTORE 2180
2180 DATA 8,31,16,71,16,82,16,69,16,65,16,8
     4,14,66,14,73,14,71,12,68,12,79,12,71
     ,12,83,10,70,10,73
2190 DATA 10,71,10,72,10,84,8,65,8,78,8,73,
     8,77,8,65,8,76,8,83
2200 GOSUB 870
2210 GOSUB 540
2220 GOSUB 830
2230 PRINT
```

```
2240 RESTORE 2250
2250 DATA 65,220,71,196,70,175,69,165,68,14
     7,67,131,66,123,65,117,71,110
2260 GOTO 1630
2270 END
```

Teaching Basic Musical Theory

A piano teacher can get bored, discouraged, impatient, or frustrated trying to drill a student in the basic fundamentals of the keyboard. A computer is an ideal teaching aid because it can choose questions randomly, perform repetitious drills without intimidating the student, and, with effective graphics and sound, can encourage the student to have fun learning.

"Music Steps and Chords" is designed as a tutorial to supplement the teacher's instructions for distinguishing between half steps and whole steps in music, counting the steps between two notes, and using this counting method to identify basic triads.

Half Step. A half step is a rise or fall in pitch from one piano key to the adjacent key. The program draws a keyboard. Examples of half steps are illustrated with arrows. A quiz asks if the arrow on the keys represents a half step. The tones are sounded so the student will see *and* hear the difference between the two notes. The student presses 1 for yes or 2 for no.

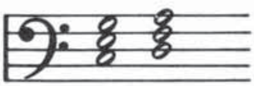
Whole Step. A whole step is equal to two half steps. Again, the program shows this on a keyboard. The quiz for this section asks the student to press 1 for a half step and 2 for a whole step for ten examples. Arrows are drawn and tones are sounded for each problem.

Count the Steps. The third section is a quiz with ten questions. Two keys are randomly chosen and played. The student must indicate the correct number of half steps between the two keys. If the answer is correct, an arpeggio is played. If the answer is incorrect, the correct answer is shown, and arrows for each half step are drawn so the student may see how to get the correct answer.

Identifying Triads. One method of teaching the identification of triads (three-note chords) and the naming of chords is to count the steps between each note of the root chord and find the pattern. Each triad has its own specific pattern of intervals. This counting method is used in this program to identify a major triad, a minor triad, an augmented triad, and a diminished triad.

First, make sure the triad is in the root position: the three notes are *all* on adjacent spaces of the musical staff, or *all* on adjacent lines.

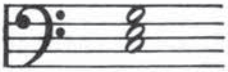
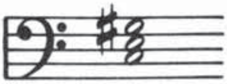
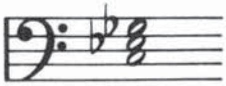
Root Position



Not Root Position

The name of the chord is the name of the bottom note of the root triad.

Next, the number of whole steps between the first and middle note are counted, then the number of steps between the middle and top note. An example is shown on the keyboard with the C chords. The number of steps is always either 1½ or 2.

	C Major	C (2 steps)	E (1½ steps)	G
	C minor	C (1½ steps)	E^{b}(2 steps)	G
	C Augmented	C (2 steps)	E (2 steps)	G#
	C diminished	C (1½ steps)	E^{b}(1½ steps)	G^{b}

Each type of chord is described, played, and illustrated on the keyboard with the number of steps labeled.

A way to learn the chords is to remember that the major chord has 2 steps, then 1½ steps. The minor chord lowers the middle note, so its order is 1½ steps, then 2 steps. The augmented chord starts with the major chord (2,1½), then "augments" or enlarges the chord; so the order is 2 steps, then 2 steps. The diminished chord always starts with the minor chord (1½, 2), then "diminishes" or reduces the chord to 1½ steps, then 1½ steps.

The quiz randomly chooses a beginning note, a middle note either 1½ or 2 steps higher, and a top note either 1½ or 2 steps higher than the middle note. The three notes are sounded separately, then together. The student chooses whether the chord is major, minor, augmented, or diminished.

If the answer chosen is correct, an arpeggio is played. If the answer is incorrect, the number of steps between each note is illustrated and the correct answer is given. After ten chords, the student's score is printed. The student may then choose to return to any section of the program.

How "Music Steps and Chords" Works

Lines	
110	DIMension variables. H(CH) is the Y-coordinate for the key chosen; NN(CH) is the frequency for the key chosen, to be used in the CALL SOUND statements.
120	Branch to the title screen subroutine.
130-210	Print menu screen and wait for student's response; branch appropriately. For a tutorial program, the first-time student should choose each option in order, repeating drills as necessary.
220-290	Subroutine to PRINT message "(PRESS ENTER)" and wait for student's response; clear message.
300-330	Subroutine to PRINT message A$ on the screen without scrolling. Row K and column J must be specified.
340-390	Subroutine to draw graphics characters from DATA statements.
400-570	Subroutines to draw arrows for half steps.
580-930	Subroutines to draw arrows for whole steps.
940-1090	Subroutine to draw keyboard.
1100-1140	Subroutine to play music after a correct answer.
1150-1420	Subroutine for half steps.

1150-1210	Print definition and keyboard.
1220-1250	Print example arrows and wait for student to press ENTER.
1260-1290	Clear arrows and printing.
1300-1400	Print instructions for drill.
1410-1420	Perform drill for half steps, then return to menu screen.
1430-1690	Subroutine for half step or whole step drill.
1430-1440	Randomize choice and perform drill for ten problems.
1450-1460	Randomly choose half step or whole step.
1470-1490	Randomly choose starting key and draw arrow.
1500-1520	Call subroutine for quiz; erase arrow.
1530-1570	Randomly choose starting key for whole step, draw arrow, present quiz, erase arrow.
1580-1590	After ten questions, return.
1600-1660	Subroutine to play tones of starting key and next tone; wait for student's response. If the answer is correct, play an arpeggio; if the answer is incorrect, play a noise.
1670-1690	Subroutine to play noise for incorrect response.
1700-2020	Subroutine to teach whole steps.
1700-1830	Print definition; draw keyboard and example arrows.
1840-1890	Wait for student to press ENTER, then clear arrows and printing.
1900-2000	Print instructions for drill.
2010-2020	Perform drill for whole steps, then return.
2030-2240	Subroutine to perform drill for counting steps between keys.
2030-2070	Print instructions and draw keyboard.
2080-2100	Perform drill for counting steps.
2110-2150	Clear printing, then print score.
2160-2220	If score was 100 percent, play a tune.
2230-2240	Wait for student to press ENTER, then return.
2250-2580	Subroutine to randomly choose keys for drill.
2250-2290	Randomly choose a key from the first nine keys of the keyboard, sound that note, and place an X on the key.
2300-2340	Randomly choose a number of steps ST, sound the correct note, and place an X on the second key.

2350-2360	Wait for student to choose an answer.
2370-2450	If the answer is incorrect, draw arrows on the keyboard, print "½" under each half step to show how the correct answer is obtained, and move another arrow down to the correct choice.
2460-2530	Wait for student to press ENTER, then clear all arrows.
2540-2550	If the answer is correct, play an arpeggio and increment the score.
2560-2580	Replace the X marks on the keys with the original key names; return.
2590-2860	Subroutine to print title screen and define characters.
2590-2610	Clear screen; print title.
2620-2730	Define graphics characters from DATA.
2740-2790	Read in from DATA the Y-coordinate and frequency for each key on the keyboard.
2800-2840	Define color sets.
2850-2860	Change the screen color to yellow and return.
2870-4500	Subroutine to teach basic triads, identifying them by the counting-steps method.
2870-2970	Print instructions about triads; wait for student to press ENTER after each screen of instructions.
2980-3010	Clear screen; print description of C major triad; draw keyboard.
3020-3070	Sound the tones of the chord as the keys are marked.
3080-3110	Illustrate number of steps between keys at top of keyboard.
3120-3400	Wait for student to press ENTER, then change major triad description to minor triad description.
3410-3600	Print description of augmented triad.
3610-3790	Print description of diminished triad.
3800-3840	Print summary table of triads.
3850-3870	Print instructions for drill and draw keyboard.
3880-3940	For ten problems, randomly choose the first key, sound the appropriate note, and print *X* on the key.
3950-3990	Randomly choose the next key three or four half steps above the first note; sound the appropriate note and print *X*.

4000-4040	Randomly choose the top key three or four half steps from middle key; sound the appropriate note and print *X*.
4050-4140	Depending on the steps between notes, determine correct answer.
4150-4180	Play the chord and wait for the student to press an answer; place an asterisk in front of the answer chosen.
4190-4350	If answer is incorrect, label the number of steps between the keys, put an arrow before the correct choice, and wait for the student to press ENTER.
4360-4380	Clear the marked answer.
4390-4430	If the answer is correct, play an arpeggio and increment the score.
4440-4480	Clear the marked keys and the chosen answer; go to the next problem.
4490-4510	Print the final score; return; END.

Program 3-6. Music Steps and Chords

```
100 REM  MUSIC STEPS AND CHORDS
110 DIM H(17),NN(18)
120 GOSUB 2590
130 CALL CLEAR
140 PRINT "CHOOSE":::"1  HALF STEPS"::"2  W
    HOLE STEPS"::"3  COUNT THE STEPS"
150 PRINT :"4  IDENTIFYING TRIADS"::"5  TRI
    ADS QUIZ":::"6  END PROGRAM":::
160 CALL KEY(0,K,S)
170 IF (K<49)+(K>54)THEN 160
180 CALL CLEAR
190 ON K-48 GOSUB 1150,1700,2030,2870,3850,
    210
200 GOTO 130
210 STOP
220 A$="(PRESS ENTER)"
230 FOR Y=1 TO 13
240 CALL HCHAR(24,Y+18,ASC(SEG$(A$,Y,1)))
250 NEXT Y
260 CALL KEY(0,K,S)
270 IF K<>13 THEN 260
```

```
280 CALL HCHAR(24,19,32,13)
290 RETURN
300 FOR I=1 TO LEN(A$)
310 CALL HCHAR(K,J+I,ASC(SEG$(A$,I,1)))
320 NEXT I
330 RETURN
340 READ N
350 FOR I=1 TO N
360 READ X,Y,G
370 CALL HCHAR(X,Y,G)
380 NEXT I
390 RETURN
400 CALL HCHAR(14,Y,112)
410 CALL HCHAR(14,Y+1,114)
420 RETURN
430 CALL HCHAR(14,Y,98)
440 CALL HCHAR(14,Y+1,99)
450 RETURN
460 CALL HCHAR(12,Y,106)
470 CALL HCHAR(13,Y+1,116)
480 RETURN
490 CALL HCHAR(12,Y,97)
500 CALL HCHAR(13,Y+1,96)
510 RETURN
520 CALL HCHAR(13,Y,115)
530 CALL HCHAR(12,Y+1,107)
540 RETURN
550 CALL HCHAR(13,Y,96)
560 CALL HCHAR(12,Y+1,97)
570 RETURN
580 CALL HCHAR(14,Y,112)
590 CALL HCHAR(14,Y+1,113,2)
600 CALL HCHAR(14,Y+3,114)
610 RETURN
620 CALL HCHAR(14,Y,96,4)
630 CALL HCHAR(14,Y+1,98)
640 CALL HCHAR(14,Y+2,99)
650 RETURN
660 CALL HCHAR(11,Y,104)
670 CALL HCHAR(11,Y+1,113)
680 CALL HCHAR(11,Y+2,105)
690 RETURN
```

```
700 CALL HCHAR(11,Y,97)
710 CALL HCHAR(11,Y+1,96)
720 CALL HCHAR(11,Y+2,97)
730 RETURN
740 CALL HCHAR(13,Y,117)
750 CALL HCHAR(13,Y+1,118)
760 CALL HCHAR(12,Y+2,119)
770 CALL HCHAR(12,Y+3,108)
780 RETURN
790 CALL HCHAR(13,Y,98)
800 CALL HCHAR(13,Y+1,99)
810 CALL HCHAR(12,Y+2,96)
820 CALL HCHAR(12,Y+3,97)
830 RETURN
840 CALL HCHAR(12,Y,109)
850 CALL HCHAR(12,Y+1,120)
860 CALL HCHAR(13,Y+2,121)
870 CALL HCHAR(13,Y+3,122)
880 RETURN
890 CALL HCHAR(12,Y,97)
900 CALL HCHAR(12,Y+1,96)
910 CALL HCHAR(13,Y+2,98)
920 CALL HCHAR(13,Y+3,99)
930 RETURN
940 CALL HCHAR(1,1,96,480)
950 RESTORE 960
960 DATA 3,6,12,15,18,24,27,9,21,30
970 FOR I=1 TO 7
980 READ Y
990 CALL VCHAR(1,Y,97,12)
1000 CALL VCHAR(1,Y+1,97,12)
1010 CALL VCHAR(13,Y,98,3)
1020 CALL VCHAR(13,Y+1,99,3)
1030 NEXT I
1040 FOR I=1 TO 3
1050 READ Y
1060 CALL VCHAR(1,Y,98,15)
1070 CALL VCHAR(1,Y+1,99,15)
1080 NEXT I
1090 RETURN
1100 CALL SOUND(150,262,2)
1110 CALL SOUND(150,330,2)
```

```
1120 CALL SOUND(150,392,2)
1130 CALL SOUND(300,523,2)
1140 RETURN
1150 PRINT "{4 SPACES}H A L F{3 SPACES}S T
     E P S"::::::::::
1160 PRINT "A HALF STEP IS FROM ONE KEY  ":
     "TO THE VERY NEXT KEY.":::
1170 FOR I=1 TO 4
1180 CALL HCHAR(19,4+I,127+I)
1190 NEXT I
1200 CALL HCHAR(22,10,113,9)
1210 GOSUB 940
1220 RESTORE 1230
1230 DATA 8,12,4,106,13,5,116,13,11,115,12,
     12,107,14,21,112,14,22,114,13,26,115,
     12,27,107
1240 GOSUB 340
1250 GOSUB 220
1260 RESTORE 1270
1270 DATA 8,12,4,97,13,5,96,13,11,96,12,12,
     97,14,21,98,14,22,99,13,26,96,12,27,97
1280 GOSUB 340
1290 CALL HCHAR(19,1,32,114)
1300 A$="IS THIS A HALF STEP?"
1310 K=19
1320 J=5
1330 GOSUB 300
1340 A$="PRESS 1 FOR YES"
1350 K=21
1360 J=9
1370 GOSUB 300
1380 A$="PRESS 2 FOR NO"
1390 K=23
1400 GOSUB 300
1410 GOSUB 1430
1420 RETURN
1430 RANDOMIZE
1440 FOR I=1 TO 10
1450 ANS=INT(RND*2+1)
1460 IF ANS=2 THEN 1530
1470 CH=INT(RND*17+1)
1480 Y=H(CH)
```

```
1490 ON CH GOSUB 520,460,520,460,400,520,46
     0,520,460,520,460,400,520,460,520,460
     ,400
1500 GOSUB 1600
1510 ON CH GOSUB 550,490,550,490,430,550,49
     0,550,490,550,490,430,550,490,550,490
     ,430
1520 GOTO 1580
1530 CH=INT(RND*16+1)
1540 Y=H(CH)
1550 ON CH GOSUB 580,660,580,840,740,580,66
     0,580,660,580,840,740,580,660,580,840
1560 GOSUB 1600
1570 ON CH GOSUB 620,700,620,890,790,620,70
     0,620,700,620,890,790,620,700,620,890
1580 NEXT I
1590 RETURN
1600 CALL SOUND(200,NN(CH),0)
1610 CALL SOUND(200,NN(CH+ANS),0)
1620 CALL KEY(0,K,S)
1630 IF (K=49)+(K=50)<>-1 THEN 1620
1640 IF K<>48+ANS THEN 1670
1650 GOSUB 1100
1660 RETURN
1670 CALL SOUND(600,-8,2)
1680 CALL SOUND(1,-8,30)
1690 RETURN
1700 PRINT "{3 SPACES}W H O L E{3 SPACES}S
     T E P S":::::::::::
1710 PRINT "A WHOLE STEP IS EQUAL TO"::"TWO
      HALF STEPS (1 = % + %)":::
1720 RESTORE 1730
1730 DATA 5,19,5,132,19,6,128,19,7,133,19,8
     ,130,19,9,134
1740 GOSUB 340
1750 GOSUB 940
1760 Y=2
1770 GOSUB 580
1780 Y=13
1790 GOSUB 660
1800 Y=21
1810 GOSUB 740
```

```
1820 Y=28
1830 GOSUB 840
1840 GOSUB 220
1850 RESTORE 1860
1860 DATA 15,14,2,96,14,3,98,14,4,99,14,5,9
     6,11,13,97,11,14,96,11,15,97,13,21,98
1870 DATA 13,22,99,12,23,96,12,24,97,12,28,
     97,12,29,96,13,30,98,13,31,99
1880 GOSUB 340
1890 CALL HCHAR(19,1,32,92)
1900 A$="WHAT KIND OF MUSICAL STEP?"
1910 K=19
1920 J=3
1930 GOSUB 300
1940 A$="PRESS 1 FOR HALF STEP"
1950 K=21
1960 J=5
1970 GOSUB 300
1980 A$="PRESS 2 FOR WHOLE STEP"
1990 K=23
2000 GOSUB 300
2010 GOSUB 1430
2020 RETURN
2030 R=0
2040 SC=0
2050 PRINT "NOW COUNT HOW MANY STEPS":"THER
     E ARE BETWEEN TWO NOTES.")::"
     {4 SPACES}1  HALF STEP"
2060 PRINT "{4 SPACES}2  WHOLE STEP":"
     {4 SPACES}3  1% STEPS":"{4 SPACES}4  2
       STEPS":"{4 SPACES}5  2% STEPS";
2070 GOSUB 940
2080 FOR II=1 TO 10
2090 GOSUB 2250
2100 NEXT II
2110 CALL HCHAR(17,1,32,256)
2120 A$="SCORE = "&STR$(SC)&"0 PERCENT"
2130 K=19
2140 J=8
2150 GOSUB 300
2160 IF SC<>10 THEN 2230
2170 RESTORE 2180
```

```
218Ø DATA 262,33Ø,392,523,392,523,33Ø,392,5
     23,659,523,659,392,523,659,784,659,78
     4,44ØØØ
219Ø FOR I=1 TO 19
22ØØ READ F
221Ø CALL SOUND(15Ø,F,2)
222Ø NEXT I
223Ø GOSUB 22Ø
224Ø RETURN
225Ø CH=INT(RND*9+1)
226Ø Y1=H(CH)
227Ø CALL SOUND(2ØØ,NN(CH),Ø)
228Ø CALL GCHAR(8,Y1,GC1)
229Ø CALL HCHAR(8,Y1,135)
23ØØ ST=INT(RND*4+1)
231Ø Y2=H(CH+ST)
232Ø CALL SOUND(2ØØ,NN(CH+ST),Ø)
233Ø CALL GCHAR(8,Y2,GC2)
234Ø CALL HCHAR(8,Y2,135)
235Ø CALL KEY(Ø,K,S)
236Ø IF (K<49)+(K>53)THEN 235Ø
237Ø IF K-48=ST THEN 254Ø
238Ø FOR I=1 TO ST
239Ø Y=H(CH+I-1)
24ØØ ON CH+I-1 GOSUB 52Ø,46Ø,52Ø,46Ø,4ØØ,52
     Ø,46Ø,52Ø,46Ø,52Ø,46Ø,4ØØ,52Ø
241Ø CALL HCHAR(16,Y,123)
242Ø CALL HCHAR(19+I-1,5,32,2)
243Ø CALL HCHAR(19+I,5,113)
244Ø CALL HCHAR(19+I,6,114)
245Ø NEXT I
246Ø GOSUB 22Ø
247Ø CALL HCHAR(16,Y1,32,8)
248Ø CALL HCHAR(19+I-1,5,32,2)
249Ø FOR I=1 TO ST
25ØØ Y=H(CH+I-1)
251Ø ON CH+I-1 GOSUB 55Ø,49Ø,55Ø,49Ø,43Ø,55
     Ø,49Ø,55Ø,49Ø,55Ø,49Ø,43Ø,55Ø
252Ø NEXT I
253Ø GOTO 256Ø
254Ø GOSUB 11ØØ
```

```
2550 SC=SC+1
2560 CALL HCHAR(8,Y1,GC1)
2570 CALL HCHAR(8,Y2,GC2)
2580 RETURN
2590 CALL CLEAR
2600 CALL CHAR(92,"3C4299A1A199423C")
2610 PRINT "{3 SPACES}M U S I C{3 SPACES}S
     T E P S"::::TAB(12);"A N D"::::TAB(9);
     "C H O R D S" ::::::::
2620 RESTORE 2630
2630 DATA 96,0,97,FFFFFFFFFFFFFFFF,98,01010
     10101010101,99,8080808080808080,37,40
     42444817214207,104,000000C0FFC
2640 DATA 123,4042444817214207,105,00300C02
     FF020C3,106,C0C0201008040201,107,3F03
     05091121408
2650 DATA 108,7C0C34C404,109,C0F00C03,112,0
     00000C0FFC,113,00000000FF,114,00300C0
     2FF020C3
2660 DATA 115,010204081020C0C,116,804021110
     905033F,117,000000000030CF0C,118,030C3
     0C,119,000000000030C30C
2670 DATA 120,00000000C0300C03,121,C0300C03
     ,122,0000004C4340C7C,128,0044447C444
     444,129,003844447C4444
2680 DATA 130,004040404407C,131,007C4078840
     404,132,0044444545428,133,007C444444
     447C,134,007C407840407C
2690 DATA 135,0044281010284 4,60,000000003F2
     0202,62,00000000FC040404,64,20202C342
     428302
2700 FOR I=1 TO 34
2710 READ A,A$
2720 CALL CHAR(A,A$)
2730 NEXT I
2740 DATA 2,131,4,139,5,147,7,156,9,165,11,
     175,13,185,14,196,16,208,17,220,19,233
2750 DATA 21,247,23,262,25,277,26,294,28,31
     1,30,330
2760 FOR I=1 TO 17
2770 READ H(I),NN(I)
2780 NEXT I
```

```
2790 NN(18)=349
2800 FOR I=9 TO 13
2810 READ A,SC
2820 CALL COLOR(I,A,SC)
2830 NEXT I
2840 DATA 2,16,9,2,9,16,9,16,9,16
2850 CALL SCREEN(12)
2860 RETURN
2870 PRINT "{5 SPACES}IDENTIFYING TRIADS"::
     :"A TRIAD CONSISTS OF 3 NOTES."::"IT I
     S IN ROOT POSITION"
2880 PRINT :"IF ALL THREE NOTES ARE ON"::"L
     INES OR IF ALL THREE"::"NOTES ARE ON
     SPACES"::"ON THE STAFF."
2890 PRINT ::"THE BASIC TRIADS ARE"::"MAJOR
     , MINOR, AUGMENTED,"::"AND DIMINISHED
     .":::
2900 GOSUB 220
2910 CALL CLEAR
2920 PRINT "TO IDENTIFY A CHORD,"::"FIRST M
     AKE SURE THE NOTES"::"ARE IN ROOT POS
     ITION"
2930 PRINT :"(INVERT IF NECESSARY).":::"THE
     NAME OF THE CHORD IS"::"THE BOTTOM N
     OTE OF THE"::"ROOT CHORD.":::
2940 GOSUB 220
2950 CALL CLEAR
2960 PRINT "THE TYPE OF TRIAD MAY BE"::"DET
     ERMINED BY COUNTING"::"STEPS BETWEEN
     NOTES OF"::"THE ROOT CHORD.":::
2970 GOSUB 220
2980 CALL CLEAR
2990 PRINT "{3 SPACES}M A J O R{3 SPACES}T
     R I A D"::"THE C MAJOR TRIAD CONSISTS
      OF C, E, AND G."
3000 PRINT :"2  STEPS FROM C TO E":"1% STEP
     S FROM E TO G"
3010 GOSUB 940
3020 CALL SOUND(200,262,0)
3030 CALL HCHAR(14,2,135)
3040 CALL SOUND(200,262,2,330,0)
3050 CALL HCHAR(14,8,135)
```

```
3060 CALL SOUND(600,262,2,330,1,392,0)
3070 CALL HCHAR(14,14,135)
3080 A$="<--2--><-1%->"
3090 J=1
3100 K=2
3110 CALL HCHAR(1,1,32,96)
3120 GOSUB 300
3130 GOSUB 220
3140 CALL HCHAR(19,1,32,160)
3150 CALL HCHAR(17,8,73)
3160 CALL HCHAR(17,10,78)
3170 A$="TO CHANGE TO A MINOR TRIAD"
3180 K=19
3190 J=2
3200 GOSUB 300
3210 A$="LOWER THE MIDDLE NOTE % STEP."
3220 K=20
3230 GOSUB 300
3240 A$="<-1%-><--2-->"
3250 J=1
3260 K=2
3270 GOSUB 300
3280 CALL SOUND(200,262,0)
3290 CALL HCHAR(14,8,96)
3300 CALL SOUND(200,262,2,311,0)
3310 CALL HCHAR(11,7,135)
3320 CALL SOUND(600,262,2,311,1,392,0)
3330 A$="1% STEPS FROM C TO E-FLAT"
3340 J=2
3350 K=22
3360 GOSUB 300
3370 A$="2  STEPS FROM E-FLAT TO G"
3380 K=23
3390 GOSUB 300
3400 GOSUB 220
3410 CALL CLEAR
3420 PRINT "{6 SPACES}AUGMENTED TRIAD":::"S
     TART WITH THE MAJOR TRIAD.":::"'AUGMEN
     T' THE TRIAD"
3430 PRINT :"BY MOVING THE TOP NOTE"::"UP O
     NE HALF STEP.":::
3440 GOSUB 220
```

```
3450 CALL CLEAR
3460 PRINT "{6 SPACES}AUGMENTED TRIAD"::"NO
     TES ARE C, E, G#"
3470 PRINT :"2  STEPS FROM C TO E":"2  STEP
     S FROM E TO G#"::
3480 GOSUB 940
3490 CALL SOUND(200,262,0)
3500 CALL HCHAR(14,2,135)
3510 CALL SOUND(200,262,2,330,0)
3520 CALL HCHAR(14,8,135)
3530 CALL SOUND(600,262,2,330,1,415,0)
3540 CALL HCHAR(11,16,135)
3550 A$="<--2--><--2-->"
3560 K=2
3570 J=1
3580 CALL HCHAR(1,1,32,96)
3590 GOSUB 300
3600 GOSUB 220
3610 CALL CLEAR
3620 PRINT "{5 SPACES}DIMINISHED TRIAD":::"
     START WITH THE MINOR TRIAD."
3630 PRINT ::"'DIMINISH' THE TRIAD BY"::"LO
     WERING THE TOP NOTE"::"ONE HALF STEP.
     ":::::
3640 GOSUB 220
3650 CALL CLEAR
3660 PRINT "{5 SPACES}DIMINISHED TRIAD"::"N
     OTES ARE C, E@, G@"::"1% STEPS FROM C
     TO E @":"1% STEPS FROM E@ TO G@"
3670 GOSUB 940
3680 CALL SOUND(200,262,0)
3690 CALL HCHAR(14,2,135)
3700 CALL SOUND(200,262,2,311,0)
3710 CALL HCHAR(11,7,135)
3720 CALL SOUND(600,262,2,311,1,370,0)
3730 CALL HCHAR(11,13,135)
3740 A$="<-1%-><-1%->"
3750 K=2
3760 J=1
3770 CALL HCHAR(1,1,32,96)
3780 GOSUB 300
3790 GOSUB 220
```

```
3800 CALL CLEAR
3810 PRINT TAB(9);"STEPS BETWEEN NOTES":TAB
     (9);"1ST{5 SPACES}2ND{5 SPACES}3RD"
3820 PRINT ::"TRIAD"::::"MAJOR";TAB(14);"2
     {7 SPACES}1%"::::"MINOR";TAB(14);"1%
     {6 SPACES} 2"
3830 PRINT ::"AUGMENTED{4 SPACES}2
     {7 SPACES}2"::::"DIMINISHED{3 SPACES}1%
     {6 SPACES}1%"::::
3840 GOSUB 220
3850 PRINT "NAME THE TYPE OF TRIAD."::"PRES
     S{3 SPACES}1  MAJOR":TAB(9);"2  MINOR"
3860 PRINT TAB(9);"3  AUGMENTED":TAB(9);"4
     DIMINISHED"
3870 GOSUB 940
3880 SC=0
3890 FOR II=1 TO 10
3900 CH=INT(RND*9+1)
3910 CALL SOUND(500,NN(CH),0)
3920 Y1=H(CH)
3930 CALL GCHAR(8,Y1,GC1)
3940 CALL HCHAR(8,Y1,135)
3950 ST=INT(RND*2+1)+2
3960 CALL SOUND(-500,NN(CH+ST),0)
3970 Y2=H(CH+ST)
3980 CALL GCHAR(8,Y2,GC2)
3990 CALL HCHAR(8,Y2,135)
4000 ST2=INT(RND*2+1)+2
4010 CALL SOUND(-500,NN(CH+ST+ST2),0)
4020 Y3=H(CH+ST+ST2)
4030 CALL GCHAR(8,Y3,GC3)
4040 CALL HCHAR(8,Y3,135)
4050 IF (ST=4)+(ST2=3)<>-2 THEN 4080
4060 ANS=1
4070 GOTO 4150
4080 IF (ST=3)+(ST2=4)<>-2 THEN 4110
4090 ANS=2
4100 GOTO 4150
4110 IF (ST=4)+(ST2=4)<>-2 THEN 4140
4120 ANS=3
4130 GOTO 4150
4140 ANS=4
```

```
4150 CALL SOUND(1000,NN(CH),2,NN(CH+ST),1,N
     N(CH+ST+ST2),0)
4160 CALL KEY(0,K,S)
4170 IF (K<49)+(K>52)THEN 4160
4180 CALL HCHAR(19+K-48,10,42)
4190 IF K-48=ANS THEN 4390
4200 CALL SOUND(500,-5,2)
4210 IF ST/2=2 THEN 4240
4220 S1$="1%"
4230 GOTO 4250
4240 S1$="-2-"
4250 IF ST2/2=2 THEN 4280
4260 S2$="1%"
4270 GOTO 4290
4280 S2$="-2-"
4290 A$="<-"&S1$&"-><-"&S2$&"->"
4300 K=2
4310 J=Y1-1
4320 CALL HCHAR(1,1,32,96)
4330 GOSUB 300
4340 CALL HCHAR(19+ANS,9,114)
4350 GOSUB 220
4360 CALL HCHAR(19+ANS,9,32)
4370 CALL HCHAR(2,1,32,32)
4380 GOTO 4440
4390 CALL SOUND(150,NN(CH),2)
4400 CALL SOUND(150,NN(CH+ST),2)
4410 CALL SOUND(150,NN(CH+ST+ST2),2)
4420 CALL SOUND(200,2*NN(CH),2)
4430 SC=SC+1
4440 CALL HCHAR(8,Y1,GC1)
4450 CALL HCHAR(8,Y2,GC2)
4460 CALL HCHAR(8,Y3,GC3)
4470 CALL VCHAR(20,10,32,4)
4480 NEXT II
4490 GOSUB 2110
4500 RETURN
4510 END
```

Choreography

Coordinating computer graphics with music is an art, very much like the art of stage choreography, which combines dance movements with music. While a tone is being played after a CALL SOUND statement, the computer goes on to execute other statements — calculations, character or color definitions, or graphics. Because of this feature, it is fun to make your TI show pictures while a song is being played — with certain drawings appearing at certain precise times.

Keeping Time

A new CALL SOUND statement will always wait for the previous sound to finish before it starts playing. Depending on the duration of each sound, you can insert a number of graphics statements between the execution of the CALL SOUND statements. Choreography takes a lot of experimentation so that you avoid jerky sounds (irregular silences between two tones) while the graphics keeps up with the music.

Here is a situation where it is a definite advantage to designate a duration variable at the beginning of the program and then use that variable in the CALL SOUND statements. This way, if you need to increase the duration slightly, you will need to change only one statement, not each CALL SOUND statement. (See "Musical Tempo Demonstration," Program 3-4.)

Be careful in your use of FOR-NEXT loops. Remember that each statement in the loop is performed again, and you will be able to hear CALL SOUND statements if they are repeated. That may be fine for a chorus — but do you also want the graphics repeated? If a FOR-NEXT loop for graphics is used between CALL SOUND statements, the duration of the first statement must be long enough for the whole loop to finish, or you may have an unwanted break in the music.

Watch Your Memory

Running out of memory may be a problem with graphics programs because defining characters consumes a lot of memory. Try to plan your color sets wisely, and use as few special characters as possible. Use CALL HCHAR and CALL VCHAR efficiently to draw with as few statements as possible.

DATA statements may be used to preserve memory if the same process or number sequence is repeated. A note of warning: sometimes there is a pause before the last iteration of (or single pass through) a FOR-NEXT loop that reads data (especially in longer programs). You can often avoid the problem by adding one more set of "dummy data" and increasing the loop limit by one. For example, if you are drawing graphics characters, draw a space where it won't be noticed, or repeat the last character:

```
100 CALL CLEAR
110 FOR I=1 TO 10
120 READ X,Y,G
130 CALL HCHAR(X,Y,G)
140 NEXT I
150 DATA 12,13,42,12,18,42,20,14,43,18,10,
    65,14,12,66
160 DATA 10,8,77,19,15,64,8,18,80,7,10,43,
    1,1,32
170 END
```

With CALL SOUND statements, it is more difficult to avoid the pause. You may add another set of data with a duration of one millisecond, a high frequency that cannot be heard, and a volume of 30 to avoid the pause before the last sound in the loop, but there will be a pause between the loop and the next CALL SOUND statement.

In general, it is easier simply to alternate CALL SOUND and graphics statements than to use DATA statements and FOR-NEXT loops.

Songs with Pictures

In "Oh! Susanna," several graphics characters are defined first (lines 110-160) and then the screen is cleared. The tempo is set with T=400, then CALL SOUND statements are alternated with graphics statements so that certain pictures appear as the appropriate words are sung in the familiar song. Some objects are drawn invisibly by assigning the color set a transparent foreground and background. Then, at the appropriate time, another CALL COLOR statement colors the object the right color.

Program 3-7. "Oh! Susanna"

```
100 REM  OH! SUSANNA
110 FOR C=104 TO 110
120 READ C$
130 CALL CHAR(C,C$)
140 NEXT C
150 DATA FFFFFFFFFFFFFFFF,00030F1F3F3F7F7F,
    03C7FEECD4AC5EAE,FDFAF5EBD7BF5FBF,7D7
    A3D3F1F0F03
160 DATA FEFEFCFCF8F0C,03070E1C3870E0C
170 CALL CLEAR
180 T=400
190 CALL CHAR(96,"FFFFFFFFFFFFFFFF")
200 CALL SOUND(T/2,392,2)
210 CALL CHAR(97,"FFFFFFFFFFFFE7C1")
220 CALL SOUND(T/2,440,2)
230 CALL CHAR(98,"808080800C0C0C0C")
240 CALL SOUND(T,494,2,392,5,294,8)
250 CALL CHAR(99,"C0C0C0E0E0E0E0E")
260 CALL SOUND(T,587,1,392,5,247,7)
270 CALL CHAR(100,"F0F0F0F0F0F0F0F")
280 CALL COLOR(9,3,1)
290 CALL SOUND(T,587,2,392,6,196,7)
300 CALL VCHAR(7,5,96,5)
310 CALL VCHAR(12,5,97)
320 CALL VCHAR(12,6,98)
330 CALL SOUND(T,659,2,392,5,262,8)
340 CALL VCHAR(7,6,96,5)
350 CALL VCHAR(7,7,96,5)
360 CALL SOUND(T,587,2,392,5,247,7)
370 CALL VCHAR(9,8,98)
380 CALL VCHAR(10,8,99)
390 CALL SOUND(T,494,2,392,5,294,7)
400 CALL VCHAR(11,8,100)
410 CALL COLOR(10,1,1)
420 CALL SOUND(T*1.5,392,2,247,5,196,6)
430 CALL VCHAR(9,12,104,3)
440 CALL HCHAR(10,11,104,3)
450 CALL HCHAR(9,11,105)
460 CALL HCHAR(9,13,106)
470 CALL SOUND(T/2,440,2,262,5)
```

```
480 CALL HCHAR(10,12,107)
490 CALL HCHAR(11,11,108)
500 CALL SOUND(T,494,2,392,5,294,6)
510 CALL HCHAR(11,13,109)
520 CALL HCHAR(8,14,110)
530 CALL HCHAR(7,15,110)
540 CALL SOUND(T,494,1,392,6)
550 CALL COLOR(10,15,1)
560 CALL HCHAR(6,16,110)
570 CALL SOUND(T,440,2,262,5,196,8)
580 CALL COLOR(11,1,1)
590 CALL CHAR(112,"FFFFFFFFFFFFFFFF")
600 CALL SOUND(T,392,2,247,5,196,8)
610 CALL CHAR(113,"7F3F1F1F1F1F1F1F")
620 CALL CHAR(114,"1F1F1F1F1F1F1F1F")
630 CALL SOUND(3*T,440,2,349,5,147,8)
640 CALL CHAR(115,"1F1F3F3F7F78E0C")
650 CALL CHAR(116,"FFFFFFFFFF1E")
660 CALL CHAR(117,"F0F0F8F8FCFCFC1C")
670 CALL CHAR(118,"E0E0E0E0F0F0F0F")
680 CALL CHAR(119,"E0E0E0C0C0808")
690 CALL SOUND(T/2,392,2)
700 CALL HCHAR(14,15,112,3)
710 CALL SOUND(T/2,440,2)
720 CALL HCHAR(15,15,112,3)
730 CALL SOUND(T,494,2,392,6,294,8)
740 CALL VCHAR(14,18,118,2)
750 CALL HCHAR(16,15,113)
760 CALL SOUND(T,587,2,392,6,247,8)
770 CALL HCHAR(17,15,114)
780 CALL HCHAR(18,15,115)
790 CALL SOUND(1.5*T,587,1,392,5,247,9)
800 CALL HCHAR(16,16,112,2)
810 CALL HCHAR(17,16,112,4)
820 CALL HCHAR(18,16,116,4)
830 CALL SOUND(T/2,659,2,392,5,262,8)
840 CALL COLOR(11,14,1)
850 CALL SOUND(T,587,2,392,5,247,7)
860 CALL VCHAR(14,18,118)
870 CALL VCHAR(16,18,119)
880 CALL SOUND(T,494,2,392,6,294,8)
890 CALL VCHAR(17,20,118)
```

```
900 CALL VCHAR(18,20,117)
910 CALL SOUND(1.5*T,392,2,247,5)
920 CALL CHAR(120,"101038387C7C7C38")
930 CALL COLOR(12,8,1)
940 CALL SOUND(T/2,440,2)
950 CALL SOUND(T,494,2,392,6,294,8)
960 CALL CHAR(128,"FFFFFFFFFFFFFFFF")
970 CALL SOUND(T,494,1,392,5,147,8)
980 CALL CHAR(129,"00030F1F3F3F7F7F")
990 CALL SOUND(T,440,2,370,5,262,8)
1000 CALL CHAR(130,"00C0F0F8FCFCFEFE")
1010 CALL SOUND(T,440,1,370,6,147,8)
1020 CALL CHAR(131,"7F7F3F3F1F0F03")
1030 CALL SOUND(3*T,392,1,247,5,196,8)
1040 CALL CHAR(132,"FEFEFCFCF8F0C")
1050 CALL COLOR(13,1,1)
1060 CALL HCHAR(6,27,128,3)
1070 CALL VCHAR(5,28,128,3)
1080 CALL SOUND(T/2,392,2)
1090 CALL SOUND(T/2,440,2)
1100 CALL SOUND(T,494,2,392,6,294,8)
1110 CALL HCHAR(6,21,120)
1120 CALL HCHAR(8,20,120)
1130 CALL HCHAR(7,22,120)
1140 CALL SOUND(T,587,2,392,5,247,8)
1150 CALL HCHAR(9,22,120)
1160 CALL HCHAR(10,21,120)
1170 CALL SOUND(T,587,1,392,6,196,8)
1180 CALL HCHAR(5,27,129)
1190 CALL SOUND(T,659,2,392,5,262,8)
1200 CALL HCHAR(5,29,130)
1210 CALL SOUND(T,587,2,392,5,247,8)
1220 CALL HCHAR(7,27,131)
1230 CALL SOUND(T,494,2,392,5,294,8)
1240 CALL HCHAR(7,29,132)
1250 CALL SOUND(1.5*T,392,2,247,5,196,8)
1260 CALL SOUND(T/2,440,2,262,4)
1270 CALL SOUND(T,494,2,392,5,294,8)
1280 CALL SOUND(T,494,1,392,4,196,8)
1290 CALL SOUND(T,440,2,277,5,165,8)
1300 CALL SOUND(T,392,2,330,5,262,8)
1310 CALL SOUND(3*T,440,2,370,5,294,8)
```

```
1320 CALL SOUND(T/2,392,2)
1330 CALL SOUND(T/2,440,2)
1340 CALL SOUND(T,494,2,392,5,294,8)
1350 CALL COLOR(13,12,1)
1360 CALL SOUND(T,587,2,392,5,247,8)
1370 CALL SOUND(T,587,0,392,6,196,8)
1380 CALL SOUND(T,659,2,392,5,262,8)
1390 CALL SOUND(T,587,2,392,5,196,8)
1400 CALL SOUND(T,494,2,392,5,294,8)
1410 CALL SOUND(2*T,392,1,247,4,196,8)
1420 CALL SOUND(T/2,440,2,131,8)
1430 CALL SOUND(T/2,494,2,392,5)
1440 CALL SOUND(1.5*T,494,1,392,4,147,8)
1450 CALL SOUND(T,440,2,370,5,262,8)
1460 CALL SOUND(T,440,1,370,4,147,8)
1470 CALL SOUND(4*T,392,0,247,5,196,8)
1480 CALL SOUND(2*T,523,0,330,5,262,8)
1490 PRINT "OH!";
1500 CALL SOUND(2*T,523,1,330,4,262,7)
1510 PRINT "  SU";
1520 CALL SOUND(T,659,0,392,5,262,8)
1530 PRINT "SAN";
1540 CALL SOUND(2*T,659,1,392,4,131,7)
1550 PRINT "NA";
1560 CALL SOUND(T,587,1,392,5,247,7)
1570 CALL SOUND(T,587,0,392,4,196,7)
1580 CALL SOUND(T,494,0,392,5,294,8)
1590 CALL SOUND(T,392,1,247,5,196,8)
1600 CALL SOUND(3*T,440,1,370,5,294,8)
1610 CALL SOUND(T/2,392,2)
1620 CALL SOUND(T/2,440,2)
1630 CALL SOUND(T,494,2,392,7,294,9)
1640 CALL SOUND(T,587,2,392,5,294,9)
1650 CALL SOUND(T,587,1,392,6,196,8)
1660 CALL SOUND(T,659,1,392,5,262,8)
1670 CALL COLOR(11,9,1)
1680 CALL SOUND(T,587,2,392,5,247,9)
1690 CALL SOUND(T,494,2,392,5,294,8)
1700 CALL SOUND(1.5*T,392,2,247,5,196,8)
1710 CALL SOUND(T/2,440,2,131,8)
1720 CALL SOUND(T,494,1,392,5,147,8)
1730 CALL COLOR(10,16,1)
```

```
1740 CALL SOUND(T,494,0,392,4,294,8)
1750 CALL COLOR(10,15,1)
1760 CALL SOUND(T,440,0,370,5,147,8)
1770 CALL SOUND(T,440,1,370,4,262,8)
1780 CALL SOUND(4*T,392,0,247,5,196,8)
1790 GOTO 1790
1800 END
```

"Hey, Diddle, Diddle" is such a short song that the graphics need to come on quickly. So, all the characters are defined at the start using DATA statements (lines 120-310). All the color sets are set to invisible (lines 320-340).

PRINT statements are faster than CALL HCHAR or CALL VCHAR, so some of the figures are PRINTed on the screen. You'll notice the PRINT statements (lines 380-480) print symbols or lowercase letters. Those characters have actually been redefined graphically, so that when they are printed on the screen they are really pictures rather than symbols and numbers. Again, the pictures are drawn invisibly. Then when the music comes to the appropriate word, the associated picture is turned on with a CALL COLOR statement.

Program 3-8. "Hey, Diddle, Diddle"

```
100 REM  HEY, DIDDLE, DIDDLE
110 CALL CLEAR
120 FOR I=1 TO 6
130 READ A,B
140 FOR C=A TO B
150 READ C$
160 CALL CHAR(C,C$)
170 NEXT C
180 NEXT I
190 DATA 40,66,FFFFFFFFFFFFFFFF,0000010D1F0
    F070F,00008080E0F0F838,0000000021331E
    0C,010F3F7FC7870707
200 DATA FFFFFFFFFFFEFEFC,38FCFE6E0C,070707
    0F1F3F7FFE,FFFFFFFFFFFEFCF8,FFFFFFFF7
    F1F,FFFFFFFFFF8703
210 DATA F8F0F0F0F8FCFEFF,76C707070E0C08,7F
    0706,80E06,,000000010307 0C7C,000030F0
    F0F8F8FC,0000806030383838
```

```
220 DATA 7FØFØØØØ1Ø73F3F,FCFFFFFFFFFFFFBF,
    ØØFFFFFFFFFFFFFFFF,FØEØFØF8F8F8F8F8,3
    F,ØFØ1,FEFC,F878F8F8
230 DATA 96,1Ø7,ØØØØØØØ7Ø797D7F,ØØØØØØØ7C
    FEFFFF,ØØØØØØØ1C7CFCF8,7F3F7FFFFFFBF
    1F1,FFFFFFFFFFFBF1F1
240 DATA FØEØFØFØFØFØFØF,ØØ1C3E3E3E3F3F1F,F
    1FF7FØF1FFFFFFF,B11FFF1EFFFFFFFF,E7CF
    9F3FFFFEFCF8
250 DATA ØFØ3Ø8ØCØC1E1F1F,1FØFØ7,112,131,FF
    FF1F1F3F7F7FFF,FFFFFFFFFFFFFFFF,EØ8ØØ
    ØØØØØ8ØEØF
260 DATA FØF8F8F8F8F8F8F8,FFFFFFFFFFFFF1F,DF
    9FØFØFØ7Ø7Ø1,F8F8FØFØFØFØC,,1F3F7FFFF
    FFFFFFF,ØØCØEØFØF8FCF8F8
270 DATA FF7F7F3F3F18Ø8,F8FCFFFFFFFFFFFF,4Ø
    4ØEØFØF8F8FCFE,FFFFFFFF7F3F1FØF,FFFFF
    FFFFFFFFFFF,ØFØ7Ø3
280 DATA F7E3C1,8ØCØEØFØ783C1CØF,Ø7Ø3Ø7Ø7Ø7
    Ø7Ø3,CØEØEØEØCØ8,136,14Ø,FFFFFFFFFFFF
    FFFF,ØØØ3ØF1F3F3F7F7F
290 DATA ØØCØFØF8FCFCFEFE,7F7F3F3F1FØFØ3,FE
    FEFCFCF8FØC,144,148,FFFFFFFFFFFFFFFF,
    ØØØ3ØF1F3F3F7F7F
300 DATA ØØCØFØF8FCFCFEFE,7F7F3F3F1FØFØ3,FE
    FEFCFCF8FØC,152,155,183C7EFFFFFFFFFF,
    FF7E3C1818181818
310 DATA 1818181818181818,1Ø2828484484 84C3
320 FOR I=2 TO 16
330 CALL COLOR(I,1,1)
340 NEXT I
350 T=35Ø
360 CALL SCREEN(8)
370 CALL SOUND(T,44Ø,2,175,8)
380 PRINT TAB(15);")*":TAB(1Ø);"+,(((-."
390 CALL SOUND(T,44Ø,1,131,9)
400 PRINT TAB(11);"/Ø123":TAB(11);"4
    {3 SPACES}56"::
410 CALL SOUND(T,44Ø,2)
420 PRINT ::::::
430 CALL SOUND(T,44Ø,2,22Ø,5,175,8)
```

```
440 PRINT " `ab":" cde xy{7 SPACES}89 :"
450 CALL SOUND(T,494,2,175,8)
460 PRINT "fghi z{¦{6 SPACES};<==>":"jpqr
      }~{6 SPACES}?@AB
470 CALL SOUND(T,523,2)
480 PRINT "kqqs":" tuv":::
490 CALL SOUND(T,392,2,131,8)
500 CALL COLOR(9,15,1)
510 CALL COLOR(10,15,1)
520 CALL COLOR(11,6,1)
530 CALL SOUND(T,392,1,196,8)
540 CALL HCHAR(20,9,127)
550 CALL HCHAR(20,10,128)
560 CALL SOUND(T,392,2)
570 CALL HCHAR(20,11,129)
580 CALL HCHAR(21,11,130)
590 CALL SOUND(T,392,0,233,5,165,8)
600 CALL COLOR(12,10,1)
610 CALL COLOR(13,10,1)
620 CALL SOUND(T,349,2,233,5,165,8)
630 CALL HCHAR(21,12,131)
640 CALL SOUND(T,392,2)
650 CALL SOUND(2*T,440,2,175,8)
660 CALL COLOR(2,14,1)
670 CALL COLOR(3,14,1)
680 CALL HCHAR(10,14,136,3)
690 CALL VCHAR(9,15,136,3)
700 CALL SOUND(T,440,1)
710 CALL HCHAR(9,14,137)
720 CALL HCHAR(9,16,138)
730 CALL SOUND(T,440,2,220,8,175,9)
740 CALL HCHAR(11,14,139)
750 CALL SOUND(T,466,2,220,8,175,9)
760 CALL SOUND(T,523,2)
770 CALL HCHAR(11,16,140)
780 CALL SOUND(5*T,392,2,131,8)
790 CALL COLOR(14,16,1)
800 CALL SOUND(T,440,2)
810 CALL SOUND(T,466,2,117,8)
820 CALL SOUND(T,466,1,233,8)
830 CALL SOUND(T,466,0)
840 CALL COLOR(4,3,1)
```

```
850 CALL COLOR(5,3,1)
860 CALL SOUND(T,466,1,175,8,294,6)
870 CALL HCHAR(20,25,144,3)
880 CALL SOUND(T,523,1,175,8,294,6)
890 CALL VCHAR(19,26,144,3)
900 CALL SOUND(T,587,1,175,8,294,6)
910 CALL HCHAR(19,25,145)
920 CALL SOUND(T*2,523,2,110,8)
930 CALL HCHAR(19,27,146)
940 CALL HCHAR(21,25,147)
950 CALL HCHAR(21,27,148)
960 CALL SOUND(T,440,2)
970 CALL SOUND(T,349,2,220,6,147,8)
980 CALL HCHAR(22,26,155)
990 CALL SOUND(T,392,2,220,6,147,8)
1000 CALL SOUND(T,440,2)
1010 CALL SOUND(2*T,262,2,233,6,165,8)
1020 CALL COLOR(15,12,1)
1030 CALL COLOR(16,11,1)
1040 CALL SOUND(T,262,1)
1050 CALL SOUND(T,262,2,233,6,165,8)
1060 CALL SOUND(T,294,2,233,6,165,8)
1070 CALL SOUND(T,330,2)
1080 CALL SOUND(6*T,349,1,220,8,131,9)
1090 CALL HCHAR(18,29,152)
1100 CALL HCHAR(19,29,153)
1110 CALL VCHAR(20,29,154,2)
1120 CALL VCHAR(22,29,155)
1130 GOTO 1130
1140 END
```

"We Wish You a Merry Christmas" is an electronic Christmas card — a computerized Christmas message with graphics and music. First, the picture was drawn on 24-by-32 graph paper. (See Figure 3-4.) The star and the edges of the Christmas tree are redefined graphics characters. The border is made up of a repeated graphics character which is red and green. At the end of the song, red foreground and green background will alternate with green foreground and red background.

To use larger letters for the word "COMPUTE!," characters needed to be defined. To create the pieces of these large letters, the more detailed graph paper with squares divided up into 8 by 8 grids can be used. The word "COMPUTE!" was traced from a magazine cover, then the tracing was approximated with filled-in squares. (See Figure 3-5.)

Since it would take quite a while to define the necessary twenty characters and place them graphically on the screen, the characters were assigned to the first twenty lowercase letters. These characters are defined in DATA statements and a READ routine in lines 120 to 190 at the beginning of the program.

The graphic letters are put on the screen very quickly with a PRINT statement (line 210), first PRINTing "abcdefghij," then, directly under those characters, PRINTing "klmnopqrst." As music is played, "COMPUTE!" scrolls up the screen as the program PRINTS blank lines below it.

All other graphics characters and colors are defined and drawn on the screen between CALL SOUND statements. At the end of the piece, the star, the lights on the tree, and the border all blink as CALL COLOR statements and a GOTO statement repeat the color definitions.

Program 3-9. "We Wish You A Merry Christmas"

```
100 REM  XMAS
110 CALL CLEAR
120 FOR C=97 TO 116
130 READ C$
140 CALL CHAR(C,C$)
150 NEXT C
160 DATA 00001F3F7FF0E0E,000001C3E7EFEE0E,0
    000F0F8FC1E0E0E,0000F0F0F8F8FDFD,0000
    7B7BFBFBFBFB
170 DATA 0000F1F9FD9D9DFD,0000C7C7C7C7C7C7,
    00007F7F7F0E0E0E,0000DFDFDF1C1F1F,000
    0DCDCDC1CDCDC
180 DATA E0E0F07F3F1F,0EEEFE7C301,0E0E1EFC
    F8F,EFEFEFE7E7E7,BBBBBB3B3B3B,F9F1818
    1808,C7C7EFFFFE7C
190 DATA 0E0E0E0E0E0E,1F1C1C1F1F1F,DC1C00DC
    DCDC
200 T=500
```

```
210 PRINT "abcdefghij":"klmnopqrst":::::
220 CALL SOUND(T,330,2)
230 PRINT :::
240 CALL SOUND(T,440,1,277,7,110,9)
250 PRINT ::::
260 CALL SOUND(T/2,440,1,277,7,165,9)
270 PRINT ::
280 CALL SOUND(T/2,494,1,277,7,165,9)
290 PRINT ::
300 CALL SOUND(T/2,440,1,277,7,220,9)
310 PRINT :TAB(7);"MER";
320 CALL SOUND(T/2,415,1,277,7,220,9)
330 PRINT "RY ";
340 CALL SOUND(T,370,1,294,7,147,9)
350 PRINT "CHRIST";
360 CALL SOUND(T,370,1,294,7,185,9)
370 PRINT "MAS":::
380 CALL SOUND(T,370,1,294,7,220,9)
390 CALL CHAR(128,"FFCDB7DD9D63ADFF")
400 CALL SOUND(T,494,1,294,7,123,9)
410 CALL COLOR(13,9,3)
420 CALL SOUND(T/2,494,1,294,7,185,9)
430 CALL HCHAR(1,1,128,32)
440 CALL SOUND(T/2,554,1,294,7,185,9)
450 CALL VCHAR(2,1,128,23)
460 CALL SOUND(T/2,494,1,294,7,247,9)
470 CALL VCHAR(2,32,128,23)
480 CALL SOUND(T/2,440,1,294,7,247,9)
490 CALL HCHAR(24,2,128,30)
500 CALL SOUND(T,415,1,330,7,165,9)
510 CALL CHAR(136,"081818FF7C3C6683")
520 CALL COLOR(14,12,1)
530 CALL SOUND(T,330,1,208,9)
540 CALL CHAR(48,"1010383838 7C7CFE")
550 CALL SOUND(T,330,1,165,9)
560 CALL COLOR(3,3,1)
570 CALL CHAR(49,"000103070F1F3FFF")
580 CALL SOUND(T,554,0,330,6,110,9)
590 CALL COLOR(2,16,3)
600 CALL CHAR(50,"FFFFFFFFFFFFFFFF")
610 CALL SOUND(T/2,554,0,330,6,165,9)
620 CALL CHAR(51,"0080C0E0F0F8FCFF")
```

```
630 CALL SOUND(T/2,587,Ø,33Ø,6,165,9)
64Ø CALL CHAR(52,"ØFØF1F1F3F3F7FFF")
65Ø CALL SOUND(T/2,554,Ø,33Ø,6,22Ø,9)
66Ø CALL CHAR(53,"FØFØF8F8FCFCFEFF")
67Ø CALL SOUND(T/2,494,Ø,33Ø,6,22Ø,9)
68Ø CALL CHAR(54,"FFFFFFFFFFFEF8C")
69Ø CALL SOUND(T,44Ø,Ø,294,6,147,8)
7ØØ CALL CHAR(55,"FFFFFFFFFF7F1FØ3")
71Ø CALL HCHAR(2,16,136)
72Ø CALL SOUND(T,37Ø,Ø,294,6,22Ø,8)
73Ø CALL HCHAR(3,16,48)
74Ø CALL HCHAR(4,15,49)
75Ø CALL HCHAR(4,16,42)
76Ø CALL HCHAR(4,17,51)
77Ø CALL SOUND(T/2,33Ø,1,22Ø,6,139,9)
78Ø CALL HCHAR(5,15,52)
79Ø CALL HCHAR(5,16,5Ø)
8ØØ CALL HCHAR(5,17,53)
81Ø CALL SOUND(T/2,33Ø,1,22Ø,6)
82Ø CALL HCHAR(6,14,49)
83Ø CALL HCHAR(6,15,42,3)
84Ø CALL HCHAR(6,16,5Ø)
85Ø CALL HCHAR(6,18,51)
86Ø CALL SOUND(T,37Ø,Ø,294,6,147,8)
87Ø CALL HCHAR(7,14,52)
88Ø CALL HCHAR(7,15,5Ø,3)
89Ø CALL HCHAR(7,18,53)
9ØØ CALL HCHAR(8,13,49)
91Ø CALL SOUND(T,494,Ø,294,6,185,8)
92Ø CALL HCHAR(8,14,5Ø,5)
93Ø CALL HCHAR(8,19,51)
94Ø CALL HCHAR(8,16,42)
95Ø CALL SOUND(T,415,1,33Ø,7,247,9)
96Ø CALL HCHAR(9,13,52)
97Ø CALL HCHAR(9,14,42,5)
98Ø CALL HCHAR(9,15,5Ø,3)
99Ø CALL HCHAR(9,19,53)
1ØØØ CALL SOUND(T*2,44Ø,Ø,277,5,11Ø,8)
1Ø1Ø CALL HCHAR(1Ø,12,49)
1Ø2Ø CALL HCHAR(1Ø,13,5Ø,7)
1Ø3Ø CALL HCHAR(1Ø,2Ø,51)
1Ø4Ø CALL HCHAR(11,12,52)
```

```
1050 CALL HCHAR(11,13,50,7)
1060 CALL HCHAR(11,20,53)
1070 CALL HCHAR(11,17,42)
1080 CALL HCHAR(12,11,49)
1090 CALL HCHAR(12,12,50,9)
1100 CALL HCHAR(12,21,51)
1110 CALL SOUND(T,330,2)
1120 CALL HCHAR(12,14,42)
1130 CALL HCHAR(13,11,52)
1140 CALL HCHAR(13,12,50,9)
1150 CALL HCHAR(13,21,53)
1160 CALL SOUND(T,440,2,277,8,110,10)
1170 CALL HCHAR(14,10,49)
1180 CALL HCHAR(14,11,50,11)
1190 CALL HCHAR(14,22,51)
1200 CALL HCHAR(13,12,42)
1210 CALL SOUND(T/2,440,2,277,8,165,10)
1220 CALL HCHAR(15,10,52)
1230 CALL HCHAR(15,11,50,11)
1240 CALL HCHAR(15,22,53)
1250 CALL SOUND(T/2,494,2,277,8,165,10)
1260 CALL HCHAR(16,9,49)
1270 CALL HCHAR(16,10,50,13)
1280 CALL HCHAR(16,23,51)
1290 CALL SOUND(T/2,440,2,277,8,220,10)
1300 CALL HCHAR(17,9,52)
1310 CALL HCHAR(17,10,50,13)
1320 CALL HCHAR(17,23,53)
1330 CALL SOUND(T/2,415,2,277,8,220,10)
1340 CALL HCHAR(18,8,49)
1350 CALL HCHAR(18,9,54)
1360 CALL HCHAR(18,10,50,14)
1370 CALL SOUND(T,370,2,294,8,147,10)
1380 CALL HCHAR(18,24,51)
1390 CALL HCHAR(18,11,54)
1400 CALL HCHAR(18,13,54)
1410 CALL HCHAR(18,15,54)
1420 CALL HCHAR(18,17,55)
1430 CALL SOUND(T,370,2,294,8,185,10)
1440 CALL HCHAR(18,19,55)
1450 CALL HCHAR(18,21,55)
1460 CALL HCHAR(18,23,55)
```

```
1470 CALL COLOR(15,13,13)
1480 CALL SOUND(T,370,2,294,8,220,10)
1490 CALL HCHAR(19,15,144,3)
1500 CALL HCHAR(20,15,144,3)
1510 CALL SOUND(T,494,1,294,7,123,9)
1520 CALL HCHAR(13,16,42)
1530 CALL HCHAR(13,19,42)
1540 CALL HCHAR(15,11,42)
1550 CALL HCHAR(15,13,42)
1560 CALL HCHAR(15,18,42)
1570 CALL SOUND(T/2,494,1,294,7,185,9)
1580 CALL HCHAR(15,20,42)
1590 CALL HCHAR(17,10,42)
1600 CALL SOUND(T/2,554,1,294,7,185,9)
1610 CALL HCHAR(17,13,42)
1620 CALL HCHAR(17,17,42)
1630 CALL HCHAR(17,20,42)
1640 CALL SOUND(T/2,494,1,294,7,247,9)
1650 CALL HCHAR(17,22,42)
1660 CALL HCHAR(16,15,42)
1670 CALL SOUND(T/2,440,1,294,7,247,9)
1680 CALL COLOR(14,11,1)
1690 CALL SOUND(T,415,1,330,7,165,9)
1700 CALL VCHAR(18,30,82)
1710 CALL VCHAR(19,30,69,3)
1720 CALL VCHAR(20,30,71)
1730 CALL SOUND(T,330,2,208,6)
1740 CALL VCHAR(22,30,78)
1750 CALL VCHAR(23,30,65)
1760 CALL SOUND(T,330,1,165,6)
1770 CALL SOUND(T,554,0,330,6,110,9)
1780 CALL COLOR(2,14,3)
1790 CALL COLOR(14,12,1)
1800 CALL SOUND(T/2,554,0,330,6,165,9)
1810 CALL COLOR(2,11,3)
1820 CALL SOUND(T/2,587,0,330,6,165,9)
1830 CALL COLOR(14,16,1)
1840 CALL SOUND(T/2,554,0,330,6,220,9)
1850 CALL COLOR(2,10,3)
1860 CALL SOUND(T/2,494,0,330,6,220,3)
1870 CALL COLOR(14,12,1)
1880 CALL SOUND(T,440,0,294,6,147,9)
```

```
1890 CALL COLOR(2,16,3)
1900 CALL COLOR(14,11,1)
1910 CALL SOUND(T,370,0,294,6,220,9)
1920 CALL COLOR(2,12,3)
1930 CALL COLOR(14,16,1)
1940 CALL SOUND(T/2,330,0,220,6,139,9)
1950 CALL COLOR(2,16,3)
1960 CALL SOUND(T/2,330,0,220,6)
1970 CALL COLOR(14,12,1)
1980 CALL SOUND(T,370,0,294,6,147,9)
1990 CALL HCHAR(23,9,72)
2000 CALL HCHAR(23,10,65)
2010 CALL HCHAR(23,11,80,2)
2020 CALL SOUND(T,494,0,370,6,123,9)
2030 CALL HCHAR(23,13,89)
2040 CALL SOUND(T,415,0,294,6,165,9)
2050 CALL HCHAR(23,15,78)
2060 CALL HCHAR(23,16,69)
2070 CALL HCHAR(23,17,87)
2080 CALL SOUND(4*T,440,0,330,6,139,9)
2090 CALL HCHAR(23,19,89)
2100 CALL HCHAR(23,20,69)
2110 CALL HCHAR(23,21,65)
2120 CALL HCHAR(23,22,82)
2130 CALL HCHAR(23,23,33)
2140 CALL COLOR(14,12,1)
2150 CALL COLOR(2,16,3)
2160 CALL COLOR(13,3,9)
2170 CALL COLOR(14,16,1)
2180 CALL COLOR(2,12,3)
2190 CALL COLOR(13,9,3)
2200 GOTO 2140
2210 END
```

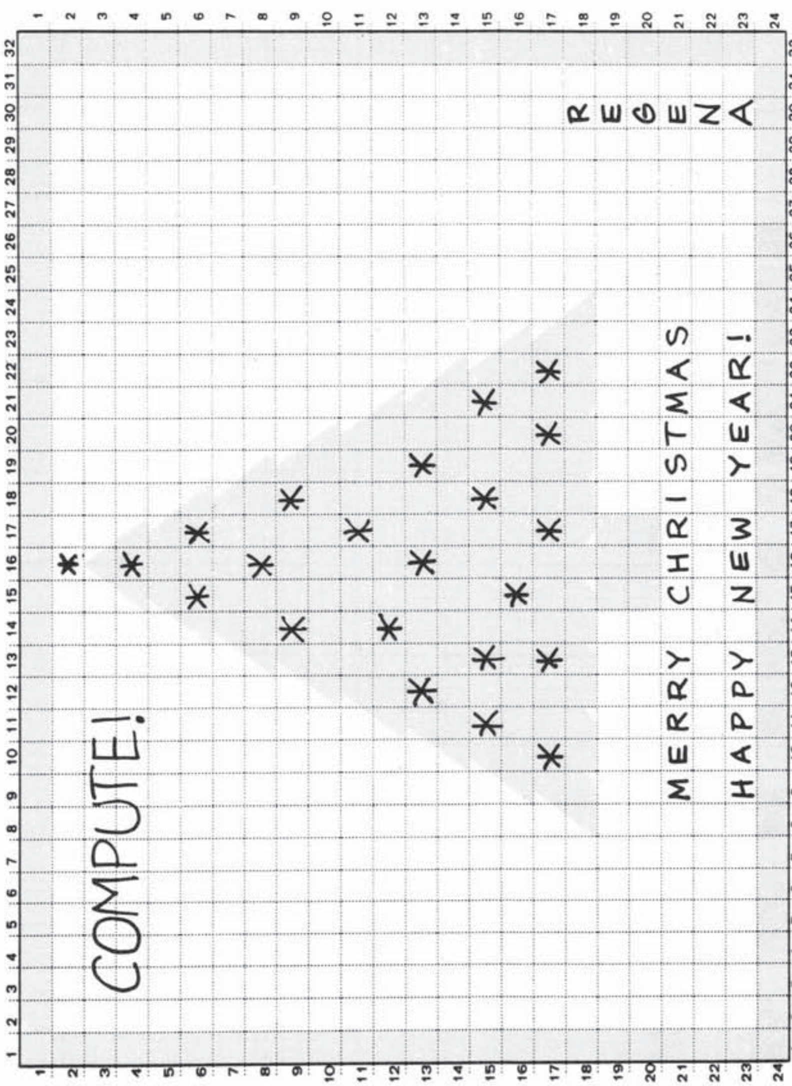


Figure 3-4. Electronic Christmas Card

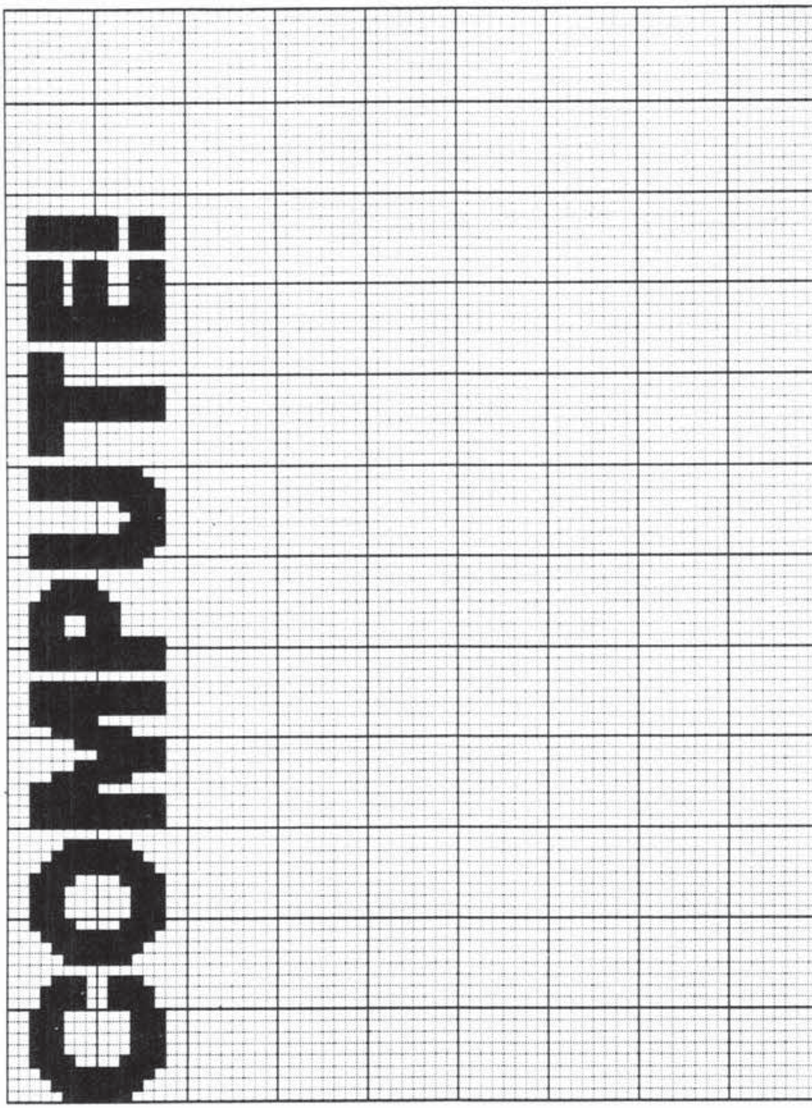


Figure 3-5. Detail of Christmas Card

Noises

You can quickly enliven your game or adventure programs with some "noises." The CALL SOUND statement isn't limited to music — it can produce any type of electronic sound from your computer. Here are a few ideas. But you'll soon find that it is more fun to experiment and create your own sounds.

Beeps

You can use the random function to create "beeps." You might want to use random sounds while you are placing characters randomly on the screen or as you are drawing lines for a maze. The following program illustrates random sounds and was originally written to simulate an old-fashioned computer with blinking lights and sounds. Each tone has a duration of 100 milliseconds, and the tone may be of a frequency from 880 to 1379.

```
100 REM RANDOM TONES
110 CALL SOUND(100,500*RND+880,2)
120 GOTO 110
130 END
```

Sirens

Since you have all the frequencies from 110 to 44733 to work with, you can try to duplicate any tone you have heard. Here is an emergency siren:

```
100 REM EMERGENCY SIREN
110 CALL SOUND(500,563,0)
120 CALL SOUND(500,282,0)
130 GOTO 110
140 END
```

Busy Signal

You may use one, two, or three frequencies and volume in each CALL SOUND statement: CALL SOUND (duration,f1,v1,f2, v2,f3,v3). A very high frequency with a volume of 30 cannot be heard and will create a gap in your noise or sound program:

```
100 REM BUSY SIGNAL
110 CALL SOUND(400,233,2,262,2)
```

```
120 CALL SOUND(10,9999,30)
130 GOTO 110
140 END
```

Interrupting a Sound

The first number in the list of parameters in the CALL SOUND statement is the duration, the number of milliseconds you would like your computer to make the sound. When the computer comes to a CALL SOUND statement, it starts the sound and goes on to execute more statements. The sound continues until the duration time runs out; the next CALL SOUND statement usually waits for the previous one to finish. In the "Emergency Siren" program above, the computer plays a frequency of 563 for 500 milliseconds, then a frequency of 282 for 500 milliseconds, and then repeats until you CLEAR the program.

Sometimes, though, you'll want to start a sound as soon as the computer comes to a CALL SOUND statement, rather than wait for the previous sound to finish. You can do this by using a *negative* number for the duration. CALL SOUND(–500,282,0) will start a sound, with a frequency of 282, as soon as the computer comes to the statement, whether the previous sound is finished or not. The duration will be 500 milliseconds unless it is interrupted by another statement with a negative duration.

Using the negative duration and a FOR-NEXT loop can create a variety of sounds. Here is an example of a FOR-NEXT loop that varies the frequency from 600 to 800, then another loop that varies the frequency from 800 back to 600 to produce a different kind of siren:

```
100 REM SIREN
110 FOR N=600 TO 800 STEP 10
120 CALL SOUND(-200,N,O)
130 NEXT N
140 FOR N=800 TO 600 STEP -10
150 CALL SOUND(-200,N,0)
160 NEXT N
170 GOTO 110
180 END
```

In loops like these, the size of the negative duration number is not really critical because the statement will be executed immediately and the previous sound has not had time to finish. You do need to specify a number large enough to give the computer time to execute the in-between statements (otherwise there will be gaps). Here is another kind of siren:

```
100 REM ALERT SIREN
110 FOR M=440 TO 784 STEP 20
120 CALL SOUND(-99,M,0)
130 NEXT M
140 GOTO 110
150 END
```

Varying the Volume

You can get a different effect by using a variable for the volume and changing it in a FOR-NEXT loop. Here is a doorbell sound:

```
100 REM DING-DONG
110 FOR V=0 TO 16 STEP 2
120 CALL SOUND(-100,659,V,784,V+5)
130 NEXT V
140 FOR V=0 TO 16 STEP 2
150 CALL SOUND(-100,523,V,659,V+5)
160 NEXT V
170 END
```

Making Noises with Negative Frequencies

Besides musical tones, the TI computer has a noise generator. In the CALL SOUND statement, specify a *negative* number from one through eight for the frequency:

CALL SOUND(1000,-8,0)

If you'd like to hear how these noises sound, run this program:

```
100 REM NOISES
110 FOR I=-1 TO -8 STEP -1
120 CALL CLEAR
130 CALL SOUND(1000,I,0)
140 CALL SCREEN(-I+2)
150 PRINT "NOISE NUMBER ";I
```

```
160 CALL SOUND(1,I,30)
170 NEXT I
180 GOTO 110
190 END
```

Notice that in line 110 I used the STEP command to make the FOR-NEXT loop count *downward* by ones instead of the normal upward count, *decrementing* instead of incrementing.

Crashes and Crunches

For one object hitting another, you may want a rather short duration:

CALL SOUND(200,–6,0)

For a bomb sound, you may want a longer duration.

```
100 REM BOMB
110 FOR S=659 to 220 STEP -15
120 CALL SOUND(-200,S,3)
130 NEXT S
140 CALL SOUND(-1000,-6,0)
150 END
```

Try varying the volume in a FOR-NEXT loop to get different effects.

```
100 REM MOTOR 1
110 FOR I=10 TO 1 STEP -1
120 CALL SOUND(-99,-6,I)
130 NEXT I
140 GOTO 110
150 END
```

The frequency can be varied in a loop.

```
100 REM MOTOR 2
110 FOR F=-5 TO -7 STEP -1
120 CALL SOUND(-99,F,0)
130 NEXT F
140 GOTO 110
150 END
```

You can also combine a noise with a regular "musical" frequency.

```
100 REM MOTOR 3
110 FOR I=10 TO 1 STEP -1
120 CALL SOUND(-99,-6,I,110,I)
130 NEXT I
140 FOR I=1 TO 10
150 CALL SOUND(-99,-6,I,110,I)
160 NEXT I
170 GOTO 110
180 END
```

It does not matter what order the noises and frequencies are listed in, but the volume always goes with the frequency that it follows:

```
100 REM OUTER SPACE
110 FOR I=1 TO 30
120 CALL SOUND(-99,1800,2,-5,8)
130 CALL SOUND(-99,1500,2,-6,8)
140 NEXT I
150 END
```

Combine a noise with more than one frequency:

```
100 REM EXPLOSION
110 FOR L=0 TO 16
120 CALL SOUND(-99,-7,L,120,L,131,L)
130 NEXT L
140 END
```

In one CALL SOUND statement you may specify as many as three frequencies, and one noise and a volume for each frequency or noise. With eight noises, 31 volume levels, and over 44,000 frequencies — and you can choose up to three at a time — you could spend quite a bit of time experimenting and trying all the combinations!

A Game to Get You Home

Here is a simple game that illustrates noises and may give you an idea of how to use them in your own games. First, 60 trees are placed randomly on the screen with a random tone for each tree (lines 400-430). Next, 30 white traps are placed randomly on the screen with Noise –1 (lines 440-470). You are placed in the upper left corner of the screen, and you need to use the arrow keys to go to the opposite corner of the screen to your home base. Lines 550 and 950 have Noises –6 and –5 to create noises for each movement of your ship. There is a different noise when you hit a tree (–7) than when you hit a trap (–8).

How "Find Home" Works

Lines

100-180	Clear screen and print title and instructions.
190-280	Define graphics characters and colors.
290-300	Define random number functions for coordinates.
310-330	Wait for the player to read the instructions and press a key.
340-380	Clear the screen; draw the border.
390-430	Randomly place 60 trees.
440-470	Randomly place 30 traps.
480-490	Draw home base.
500-540	Initialize variables for the position of the ship.
550-960	Move the ship, depending on which arrow key was pressed. GCHAR determines what character is in the next square — 120 is the red home base, 128 is a trap, 32 is a blank space, 104 is a tree, and 112 is a border.
970-1010	Procedure if a white trap is hit.
1020-1090	Procedure if the ship reaches home base.
1100-1110	Clear screen and end.

Program 3-10. Find Home

```
100 CALL CLEAR
110 CALL CHAR(64,"3C4299A1A1994237")
120 PRINT "{3 SPACES}** FIND HOME **"
130 PRINT :::"YOU ARE IN A FOREST."
140 PRINT :"USE THE ARROW KEYS TO"
150 PRINT :"GO AS FAST AS YOU CAN"
160 PRINT :"TO YOUR RED HOME BASE."
170 PRINT ::"BEWARE OF WHITE TRAPS!"
```

```
180 PRINT ::
190 CALL CHAR(96,"815A3C66663C5A81")
200 CALL COLOR(9,7,1)
210 CALL CHAR(104,"101038107C10FE1")
220 CALL COLOR(10,3,1)
230 CALL COLOR(11,5,5)
240 CALL COLOR(12,9,9)
250 CALL CHAR(128,"1038387C7CFEFEFF")
260 CALL COLOR(13,16,1)
270 CALL CHAR(136,"FF81BDA5A5BD81FF")
280 CALL COLOR(14,7,16)
290 DEF R22=INT(RND*22)+2
300 DEF R30=INT(RND*30)+2
310 PRINT :::"PRESS ANY KEY TO START."
320 CALL KEY(0,K,S)
330 IF S<1 THEN 320
340 CALL CLEAR
350 CALL HCHAR(1,1,112,32)
360 CALL VCHAR(2,32,112,23)
370 CALL HCHAR(24,1,112,31)
380 CALL VCHAR(2,1,112,22)
390 RANDOMIZE
400 FOR I=1 TO 60
410 CALL SOUND(-50,INT(RND*500)+800,4)
420 CALL HCHAR(R22,R30,104)
430 NEXT I
440 FOR I=1 TO 30
450 CALL SOUND(-50,-1,2)
460 CALL HCHAR(R22,R30,128)
470 NEXT I
480 CALL VCHAR(21,31,120,3)
490 CALL VCHAR(21,32,120,3)
500 X=2
510 Y=2
520 T=0
530 DX=0
540 DY=0
550 CALL SOUND(-200,-6,1)
560 CALL HCHAR(X,Y,96)
570 CALL KEY(1,K,S)
580 T=T+1
590 IF S=0 THEN 950
```

```
600 IF K>5 THEN 950
610 ON K+1 GOTO 830,950,860,890,950,920
620 IF X+DX<24 THEN 650
630 DX=0
640 GOTO 670
650 IF X+DX>1 THEN 670
660 DX=0
670 IF Y+DY<32 THEN 700
680 DY=0
690 GOTO 720
700 IF Y+DY>1 THEN 720
710 DY=0
720 CALL GCHAR(X+DX,Y+DY,CC)
730 IF CC=120 THEN 1020
740 IF CC=128 THEN 970
750 IF CC=32 THEN 790
760 CALL SOUND(100,-7,0)
770 CALL HCHAR(X,Y,32)
780 GOTO 550
790 CALL HCHAR(X,Y,32)
800 X=X+DX
810 Y=Y+DY
820 GOTO 550
830 DX=1
840 DY=0
850 GOTO 620
860 DY=-1
870 DX=0
880 GOTO 620
890 DY=1
900 DX=0
910 GOTO 620
920 DX=-1
930 DY=0
940 GOTO 620
950 CALL SOUND(-200,-5,1)
960 GOTO 620
970 CALL SOUND(500,-8,0,131,0)
980 CALL HCHAR(X,Y,32)
990 CALL HCHAR(X+DX,Y+DY,136)
1000 PRINT "SORRY, GOT CAUGHT!"
1010 GOTO 1060
```

```
1020 CALL HCHAR(X,Y,32)
1030 CALL HCHAR(X+DX,Y+DY,96)
1040 CALL SOUND(1000,-1,0)
1050 PRINT "CONGRATULATIONS! TIME=";T
1060 PRINT :"TRY AGAIN? (Y/N)";
1070 CALL KEY(0,K,S)
1080 IF K=89 THEN 340
1090 IF K<>78 THEN 1070
1100 CALL CLEAR
1110 END
```

Speech

To hear speech on the TI-99/4A, you will need a module that has speech built in and the TI Speech Synthesizer, a small box that attaches to the right side of the computer.

You also need a module to program your own speech. At this writing, there are three modules available. *Speech Editor* was the first module designed to be used with the Speech Synthesizer. *Speech Editor* has about 400 letters, numbers, words, and phrases that can be used with the CALL SAY and CALL SPGET commands.

TI Extended BASIC is another module that allows the use of speech in your own programming. It has the same vocabulary as *Speech Editor* and is designed so you can use speech at the same time you use the features of *Extended BASIC*.

The most versatile command module for speech capabilities is *Terminal Emulator II*. This module is also used, with an RS-232 Interface and a telephone modem, to make your computer act as a terminal to another computer or a large data base. The advantage of *Terminal Emulator II* is that there is unlimited speech — you are not restricted to certain words. You can use allophone numbers to create speech, or you may print words for the computer to speak phonetically. The module comes with an instruction manual.

Programs in this part of the book require *Terminal Emulator II* and the TI Speech Synthesizer. To program with speech, turn the monitor or television on, turn the computer on, and then plug in the *Terminal Emulator II* command module. Press 1 for TI BASIC.

OPEN and PRINT

To use speech in a program, you will need to OPEN the speech device. You may use any number. The statement is:

110 OPEN #1:''SPEECH'',OUTPUT

Once speech has been OPENed, whenever you want the computer to speak simply use the command PRINT #1. Remember to CLOSE speech when you're through with it.

Here is a little program for you to try. You may type in any word or phrase; then the computer will speak it. Notice that the computer pronounces phonetically, according to a few standard rules, and our spoken language does not always follow those rules.

```
100 CALL CLEAR
110 OPEN #1:"SPEECH",OUTPUT
120 PRINT :::"TYPE A WORD OR PHRASE."::
130 INPUT A$
140 PRINT #1:A$
150 GOTO 120
160 END
```

To illustrate that the computer can say anything, try this language demonstration (Program 3-11). Notice that in lines 360-420 the words are spelled phonetically. Your programs involving speech will take some experimentation for the words to sound right.

Program 3-11. Language Demonstration

```
100 REM{3 SPACES}LANGUAGES DEMO
110 OPEN #1:"SPEECH",OUTPUT
120 CALL CLEAR
130 CALL CHAR(128,"080818FF7E346681")
140 CALL COLOR(13,16,6)
150 PRINT TAB(5);"LANGUAGES DEMO"
160 PRINT :::TAB(5);"CHOOSE"
170 PRINT :TAB(7);"1 ENGLISH"
180 PRINT :TAB(7);"2 FRENCH"
190 PRINT :TAB(7);"3 SPANISH"
200 PRINT :TAB(7);"4 GERMAN"
210 PRINT :TAB(7);"5 JAPANESE"
```

```
220 PRINT :TAB(7);"6 END PROGRAM"::::
230 CALL HCHAR(2,2,128,30)
240 CALL VCHAR(3,2,128,22)
250 CALL VCHAR(3,31,128,22)
260 CALL HCHAR(24,2,128,30)
270 CALL SOUND(150,1397,4)
280 CALL KEY(0,K,S)
290 IF (K<49)+(K>54)THEN 280
300 CALL HCHAR(2*(K-48)+8,7,62)
310 ON K-48 GOSUB 340,360,380,400,420,440
320 CALL VCHAR(10,7,32,10)
330 GOTO 270
340 PRINT #1:"^1 2 3 4 5 6 7 8 9 TEN"
350 RETURN
360 PRINT #1:"^UN DU TWA KATR SAYNK CEES SE
    T WEET NUF DEES"
370 RETURN
380 PRINT #1:"^OONO DOSE TRACE QUATRO SEENQ
    O SASE SEE ETA O CHO NUEVA DEE S"
390 RETURN
400 PRINT #1:"^EYENS TSWIE DRY FEAR FOONF S
    ECHS ZEEBEN AUKT NOYN TSAYN"
410 RETURN
420 PRINT #1:"^EECHEE NEE SAWN SHE GO HEECH
    EE HAWCHEE HRO KU KOO JOO"
430 RETURN
440 CALL CLEAR
450 END
```

Speech Separators

Speech separator symbols may be used to create pauses and some inflections in the voice. You may use a space between words or letters for a slight pause. Other separating symbols are the comma, the semicolon, the colon, the period, the exclamation point, and the question mark.

Listen to the differences in the following program.

```
100 REM SEPARATORS
110 CALL CLEAR
120 OPEN #1:"SPEECH",OUTPUT
130 PRINT "SPEECH WITH SEPARATORS"::
```

```
140 FOR I=1 TO 7
150 READ A$
160 PRINT ::A$
170 PRINT #1:A$
180 NEXT I
190 DATA HELLO LEWIS THIS IS A TEST
200 DATA "HELLO LEWIS, THIS IS A TEST,"
210 DATA "HELLO LEWIS; THIS IS A TEST;"
220 DATA "HELLO LEWIS: THIS IS A TEST:"
230 DATA "HELLO LEWIS. THIS IS A TEST."
240 DATA "HELLO LEWIS! THIS IS A TEST!"
250 DATA "HELLO LEWIS? THIS IS A TEST?"
260 END
```

Inflections

You may also change inflection with a stress mark. The caret (^) is used to indicate a primary stress point, and you may use only one such mark per line. The underline symbol (_) is used to indicate a secondary stress point. The greater-than sign (>) is used to shift stress points within a word.

This demonstration program shows what happens when you put the primary stress point in different places in a sentence.

```
100 REM STRESS POINT
110 CALL CLEAR
120 OPEN #1:"SPEECH",OUTPUT
130 PRINT "SPEECH WITH PRIMARY STRESS"
140 FOR I=1 TO 5
150 READ A$
160 PRINT ::A$
170 PRINT #1:A$
180 NEXT I
190 DATA "HEAR THIS STRESS POINT"
200 DATA "^HEAR THIS STRESS POINT"
210 DATA "HEAR ^THIS STRESS POINT"
220 DATA "HEAR THIS ^STRESS POINT"
230 DATA "HEAR THIS STRESS ^POINT"
240 END
```

Two more parameters which you may specify to vary the voice are the *pitch period* and the *slope level*. The form is the string *"//xx yyy"* where *xx* is the pitch period and *yyy* is the slope level indication. The space between *xx* and *yyy* is required.

The pitch should be a number from 0 through 63, and the slope level should be a number from 0 through 255. The manual recommends that the best results occur when the slope is 32 times 10% of the pitch. If you do not specify pitch and slope, the default values are 43 and 128. With a little experimentation, you can make the computer voice do just what you want it to do.

To give you an idea of how the pitch and slope level numbers change the sound of the voice, here are some demonstration programs. The first program varies the pitch from 0 to 63 and sets the slope level at the recommended ratio.

```
100 REM PITCH & SLOPE 1
110 CALL CLEAR
120 OPEN #1:"SPEECH",OUTPUT
130 FOR P=0 TO 63
140 S=INT(3.2*P+.5)
150 B$="//"&STR$(P)&" "&STR$(S)
160 PRINT B$
170 PRINT #1:B$
180 PRINT #1:"NOW HEAR THIS."
190 NEXT P
200 END
```

The second demonstration program varies the pitch and the slope. You will notice that in some combinations of pitch and slope the speech is garbled.

```
100 REM PITCH & SLOPE 2
110 CALL CLEAR
120 OPEN #1:"SPEECH",OUTPUT
130 FOR S=0 TO 255
140 FOR P=0 TO 63
150 B$="//"&STR$(P)&" "&STR$(S)
160 PRINT B$
170 PRINT #1:B$
```

```
180 PRINT #1:"NOW HEAR THIS."
190 NEXT P
200 NEXT S
210 END
```

The third demonstration program varies the slope for different pitches. Since there are 255 variations for the slope level, I increment the slope level by 20 instead of 1. If you want to get to the next pitch level without going through all the slope levels, press any key.

```
100 REM PITCH & SLOPE 3
110 CALL CLEAR
120 OPEN #1:"SPEECH",OUTPUT
130 FOR P=0 TO 63
140 FOR S=0 TO 255 STEP 20
150 B$="//"&STR$(P)&" "&STR$(S)
160 PRINT B$
170 PRINT #1:B$
180 PRINT #1:"NOW HEAR THIS."
190 CALL KEY(0,K,ST)
200 IF ST<>0 THEN 220
210 NEXT S
220 NEXT P
230 END
```

The fourth demonstration program on pitch and slope allows you to enter values for the pitch and the slope. The computer will then say the phrase ''Hear this test'' using the values you have entered. To stop the program, press CLEAR.

```
100 REM PITCH & SLOPE 4
110 CALL CLEAR
120 OPEN #1:"SPEECH",OUTPUT
130 PRINT "SPEECH WITH PITCH AND SLOPE"
140 PRINT :"PITCH MAY BE FROM 0 TO 63."
150 PRINT :"SLOPE MAY BE FROM 0 TO 255."
160 PRINT :"BEST RATIO: SLOPE=3.2*PITCH":::
170 INPUT "PITCH = ":P
180 IF (P>=0)+(P<=63)=-2 THEN 210
190 PRINT :"SORRY.  0<P<63"::
```

```
200 GOTO 170
210 INPUT "SLOPE = ":S
220 IF (S>=0)+(S<=255)=-2 THEN 250
230 PRINT :"SORRY.  0<S<255"::
240 GOTO 210
250 B$="//"&STR$(P)&" "&STR$(S)
260 PRINT ::B$::::
270 PRINT #1:B$
280 PRINT #1:"HEAR THIS TEST."
290 GOTO 170
300 END
```

You can probably think of all sorts of uses for speech in your programs — everything from comments in games to teaching foreign languages.

Spelling Practice

It seems that one of the standards in education is weekly spelling tests. This program lets you set up the week's words and save the program on cassette. Any time during the week, students can load the tape and practice at their own pace.

The words are chosen in a random order from the original list. A word is spoken. If you want to hear the word again, press ENTER.

When you're ready, type in the word. If it is correct, a star appears and the word won't appear again. If the spelling is wrong, you get one more chance to try. If it is spelled incorrectly the second time, the correct word appears and you are given time to review it. That word will then be used again, later in the list.

A sample list of words is given in this program. To use your own word list, start at line 810 DATA. First type the word correctly spelled, then a comma, then type the word spelled phonetically. Continue through the spelling list, separating each pair of items with a comma. Be sure the last two entries are @,@ to signify the end of the data. If you have more than 20 words, change the DIM statement in line 110.

Be sure to experiment to make sure the words sound right as spoken by the computer. You may want to add your own graphics to make this program more interesting for your student.

Program 3-12. Spelling Practice

```
100 REM  SPELLING
110 DIM W$(20),S$(20)
120 OPEN #1:"SPEECH",OUTPUT
130 CALL CLEAR
140 CALL CHAR(64,"3C4299A1A199423C")
150 CALL COLOR(2,14,16)
160 PRINT "{3 SPACES}**********************"
    :"{3 SPACES}*";TAB(24);"*":"{3 SPACES}*
     SPELLING PRACTICE *"
170 PRINT "{3 SPACES}*";TAB(24);"*":"
    {3 SPACES}**********************"::::::
    ::::
180 CALL CHAR(128,"0101010303FF3F")
190 CALL CHAR(129,"000080C0C0E0FFFC")
200 CALL CHAR(130,"0F070F0F1E1C302")
210 CALL CHAR(131,"F0F0F878381C0C04")
220 CALL COLOR(13,12,1)
230 I=1
240 RESTORE
250 READ W$(I),S$(I)
260 IF W$(I)="@" THEN 310
270 CALL COLOR(2,16,14)
280 I=I+1
290 CALL COLOR(2,14,16)
300 GOTO 250
310 N=I-1
320 CALL CLEAR
330 CALL COLOR(2,2,1)
340 CALL SCREEN(12)
350 PRINT "PRACTICE YOUR SPELLING!":::"YOU
    WILL HEAR A WORD."::"TYPE THE WORD"::
    "THEN PRESS <ENTER>."
360 PRINT :::"IF YOU WANT TO HEAR THE"::"WO
    RD AGAIN, JUST"::"PRESS <ENTER>."
370 PRINT ::::"PRESS ANY KEY TO START."::
380 CALL KEY(0,K,S)
390 IF S=0 THEN 380
400 CALL CLEAR
410 CALL SCREEN(8)
420 FOR I=1 TO N
```

```
430 T=Ø
440 RANDOMIZE
450 R=INT(N*RND)+1
460 IF W$(R)="" THEN 45Ø
470 CALL CLEAR
480 PRINT #1:S$(R)
490 CALL SOUND(15Ø,1397,2)
500 INPUT X$
510 IF X$="" THEN 47Ø
520 IF X$=W$(R)THEN 64Ø
530 CALL SOUND(1ØØ,33Ø,2)
540 CALL SOUND(1ØØ,262,2)
550 T=T+1
560 IF T=2 THEN 6ØØ
570 PRINT #1:"^TRY AGAIN."
580 PRINT #1:"^SPELL",S$(R)
590 GOTO 49Ø
600 PRINT ::"  ";W$(R)
610 PRINT ::"PRESS ANY KEY TO CONTINUE."
620 CALL KEY(Ø,K,S)
630 IF S=1 THEN 43Ø ELSE 62Ø
640 CALL SOUND(1ØØ,262,2)
650 CALL SOUND(1ØØ,33Ø,2)
660 CALL SOUND(1ØØ,392,2)
670 CALL SOUND(3ØØ,523,2)
680 PRINT TAB(15);CHR$(128);CHR$(129):TAB(1
    5);CHR$(13Ø);CHR$(131)::::
690 W$(R)=""
700 NEXT I
710 CALL CLEAR
720 PRINT "WANT TO TRY AGAIN? (Y/N)"::
730 CALL KEY(Ø,K,S)
740 IF K=78 THEN 78Ø
750 IF K<>89 THEN 73Ø
760 CALL CLEAR
770 GOTO 23Ø
780 CALL CLEAR
790 CLOSE #1
800 STOP
810 DATA ALWAYS,^ALWAYS.,DADDY,^DADDY.,OFF,
    ^OFF.,SISTER,^SISTER.,LETTER,^LETTER.
    ,START,^START.,HAPPY,^HAPPY.
```

```
820 DATA RING,^RING.,WASH,^WASH.,FALL,^FALL
    .,SLEEP,^SLEEP.,ONCE,^ONCE.,SADLY,^SA
    DLY.,DRESS,^DRESS.
830 DATA SET,^SET.,ROUND,^ROUND.,@,@
840 END
```

Using the Speech Synthesizer with Non-readers

"Colors" is a program designed to teach a two-year-old the names of the colors. It could also be used to teach beginning readers how to read the color names in lowercase letters.

In a random order, a color name appears on the screen. After a short delay, the color itself appears and the computer says the color name. If you would like a longer delay (for example, for someone practicing reading), put a larger number in place of 300 in line 540.

After the color appears, the user may press ENTER for another color or E to end the program.

How "Colors" Works

Lines	
110-120	Clear screen, select cyan as screen color.
130-170	DATA to define characters for lowercase letters.
180-200	Print title screen.
210-280	Define blocks of solid colors for the first two character numbers in each of sets 13 through 16.
290-380	Draw color bars on title screen.
390	Print instructions.
400	OPEN the speech device.
410	Randomize choices.
420-470	Define characters for letters; branch.
480-510	Choose a color.
520	Read character number; three lines to PRINT in order to spell the color name in big lowercase letters; and the phonetic pronunciation.
530	Print the color name.
540-550	Delay before drawing color.
560-580	Draw block of color.
590	Say color name.
600-620	Wait for user to press a key. If E is pressed, the program ends; if ENTER is pressed, the program branches back to choose another color; any other key is ignored.

630-860	RESTORE the appropriate DATA for the color chosen.
870-890	Clear screen, CLOSE speech device, and end program.

Program 3-13. Colors

```
100 REM{4 SPACES}COLORS
110 CALL CLEAR
120 CALL SCREEN(8)
130 DATA Ø,3D4381818181433D,BCC2818181C2B
    C,3C4280808080423C,ØØØØ0101010101,3
    C4281FF8Ø8Ø423C,1824222020202ØF8
140 DATA Ø1Ø1Ø10131221C,ØØØØ8Ø8Ø8Ø8Ø8Ø8,ØØØ
    ØØØØ8,Ø8Ø8Ø8Ø8Ø8887,889ØAØCØAØ8Ø8884,
    Ø8Ø8Ø8Ø8Ø8Ø8Ø8Ø8
150 DATA 3C84Ø2Ø2Ø2Ø2Ø2Ø2,BCC2818181818181,
    3C4281818181423C,8Ø8Ø8Ø8Ø8Ø8,Ø1Ø1Ø1Ø1
    Ø1Ø1,BCC2818Ø8Ø8Ø8Ø8
160 DATA 3C42403CØ2Ø2423C,ØØØØ8Ø8Ø8Ø87FØ8,
    818181818181433D,414122221414Ø8Ø8,Ø4Ø
    48888505Ø2Ø2,824428102844828 2
170 DATA 1Ø1Ø2Ø2Ø4Ø4,7FØ2Ø4Ø81Ø2Ø4Ø7F,2Ø2Ø2
    Ø2Ø2Ø2Ø2Ø2
180 PRINT TAB(1Ø);"C O L O R S"::::
190 CALL CHAR(64,"3C4299A1A199423C")
200 PRINT
210 FOR C=128 TO 152 STEP 8
220 CALL CHAR(C,"Ø")
230 CALL CHAR(C+1,"FFFFFFFFFFFFFFFF")
240 NEXT C
250 CALL COLOR(13,7,12)
260 CALL COLOR(14,6,4)
270 CALL COLOR(15,14,16)
280 CALL COLOR(16,2,15)
290 C=128
300 FOR D=5 TO 23 STEP 6
310 CALL VCHAR(8,D,C,6)
320 CALL VCHAR(8,D+1,C,6)
330 CALL VCHAR(8,D+2,C,6)
340 CALL VCHAR(8,D+3,C+1,6)
```

```
350 CALL VCHAR(8,D+4,C+1,6)
360 CALL VCHAR(8,D+5,C+1,6)
370 C=C+8
380 NEXT D
390 PRINT :::"AFTER EACH SCREEN,"::"PRESS <
    ENTER> TO CONTINUE"::"PRESS <E> TO EN
    D PROGRAM.";
400 OPEN #1:"SPEECH",OUTPUT
410 RANDOMIZE
420 FOR C=96 TO 123
430 READ C$
440 CALL CHAR(C,C$)
450 CALL SOUND(-150,INT(RND*1000)+200,4)
460 NEXT C
470 GOTO 600
480 RANDOMIZE
490 C=INT(RND*8)+1
500 CALL CLEAR
510 ON C GOSUB 630,660,690,720,750,780,810,
    840
520 READ D,A$,B$,C$,D$
530 PRINT TAB(9);A$:TAB(9);B$:TAB(9);C$
540 FOR T=1 TO 300
550 NEXT T
560 FOR I=8 TO 13
570 CALL HCHAR(I,10,D,13)
580 NEXT I
590 PRINT #1:D$
600 CALL KEY(0,K,S)
610 IF K=69 THEN 870
620 IF K=13 THEN 490 ELSE 600
630 RESTORE 640
640 DATA 129,`````d,`r`e`a,,^RED
650 RETURN
660 RESTORE 670
670 DATA 128,````l`l,v`e`l`l`o`vw,y,^YELLO
680 RETURN
690 RESTORE 700
700 DATA 136,,a`r`e`e`n,g,^GREEN
710 RETURN
720 RESTORE 730
730 DATA 137,h`l,b`l`u`e,,^BLUE
```

```
740 RETURN
750 RESTORE 760
760 DATA 144,```h`i`t,vw`n`l`l`e,,^WHITE
770 RETURN
780 RESTORE 790
790 DATA 145,````````l,b`u`r`b`l`e,p`````p,
    ^PURPL
800 RETURN
810 RESTORE 820
820 DATA 153,h`l`````h,b`l`a`c`k,,^BLACK
830 RETURN
840 RESTORE 850
850 DATA 152,,a`r`a`v,g`````y,^GRAY
860 RETURN
870 CALL CLEAR
880 CLOSE #1
890 END
```

Teaching a Foreign Language

Here is a program to teach ten basic German words. The same logic may be used to teach different words or even a different language.

As pictures are drawn, the German word is spoken. After the ten words are presented, there is a quiz in which a German word is spoken and a question mark appears on one of the pictures.

If the question mark is on the correct picture, press ENTER.

If you want to move the question mark, press the space bar and the question mark will move to a different picture.

The words are chosen in a random order. You must get the picture correct to continue the quiz. If you get the picture correct with the first response, that word will not reappear; however, if the word has been missed at least once, the word will reappear before the end of the quiz.

Program 3-14. German

```
110 REM  GERMAN
120 OPEN #1:"SPEECH",OUTPUT
130 CALL CLEAR
140 PRINT TAB(8);"G E R M A N"
```

```
150 PRINT :::"FIRST YOU WILL BE TOLD"::"TEN
     GERMAN WORDS."
160 PRINT ::"NEXT THERE WILL BE A QUIZ."
170 PRINT :"LISTEN TO THE GERMAN WORD."
180 PRINT :"PRESS <ENTER> IF '?' IS"
190 PRINT :"ON THE RIGHT PICTURE."
200 PRINT :"PRESS THE SPACE BAR TO MOVE"::"
    THE QUESTION MARK."
210 PRINT ::"PRESS ANY KEY TO START.";
220 CALL KEY(Ø,K,S)
230 IF S<1 THEN 22Ø
240 CALL CLEAR
250 FOR I=2 TO 1Ø
260 READ S$(I),X(I),Y(I)
270 T$(I)=S$(I)
280 J=8*(I+3)
290 CALL CHAR(J,"FFFFFFFFFFFFFFFF")
300 READ N
310 FOR C=J+1 TO J+N
320 READ C$
330 CALL CHAR(C,C$)
340 NEXT C
350 READ F
360 CALL COLOR(I,F,8)
370 NEXT I
380 DATA _DOS ^HOUSE,21,1Ø,2,Ø1Ø3Ø7ØF1F3F7F
    FF,8ØCØEØFØF8FCFEFF,9,_D ^TEUR,2Ø,14,
    Ø,4,_DOS ^FENSTER,18,18,Ø,14
390 DATA _DOS ^DOGHC,11,13,2,Ø1Ø3Ø7ØF1F3F7F
    FF,8ØCØEØFØF8FCFEFF,11
400 DATA _D ^SHORNSTINE,12,17,1,FF7F3F1FØFØ
    7Ø3Ø1,2
410 DATA _DARE ^ROZN,23,26,1,8Ø88A8ECEEFEFE
    FF,3
420 DATA _D ^VO KA,4,6,4,Ø3Ø7ØFØF1F7FFFFF,7
    F3FØ7Ø1,FFFFFFFFFF7F3F1F,FFFCF8F8FØEØ
    CØC,16
430 DATA _D ^ZO NA,3,3Ø,4,7F7F7F7F3F3F1F1F,
    1FØFØFØ7Ø7Ø3Ø1,7F3F1FØ7Ø1,FFFFFFFFFF7
    FØF,12
440 DATA _D ^ROUHC,7,17,2,1Ø18183838383C3C,
    3C3C3E7E7E7E7E7E,15
```

```
450 CALL CHAR(112,"FFFEFCF8F0E0C08")
460 CALL CHAR(113,"FF7F3F1F0F070301")
470 CALL COLOR(11,11,9)
480 CALL COLOR(6,2,11)
490 S$(1)=" DARE HIMMEL"
500 T$(1)=S$(1)
510 X(1)=9
520 Y(1)=22
530 CALL COLOR(1,1,8)
540 PRINT #1:S$(1)
550 GOSUB 1590
560 R=16
570 FOR C=8 TO 13
580 CALL HCHAR(R,C,41)
590 CALL VCHAR(R+1,C,40,C)
600 R=R-1
610 NEXT C
620 R=11
630 FOR C=14 TO 19
640 CALL HCHAR(R,C,42)
650 CALL VCHAR(R+1,C,40,24-R)
660 R=R+1
670 NEXT C
680 PRINT #1:S$(2)
690 GOSUB 1590
700 CALL VCHAR(18,13,48,7)
710 CALL VCHAR(18,14,48,7)
720 CALL VCHAR(18,15,48,7)
730 PRINT #1:S$(3)
740 GOSUB 1590
750 CALL VCHAR(18,10,56,2)
760 CALL VCHAR(18,11,56,2)
770 CALL VCHAR(18,17,56,2)
780 CALL VCHAR(18,18,56,2)
790 PRINT #1:S$(4)
800 GOSUB 1590
810 FOR C=7 TO 12
820 CALL HCHAR(R,C,65)
830 CALL HCHAR(R,C+1,112)
840 R=R-1
850 NEXT C
```

```
860 CALL HCHAR(11,13,65)
870 CALL HCHAR(11,14,66)
880 R=12
890 FOR C=14 TO 19
900 CALL HCHAR(R,C,113)
910 CALL HCHAR(R,C+1,66)
920 R=R+1
930 NEXT C
940 PRINT #1:S$(5)
950 GOSUB 1590
960 CALL VCHAR(11,17,72,3)
970 CALL VCHAR(14,17,73)
980 PRINT #1:S$(6)
990 GOSUB 1590
1000 CALL HCHAR(23,1,81,32)
1010 CALL HCHAR(24,1,80,32)
1020 PRINT #1:S$(7)
1030 GOSUB 1590
1040 CALL HCHAR(4,3,89)
1050 CALL HCHAR(4,4,88,6)
1060 CALL HCHAR(4,10,92)
1070 CALL HCHAR(5,3,90)
1080 CALL HCHAR(5,4,91)
1090 CALL HCHAR(5,5,88,4)
1100 CALL HCHAR(5,9,92)
1110 PRINT #1:S$(8)
1120 GOSUB 1590
1130 CALL HCHAR(1,29,96,4)
1140 CALL HCHAR(2,29,97)
1150 CALL HCHAR(2,30,96,3)
1160 CALL HCHAR(3,29,98)
1170 CALL HCHAR(3,30,96,3)
1180 CALL HCHAR(4,30,99)
1190 CALL HCHAR(4,31,100)
1200 CALL HCHAR(4,32,96)
1210 PRINT #1:S$(9)
1220 GOSUB 1590
1230 CALL VCHAR(7,17,106,4)
1240 CALL VCHAR(6,17,105)
1250 PRINT #1:S$(10)
1260 GOSUB 1590
1270 FOR I=1 TO 10
```

```
1280 T=Ø
1290 RANDOMIZE
1300 R=INT(RND*1Ø+1)
1310 IF S$(R)="" THEN 1300
1320 FOR J=1 TO 1Ø
1330 PRINT #1:S$(R)
1340 CALL GCHAR(X(J),Y(J),C)
1350 CALL KEY(Ø,K,S)
1360 CALL HCHAR(X(J),Y(J),63)
1370 CALL HCHAR(X(J),Y(J),C)
1380 IF K=13 THEN 1420
1390 IF K<>32 THEN 1350
1400 NEXT J
1410 GOTO 1320
1420 IF J=R THEN 1480
1430 CALL SOUND(150,330,2)
1440 CALL SOUND(150,262,2)
1450 T=1
1460 CALL SOUND(1,9999,30)
1470 GOTO 1330
1480 CALL SOUND(150,262,2)
1490 CALL SOUND(150,330,2)
1500 CALL SOUND(150,392,2)
1510 CALL SOUND(300,523,2)
1520 IF T=Ø THEN 1550
1530 I=I-1
1540 GOTO 1560
1550 S$(R)=""
1560 CALL SOUND(1,9999,30)
1570 NEXT I
1580 GOTO 1620
1590 FOR D=1 TO 500
1600 NEXT D
1610 RETURN
1620 CALL CLEAR
1630 FOR I=1 TO 1Ø
1640 CALL COLOR(I,2,1)
1650 NEXT I
1660 CALL CHAR(65,"ØØ3844447C444444")
1670 CALL CHAR(72,"ØØ4444447C444444")
1680 CALL CHAR(73,"ØØ381Ø1Ø1Ø1Ø1Ø38")
1690 PRINT "DER HIMMEL"
```

```
1700 PRINT #1:T$(1)
1710 PRINT :"DAS HAUS"
1720 PRINT #1:T$(2)
1730 CALL CHAR(144,"00000000000044")
1740 PRINT :"DIE TUR"
1750 CALL HCHAR(22,8,144)
1760 PRINT #1:T$(3)
1770 PRINT :"DAS FENSTER"
1780 PRINT #1:T$(4)
1790 PRINT :"DAS DACH"
1800 PRINT #1:T$(5)
1810 PRINT :"DIE SCHORNSTEIN"
1820 PRINT #1:T$(6)
1830 PRINT :"DER RASEN"
1840 PRINT #1:T$(7)
1850 PRINT :"DIE WOLKE"
1860 PRINT #1:T$(8)
1870 PRINT :"DIE SONNE"
1880 PRINT #1:T$(9)
1890 PRINT :"DIE RAUCH"
1900 PRINT #1:T$(10)
1910 CLOSE #1
1920 END
```

Chapter 4

Going Somewhere

Chapter 4

Going Somewhere

Changing the Sequence

As you enter a program, each line is numbered. As the program is run, the computer executes each statement in numerical order, unless there is a command telling the computer to branch — to go to some other line number. I explained some of these commands earlier, as they were needed.

Unconditional Branching

GOTO is a command that tells the computer to go immediately to a different line rather than to the next one in numerical order. You may GOTO a previous line number, a later line number, or the same line number. If you have a statement that commands the computer to GOTO its own line number, the computer stays at that line until you press CLEAR to interrupt the program.

Figure 4-1. Unconditional Branches

```
100 REM GOTO
110 GOTO 1000
120 PRINT "HELLO"

500 GOTO 500

1000 PRINT "PROGRAM BRANCHED TO 1000"
1010 GOTO 120
1020 END
```

Conditional Branches

You may have a conditional branch by using an IF-THEN statement. The command IF is followed by an expression, like A=B, or N>=A/55. If the expression is true, the computer branches to the line number that follows the command THEN. If the condition is false, the computer goes on to execute the very next line in order.

```
100 REM IF-THEN
110 CALL CLEAR
120 I=1
130 PRINT I
140 I=I+1
150 IF I<11 THEN 130
160 END
```

In this simple program, the numbers are printed. The variable *I* is incremented by one each time. If *I* is less than 11, the program branches to line 130, where the value is printed. As soon as *I* is equal to or greater than 11, the program goes to line 160 and ends.

TI BASIC also allows the use of ELSE. Change line 150 above to

150 IF I<11 THEN 130 ELSE 120

This statement says IF the variable I has a value less than the number 11, THEN branch to line 130, otherwise (or ELSE) branch to line 120. Your program should now be:

```
100 REM IF-THEN-ELSE
110 CALL CLEAR
120 I=1
130 PRINT I
140 I-I+1
150 IF I<11 THEN 130 ELSE 120
160 END
```

RUN the program and see how it has changed from the first example. Notice that there is now no way for the computer to reach line 160.

Remember, there are almost always several ways a program can be written to produce the same result. Line 150 could be written:

150 IF I > 10 THEN 120 ELSE 130

Make this change, then RUN the program. The result should be the same as in the previous example.

Finite Loops

Another way to change the order in which the computer executes statements is to use a FOR-NEXT loop. When the computer reaches NEXT, there is a conditional branch, either back to FOR or on to the next line. Here is a program to print the numbers from 1 to 10.

```
100 REM FOR-NEXT
110 CALL CLEAR
120 FORI=1 TO 10
130 PRINT I
140 NEXT I
150 END
```

RUN the program. The results are the same as those in the IF-THEN example.

In this program, first the screen is cleared. Next the computer lets the variable *I* start at the value of 1. Line 130 prints *I*. Line 140 tells the computer to increment *I* and go to the statement just after the FOR statement. This is looping. *I* is assigned the value of 2; the computer prints 2, increments *I*, and so forth — until the limit of 10 is reached. Unlike a GOTO loop, a FOR-NEXT loop has an ending built in.

The last time through the loop, the value for *I* is 10. Then, when the computer hits the NEXT statement, it increments *I*, which makes *I* equal to 11. The computer then tests to see if *I* is within the limit of 10. Since *I* has exceeded the limit, the computer goes to the next statement.

If you would like this program to be like the IF-THEN-ELSE example, add line 145:

145 GOTO 120

The computer will go through the FOR-NEXT loop printing the numbers. After the loop is finished, the computer hits the statement GOTO 120, branches to line 120, and starts the

FOR-NEXT loop again. This GOTO loop will go on endlessly, unless you intervene to stop it.

Controlling the Counter

In a regular FOR-NEXT loop, the counter variable is incremented by 1 each time the loop is performed. However, you can specify the step size if you do not want the increment to be 1. Let's go back to the original FOR-NEXT example program and change line 120:

120 FOR I=1 TO 10 STEP 2

Your program should now look like this:

```
100 REM FOR-NEXT
110 CALL CLEAR
120 FOR I=1 TO 10 STEP 2
130 PRINT I
140 NEXT I
150 END
```

RUN the program. You will see from the PRINTed values of *I* that *I* starts with the value of 1, then increments by twos until it is past the limit of 10.

Your step size can be a negative number:

```
100 REM FOR-NEXT-STEP
110 CALL CLEAR
120 FOR I=10 TO 1 STEP -1
130 PRINT I
140 NEXT I
150 END
```

Multiple Branches

A variation of the IF-THEN statement is ON-GOTO, which allows more possible branches from the same statement. However, the value of the evaluated expression must be more tightly controlled.

```
100 REM ON-GOTO
110 CALL CLEAR
120 PRINT:"PRESS 1, 2, 3, OR 4"
130 CALL KEY(0,K,S)
```

```
140 IF K<49 THEN 130
150 IF K>52 THEN 130
160 A=K-48
170 ON A GOTO 1000,2000,3000,4000
1000 PRINT "1"
1010 GOTO 120
2000 PRINT "2"
2010 GOTO 120
3000 PRINT "3"
3010 GOTO 120
4000 PRINT "4"
4010 GOTO 120
4020 END
```

The screen is cleared, and the message ''PRESS 1, 2, 3, OR 4'' is printed. CALL KEY scans which key you press.

If *K*, the value in ASCII code of the key pressed, is less than 49 or greater than 52, this means 1, 2, 3, or 4 has not been pressed, and the program branches back to the CALL KEY statement.

When one of the correct keys has been pressed, the value of *A* is set equal to *K*-48, so *A* will be a number from 1 to 4. In line 170, the computer branches depending on the value of *A*. If *A* is equal to 1, the program goes to the first number, line 1000. If A is equal to 2, the program goes to the second number, line 2000, etc.

You do have to be careful when using ON-GOTO. The value of *A* must not be less than 1, or greater than the number of line numbers you have listed, or your program will crash. In the above program, the values for *K* are checked so the value of *A* will always be 1, 2, 3, or 4.

You may use an expression rather than a variable in the ON-GOTO statement. This program could be changed by deleting line 160 and changing line 170 to

170 ON K-48 GOTO 1000,2000,3000,4000

Again, just make sure, before using the ON-GOTO statement, that the expression cannot turn out to be less than 1 or greater than the number of line numbers listed.

Logical OR and AND

In this sample program, lines 140 and 150 may be combined into one IF statement. Delete line 150 and change line 140 to

140 IF (K<49)+(K>52) THEN 130

The plus sign (+) is a "logical OR," which indicates that if *either* K<49 *or* K>52 is true, then the program must branch to line 130.

The IF-THEN statement may be written other ways to get the same result. Line 140 could also be written:

140 IF (K>48)*(K<53) THEN 170 ELSE 130

The asterisk (*) is the "logical AND" sign. If *both* K>48 *and* K<53 are true, then the program must branch to line 170; otherwise it goes back to line 130.

True Plus True Is Minus Two

Another way line 140 could be written is:

140 IF (K>48)+(K<53)< >-2 THEN 130

This statement depends on the fact that *true* and *false* have a numerical value in BASIC. If a true expression is evaluated, you will get a value of -1. If the expression is false, the value is zero.

In this case, if K>48 is true and K<53 is true, the sum of the two expressions added together will be -2. If at least one of the statements is false, then the value is not -2, and you branch back to line 130.

This should give you an idea of how logical OR and AND work.

Here is an example of logical OR: IF (K=50)+(X=55) THEN 500. The IF command tests to see whether the expression is false — whether it returns a value of zero. In this case, if either K=50 or X=55 is true, then IF will evaluate a result of either (-1)+0 or 0+(-1). The value is not zero, the expression is not false, and so the program will branch. If *both* are true, then IF evaluates a result of (-1)+(-1), which is -2; this is still not zero, and so is not false; again the program will branch.

With logical AND, however, the computer *multiplies* the values, so that if only one of the expressions is true, IF will end up evaluating either 0*(-1) or (-1)*0. Either way, the result is 0, or false, and the program will *not* branch. Only if *both* expressions are true will the result be non-zero: (-1)*(-1)=1. So only if both expressions are true will the program branch.

Branching in Action

This program is called "Homework Helper: Factors" because it is designed to help a student quickly check the answers to an assignment with problems involving factoring. The student will learn most by doing the class assignment in the usual way, writing the problem down on paper and working it out step by step. "Homework Helper" is then used only to check the answers.

The program has four sections.

All factors. The student enters a number, and all possible factors or divisors of that number are listed from largest to smallest. The list of factors includes the number itself and the number 1. To return to the menu screen, the student enters zero.

Prime factors. Finding the prime factors is also called *complete factorization* or the *prime factor tree.* The student enters a number, and the prime factors of that number are listed from smallest to largest. The student's answer does not have to list the factors in exact order to be correct. If only the prime factors are desired, the student would still choose this option of the program, and the answer would consist of the list of factors not including duplicated numbers.

For example, all factors of 12 would be 12, 6, 4, 3, 2, and 1; prime factors of 12 would be 2, 2, and 3. The prime factors, without duplication, would be 2 and 3.

Greatest common factor. The student enters two numbers. The program lists the greatest common factor, which is the largest number that can be divided evenly into both the input numbers. If both numbers are prime or if they have no common factors, then the greatest common factor is 1.

Least common multiple. The student first presses 0, 2, or 3. A zero will return the program to the menu screen. A two or three indicates that the student will input either two or three numbers. (This is adequate for fifth- or sixth-grade mathematics.) The program will list the least common multiple, or the lowest number that all the given numbers may be divided into without remainders. For example, the least common multiple of 4 and 12 is 12. The least common multiple of 5, 7, and 2 is 70.

How "Homework Helper: Factors" Works

Lines	
130-210	Clear screen and print title.
220-250	FOR-NEXT loop blinks colors 20 times.
260-300	Define graphics characters and colors.
310-420	Print menu screen of options.
430-470	Receive student's option, clear screen, and branch appropriately.
480-550	PRINT option "Finding all the Factors" and draw graphics.
560-660	Receive student's given number and test that it is larger than 1 and less than 1000.
670-760	Print all factors of the number.
770-790	Wait for student to press a key to continue.
800-860	Print option "Finding the Prime Factors" and draw a sample factor tree.
870-970	Receive student's given number and test that it is larger than 1 and less than 10000.
980-1080	Print prime factors of given number.
1090-1110	Wait for student to press a key to continue.
1120-1220	PRINT option "Greatest Common Factor" and draw graphics.
1230-1400	Receive student's given numbers and test that they are larger than 1 and less than 10000.
1410-1560	Calculate and print greatest common factor.
1570-1590	Wait for student to press a key to continue.
1600-1700	PRINT option "Least Common Multiple" and draw graphics.
1710-1750	Receive student's option for number of given numbers.
1760-1850	Receive student's given numbers and test that they are larger than 1 and less than 1000.
1860-2000	Calculate least common multiple for two numbers.
2010	Print least common multiple.
2020-2040	Wait for student to press a key to continue.
2050-2140	For three numbers, arrange numbers from smallest to largest.
2150-2270	Calculate least common multiple for three numbers.
2280-2310	Subroutine for warning message for large numbers.

Program 4-1. Homework Helper: Factors

```
100 REM  ***********
110 REM  * FACTORS *
120 REM  ***********
130 CALL CLEAR
140 CALL CHAR(64,"3C4299A1A1994237")
150 PRINT TAB(7);"H O M E W O R K"
160 PRINT ::TAB(9);"H E L P E R"
170 CALL COLOR(2,9,16)
180 PRINT ::::TAB(9);"***********"
190 PRINT TAB(9);"* FACTORS *"
200 PRINT TAB(9);"***********"
210 PRINT ::::::::
220 FOR I=1 TO 20
230 CALL COLOR(2,16,9)
240 CALL COLOR(2,9,16)
250 NEXT I
260 CALL CHAR(60,"0102040810204080")
270 CALL CHAR(62,"8040201008040201")
280 CALL CHAR(96,"FFFFFFFFFFFFFFFF")
290 CALL CHAR(97,"0")
300 CALL COLOR(9,5,9)
310 CALL CLEAR
320 CALL COLOR(2,16,9)
330 PRINT "CHOOSE:"
340 PRINT ::"1  ALL FACTORS"
350 PRINT ::"2  PRIME FACTORS"
360 PRINT ::"3  GREATEST COMMON FACTOR"
370 PRINT ::"4  LEAST COMMON MULTIPLE"
380 PRINT ::"5  END PROGRAM"::::
390 CALL HCHAR(1,1,42,32)
400 CALL VCHAR(2,32,42,22)
410 CALL VCHAR(2,1,42,22)
420 CALL HCHAR(24,1,42,32)
430 CALL KEY(0,K,S)
440 IF (K<49)+(K>53)THEN 430
450 CALL CLEAR
460 CALL COLOR(2,2,1)
470 ON K-48 GOTO 480,800,1120,1600,2310
480 PRINT :::"  FINDING ALL THE FACTORS":::
    ::::::::
```

```
490 CALL VCHAR(17,9,96,5)
500 CALL VCHAR(17,10,96,5)
510 CALL VCHAR(17,11,96,5)
520 CALL HCHAR(19,14,61)
530 CALL VCHAR(17,17,96,5)
540 CALL HCHAR(19,20,42)
550 CALL HCHAR(19,23,96,3)
560 PRINT ::"ENTER '0' TO STOP"::
570 INPUT "WHAT IS THE NUMBER? ":N
580 IF N=0 THEN 310
590 IF N>1 THEN 620
600 PRINT :"PLEASE ENTER A NUMBER":"LARGER
    THAN 1."::
610 GOTO 570
620 IF N<1000 THEN 670
630 GOSUB 2280
640 CALL KEY(0,K,S)
650 IF K=78 THEN 480
660 IF K<>89 THEN 640
670 PRINT ::"ALL THE FACTORS OF";N;"ARE:"::
    N;
680 L2=INT(N/2+1)
690 FOR TRY=2 TO L2
700 IF N/TRY<>INT(N/TRY)THEN 750
710 L2=N/TRY
720 PRINT L2;
730 IF L2=1 THEN 770
740 IF L2=2 THEN 760
750 NEXT TRY
760 PRINT " 1"
770 PRINT ::"PRESS ANY KEY TO CONTINUE."
780 CALL KEY(0,K,S)
790 IF S<1 THEN 780 ELSE 480
800 PRINT :::"FINDING THE PRIME FACTORS":::
    :
810 CALL HCHAR(22,14,96,4)
820 CALL HCHAR(23,14,96,4)
830 PRINT TAB(13);"<>":TAB(12);"<  >":TAB(1
    1);"<{4 SPACES}>"
840 PRINT TAB(10);"<>{5 SPACES}>":TAB(9);"<
      >{5 SPACES}>"
850 PRINT TAB(8);"<{3 SPACES}<>{4 SPACES}<>
```

```
    ":TAB(7);"<{3 SPACES}<  >  <  >"
860 PRINT TAB(7);CHR$(96)&"{3 SPACES}"&CHR$
    (96)&"  "&CHR$(96)&"  "&CHR$(96)&"  "&C
    HR$(96)::::
870 PRINT ::"ENTER 'Ø' TO STOP."::
880 INPUT "WHAT IS THE NUMBER? ":N
890 IF N=Ø THEN 31Ø
900 IF N>1 THEN 93Ø
910 PRINT :"PLEASE ENTER A NUMBER":"LARGER
    THAN 1."::
920 GOTO 88Ø
930 IF N<1ØØØØ THEN 98Ø
940 GOSUB 228Ø
950 CALL KEY(Ø,K,S)
960 IF K=78 THEN 8ØØ
970 IF K<>89 THEN 95Ø
980 PRINT ::"THE PRIME FACTORS OF";N;"ARE:"
    ::
990 L2=INT(N/2)
1000 FOR TRY=2 TO L2
1010 IF N/TRY<>INT(N/TRY)THEN 1Ø6Ø
1020 N=N/TRY
1030 L2=N
1040 PRINT TRY;
1050 GOTO 1ØØØ
1060 NEXT TRY
1070 IF N=1 THEN 1Ø9Ø
1080 PRINT N
1090 PRINT ::"PRESS ANY KEY TO CONTINUE."
1100 CALL KEY(Ø,K,S)
1110 IF S<1 THEN 11ØØ ELSE 8ØØ
1120 PRINT ::::"  GREATEST COMMON FACTOR"
1130 PRINT :"  OF TWO NUMBERS":::::::
1140 CALL VCHAR(18,7,96,3)
1150 CALL VCHAR(18,8,96,3)
1160 CALL VCHAR(18,9,97,3)
1170 CALL HCHAR(18,13,96,3)
1180 CALL HCHAR(19,13,97,3)
1190 CALL HCHAR(2Ø,13,96,3)
1200 CALL HCHAR(21,13,96,3)
1210 CALL HCHAR(19,18,46,3)
1220 CALL VCHAR(18,23,97,3)
```

```
1230 PRINT :"ENTER 'Ø' TO STOP.":::
1240 INPUT "FIRST NUMBER =  ":A
1250 IF A=Ø THEN 31Ø
1260 IF A>1 THEN 129Ø
1270 PRINT :"SORRY, PLEASE ENTER NUMBERS":"
     LARGER THAN 1."::
1280 GOTO 124Ø
1290 IF A<1ØØØØ THEN 133Ø
1300 PRINT :"SORRY,":"MUST BE LESS THAN 1ØØ
     ØØ.":"TRY AGAIN."::
1310 GOTO 124Ø
1320 PRINT
1330 INPUT "SECOND NUMBER = ":B
1340 IF B=Ø THEN 31Ø
1350 IF B>1 THEN 138Ø
1360 PRINT :"SORRY, PLEASE ENTER A NUMBERLA
     RGER THAN 1."::
1370 GOTO 133Ø
1380 IF B<1ØØØ THEN 141Ø
1390 PRINT :"SORRY,":"MUST BE LESS THAN 1ØØ
     ØØ.":"TRY AGAIN."::
1400 GOTO 133Ø
1410 PRINT ::"GREATEST COMMON FACTOR IS"::
1420 IF A=B THEN 155Ø
1430 IF A<B THEN 147Ø
1440 D=A
1450 A=B
1460 B=D
1470 FOR TRY=1 TO A
1480 IF (A/TRY)<>INT(A/TRY)THEN 153Ø
1490 L2=A/TRY
1500 IF B/L2<>INT(B/L2)THEN 153Ø
1510 GCF=L2
1520 GOTO 156Ø
1530 NEXT TRY
1540 GCF=1
1550 GCF=A
1560 PRINT GCF
1570 PRINT ::"PRESS ANY KEY TO CONTINUE."
1580 CALL KEY(Ø,K,S)
1590 IF S<1 THEN 158Ø ELSE 112Ø
```

```
1600 PRINT :::"{3 SPACES}LEAST COMMON MULTI
     PLE"::"{4 SPACES}OF 2 OR 3 NUMBERS":::
     ::::::
1610 CALL VCHAR(18,7,96,3)
1620 CALL VCHAR(18,8,96,3)
1630 CALL VCHAR(18,11,96,4)
1640 CALL VCHAR(18,12,96,4)
1650 CALL VCHAR(18,13,96,4)
1660 CALL HCHAR(19,16,46,3)
1670 CALL HCHAR(18,21,96,6)
1680 CALL HCHAR(19,21,96,6)
1690 CALL HCHAR(20,21,96,6)
1700 CALL HCHAR(21,21,96,6)
1710 PRINT "HOW MANY NUMBERS ARE GIVEN--":"
     PRESS 0, 2, OR 3.":::
1720 CALL KEY(0,K,S)
1730 IF K=48 THEN 310
1740 IF (K<50)+(K>51)THEN 1720
1750 CALL HCHAR(21,22,K)
1760 FOR I=1 TO K-48
1770 PRINT "NUMBER";I;":  ";
1780 INPUT M(I)
1790 IF M(I)>1 THEN 1820
1800 PRINT :"SORRY, NUMBER MUST BE":"GREATE
     R THAN 1.":"TRY AGAIN."::
1810 GOTO 1780
1820 IF M(I)<1000 THEN 1850
1830 PRINT :"SORRY, NUMBER MUST BE":"LESS T
     HAN 1000.  TRY AGAIN."::
1840 GOTO 1780
1850 NEXT I
1860 I=K-48
1870 IF I=3 THEN 2050
1880 IF M(1)<>M(2)THEN 1910
1890 LCM=M(1)
1900 GOTO 2010
1910 IF M(1)<M(2)THEN 1950
1920 D=M(1)
1930 M(1)=M(2)
1940 M(2)=D
1950 FOR J=1 TO M(1)
```

```
1960 IF J*M(2)/M(1)=INT(J*M(2)/M(1))THEN 20
     00
1970 NEXT J
1980 LCM=M(1)*M(2)
1990 GOTO 2010
2000 LCM=J*M(2)
2010 PRINT :"LEAST COMMON MULTIPLE IS"::LCM
2020 PRINT ::"PRESS ANY KEY TO CONTINUE."
2030 CALL KEY(0,K,S)
2040 IF S<1 THEN 2030 ELSE 1600
2050 IF (M(1)=M(2))+(M(2)=M(3))=-2 THEN 189
     0
2060 SW=0
2070 FOR J=1 TO 2
2080 IF M(J)<=M(J+1)THEN 2130
2090 D=M(J)
2100 M(J)=M(J+1)
2110 M(J+1)=D
2120 SW=1
2130 NEXT J
2140 IF SW=1 THEN 2060
2150 FOR J=1 TO M(2)
2160 TRY=J*M(3)
2170 IF (TRY/M(1)=INT(TRY/M(1)))+(TRY/M(2)=
     INT(TRY/M(2)))=-2 THEN 2260
2180 NEXT J
2190 LCM1=M(2)*M(3)
2200 FOR J=1 TO M(1)
2210 TRY=J*LCM1
2220 IF TRY/M(1)=INT(TRY/M(1))THEN 2260
2230 NEXT J
2240 LCM=LCM1*M(1)
2250 GOTO 2010
2260 LCM=TRY
2270 GOTO 2010
2280 PRINT :"ARE YOU SURE?":"IT TAKES LONGE
     R TO DO":"LARGE NUMBERS."
2290 PRINT :"IF YOU STILL WANT THIS":"NUMBE
     R PRESS 'Y'":"OR PRESS 'N' FOR NO."
2300 RETURN
2310 END
```

Subroutines

A subroutine and the GOSUB command are used when a process is performed several times. Rather than enter identical lines of code several places in the program, you may put the process in a subroutine, and then use GOSUB to perform the routine each time you want it.

GOSUB is similar to GOTO. It is followed by a line number, and when the program comes to the GOSUB statement it will branch to the line number, just as it does with GOTO. However, with GOSUB the computer will now remember where it branched *from*. When it comes to the command RETURN, it will branch back to the first line after the GOSUB statement. GOSUB works like a boomerang — it goes where you want it to, but it always comes back.

Be careful to make sure that every GOSUB is matched with a RETURN — and that your program never runs into a RETURN without having first executed the corresponding GOSUB. Whenever the computer encounters a RETURN statement, it branches back to the last GOSUB it executed, whether that was the GOSUB you had in mind or not. The advantage of this is that you can have many different GOSUBs branching to the same subroutine. As long as the subroutine ends with RETURN, the computer will always go back where it came from.

A GOSUB Demonstration

The following sample program illustrates the use of subroutines. Lines 410 to 460 are a subroutine to play a little music. Everywhere in the program that you see the statement GOSUB 410, the music will be played. Lines 340-400 are a subroutine to draw a yellow circle starting in row X and column Y.

Lines 130 to 170 define graphics characters for the circle, and line 180 makes the characters yellow. Lines 190-220 draw a red box. Lines 230-250 draw a yellow wheel at row 21 and column 8. The subroutine is called by line 250.

Line 260 changes the column coordinate to 21, and GOSUB 340 in line 270 draws the yellow wheel on the same row, but in a different column. Lines 290-300 set different coordinates, and GOSUB 340 in line 310 draws the yellow circle for the sun higher in the picture. Notice that even though the same subroutine draws the yellow wheel every time, by changing the variables I control *where* the subroutine puts the wheel each time.

Line 330 branches to line 330 to hold the picture on the screen. Press CLEAR to stop the program.

Program 4-2. GOSUB Demonstration

```
100 REM  GOSUB
110 CALL CLEAR
120 GOSUB 410
130 CALL CHAR(96,"FFFFFFFFFFFFFFFF")
140 CALL CHAR(97,"00030F1F3F3F7F7F")
150 CALL CHAR(98,"00C0F0F8FCFCFEFE")
160 CALL CHAR(99,"7F7F3F3F1F0F03")
170 CALL CHAR(100,"FEFEFCFCF8F0C")
180 CALL COLOR(9,12,1)
190 CALL COLOR(10,9,9)
200 CALL HCHAR(18,8,104,16)
210 CALL HCHAR(19,8,104,16)
220 CALL HCHAR(20,8,104,16)
230 X=21
240 Y=8
250 GOSUB 340
260 Y=21
270 GOSUB 340
280 GOSUB 410
290 X=4
300 Y=27
310 GOSUB 340
320 GOSUB 410
330 GOTO 330
340 CALL HCHAR(X,Y,97)
350 CALL VCHAR(X,Y+1,96,3)
360 CALL HCHAR(X,Y+2,98)
370 CALL HCHAR(X+1,Y,96,3)
380 CALL HCHAR(X+2,Y,99)
390 CALL HCHAR(X+2,Y+2,100)
400 RETURN
410 CALL SOUND(150,262,2)
420 CALL SOUND(150,330,2)
430 CALL SOUND(150,392,2)
```

```
440 CALL SOUND(150,330,2)
450 CALL SOUND(150,262,2)
460 RETURN
470 END
```

The subroutines may be placed anywhere in the program — just make sure the computer can get to the subroutine *only* from GOSUB statements. Running into an unexpected RETURN can lead to unpredictable branching or a program crash.

You can avoid this problem by putting a GOTO statement just before the subroutine that will force the program to branch around the subroutine. Or put all your subroutines at the end of the program, right after a STOP statement. Some BASICs will execute programs faster if the subroutines are at the beginning of the program; however, in numerous stopwatch tests of the TI, I haven't noticed a difference dependent on placement of subroutines.

Conditional GOSUBs

The ON-GOSUB statement works just like the ON-GOTO statement, except that the computer will come back when it reaches RETURN.

```
100 REM ON-GOSUB
110 CALL CLEAR
120 PRINT "CHOOSE:"
130 PRINT :"1 GAME 1"
140 PRINT :"2 GAME 2"
150 PRINT :"3 GAME 3"
160 PRINT :"4 END PROGRAM"
170 CALL KEY (0,K,S)
180 IF (K<49)+(K>52)THEN 170
190 CALL CLEAR
200 ON K-48 GOSUB 1000,2000,3000,4000
210 PRINT ::"PRESS ANY KEY"
220 CALL KEY(0,K,S)
230 IF S=1 THEN 110 ELSE 220
240 STOP
1000 PRINT :::"YOU CHOSE GAME 1"
1010 RETURN
2000 PRINT :::"YOU CHOSE GAME 2"
```

```
2010 RETURN
3000 PRINT :::"YOU CHOSE GAME 3"
3010 RETURN
4000 END
```

A Game with GOSUB

This program illustrates the ON-GOSUB statement. Five dice are drawn. For each die, the number of dots (*D*) is chosen randomly, from one to six. Depending on *D*, the computer draws the correct number of dots on the screen by going to the correct subroutine to draw the dots.

Program 4-3. Dice Throw

```
1ØØ REM  DICE
11Ø REM  BY REGENA
12Ø CALL CLEAR
13Ø PRINT "SAMPLE DICE THROW"::::::::::::::
14Ø CALL CHAR(96,"ØØØØ183C3C18")
15Ø CALL COLOR(9,7,6)
16Ø CALL CHAR(1Ø4,"ØØØØØØØØØØ7Ø4Ø4")
17Ø CALL CHAR(1Ø5,"ØØØØØØØØØØFF")
18Ø CALL CHAR(1Ø6,"ØØØØØØØØØØEØ2Ø2")
19Ø CALL CHAR(1Ø7,"2Ø2Ø2Ø2Ø2Ø2Ø2Ø2")
2ØØ CALL CHAR(1Ø8,"2Ø2ØE")
21Ø CALL CHAR(1Ø9,"ØØØØFF")
22Ø CALL CHAR(11Ø,"Ø4Ø4Ø7")
23Ø CALL CHAR(111,"Ø4Ø4Ø4Ø4Ø4Ø4Ø4Ø4")
24Ø CALL COLOR(1Ø,2,6)
25Ø CALL COLOR(16,6,6)
26Ø R=12
27Ø FOR C=2 TO 3Ø STEP 6
28Ø RANDOMIZE
29Ø FRE=INT(11ØØ*RND)+44Ø
3ØØ CALL SOUND(3ØØ,FRE,2)
31Ø CALL HCHAR(R,C,1Ø4)
32Ø CALL HCHAR(R,C+1,1Ø5,3)
33Ø CALL HCHAR(R,C+4,1Ø6)
34Ø CALL VCHAR(R+1,C+4,1Ø7,3)
35Ø CALL VCHAR(R+4,C+4,1Ø8)
36Ø CALL HCHAR(R+4,C+1,1Ø9,3)
37Ø CALL HCHAR(R+4,C,11Ø)
```

```
380 CALL VCHAR(R+1,C,111,3)
390 FOR I=R+1 TO R+3
400 CALL HCHAR(I,C+1,157,3)
410 NEXT I
420 NEXT C
430 REM  PRINT DOTS
440 FOR N=1 TO 5
450 RANDOMIZE
460 D=INT(6*RND+1)
470 J=2+6*(N-1)
480 ON D GOSUB 570,600,640,690,750,790
490 NEXT N
500 PRINT "TRY AGAIN? (Y/N)"
510 CALL KEY(0,K,S)
520 IF K=78 THEN 850
530 IF K<>89 THEN 510
540 CALL CLEAR
550 GOTO 270
560 REM  ONE
570 CALL HCHAR(R+2,J+2,96)
580 RETURN
590 REM  TWO
600 CALL HCHAR(R+1,J+1,96)
610 CALL HCHAR(R+3,J+3,96)
620 RETURN
630 REM  THREE
640 FOR I=1 TO 3
650 CALL HCHAR(R+I,J+I,96)
660 NEXT I
670 RETURN
680 REM  FOUR
690 CALL HCHAR(R+1,J+1,96)
700 CALL HCHAR(R+1,J+3,96)
710 CALL HCHAR(R+3,J+1,96)
720 CALL HCHAR(R+3,J+3,96)
730 RETURN
740 REM  FIVE
750 GOSUB 690
760 CALL HCHAR(R+2,J+2,96)
770 RETURN
780 REM  SIX
790 FOR I=1 TO 3
```

```
800 FOR JJ=1 TO 3 STEP 2
810 CALL HCHAR(R+I,J+JJ,96)
820 NEXT JJ
830 NEXT I
840 RETURN
850 CALL CLEAR
860 END
```

Nested Subroutines

Subroutines can be nested. That is, a second GOSUB can be executed before RETURNing from the first. Remember that the computer always RETURNs to the most *recent* GOSUB.

Program 4-4 consists of four main sections:

Plotting points. A rectangular coordinate system is printed with a random point. The point is defined by its x-coordinate and y-coordinate. If you press *Y* for another example, a different point may be chosen with the coordinates labeled. If you press *N*, the screen is cleared. This time a point is shown, and you must press the numbers for the coordinates. If your answer is incorrect, you will be shown the correct answer and given another problem. If your answer is correct, you have the option to choose another problem of the same type or to continue the program.

The next part gives the coordinates, and you must locate the point. You move the point by pressing the arrow keys. When your point is at the desired position, press ENTER. If your point is incorrect, the correct answer is shown, and you will be given another problem. If your point is correct, you have the option of choosing another problem of the same type or continuing the program.

Positive and negative coordinates. This section is just like the first section, except that you may have positive and negative coordinates.

Slope of a line. Given two points on a line, you can find the slope of the line by calculating the ratio of the difference between the two y-coordinates and the difference between the two x-coordinates. After some instruction, you are given a quiz.

Distance between points. This section teaches you how to find the distance between two given points on a graph, using the Pythagorean theorem. A problem is also presented.

If you are using this program for the first time, it would be best to choose the options in numerical order.

Since this program is a tutorial program, I have tried to make it as user-friendly as possible. Whenever one key-press is required, a CALL KEY statement is used rather than an INPUT procedure. Any time INPUT is used, there is a greater possibility of the program "crashing." In this program, all INPUT prompts require that numbers be entered. Whenever an answer is incorrect, the correct answer is given and another problem of the same type is presented.

After each correct answer, the student has the choice of doing another problem of the same type or continuing the program.

How "Coordinate Geometry" Works

Lines	
100	DEFine a function R(N) to be a random number from 1 to N.
110-130	Clear the screen and print the title.
140-350	Define the graphics characters and colors.
360-380	Define strings to be printed for graphics.
390-410	Print the menu screen of options.
420-460	Receive the student's option and branch appropriately with an ON-GOSUB statement.
470-480	After a section is complete, clear the screen and return to line 390 to print the main options.
490-510	Subroutine to play music for incorrect answer.
520-560	Subroutine to play music for correct answer.
570-610	Subroutine to print coordinate system.
620-660	Subroutine to PRINT "PRESS ENTER " and wait for the student to press ENTER key.
670-710	Subroutine to draw graphics. For N number of characters, draw character number C at row A and column B.
720-1840	Subroutine for main option 1, *Plotting Points.*
720-830	Draw a coordinate system, plot a random example point, and illustrate the coordinates.
840-870	Print option for another example, branch appropriately.
880-910	Print the instructions.
920-1110	Plot a random point and wait for the student to press coordinate numbers.

1120-1130	Test for the correct answer.
1140-1180	For a correct answer, play music and print the next option; branch appropriately.
1190-1240	For an incorrect answer, play music, show the correct answer, and branch to line 920 for another problem.
1250-1280	Print instructions.
1290-1410	Choose random coordinates for problem and initialize coordinates for the point.
1420-1690	Move the point depending on the arrow key pressed.
1700-1720	Sound a beep, then test to see if the point has been placed correctly.
1730-1780	If the point is incorrect, play music, show the correct answer, and branch to line 1290 for another problem.
1790-1840	If the point is correct, play music, print the next option, branch appropriately, and return to the main menu screen.
1850-1900	Subroutine to plot a random point on graph.
1910-1940	Subroutine to show coordinate for x-value.
1950-1980	Subroutine to show coordinate for y-value.
1990-2040	Subroutine to draw the coordinate system for positive and negative coordinates.
2050-2970	Subroutine for main option 2, *Positive and Negative Coordinates.*
2050-2140	Draw four example points on the coordinate system.
2150-2220	Draw a random point and ask for coordinates.
2230-2240	Receive the student's INPUT coordinates.
2250-2260	Test to see if the answer is correct.
2270-2310	For a correct answer, play music, print the next option, and branch appropriately.
2320-2380	For an incorrect answer, play music, give the correct answer, and branch to line 2150 for another point.
2390-2420	Print instructions.
2430-2550	Choose a random point for the student to plot.
2560-2830	Move the point depending on the arrow key pressed.
2840-2860	Sound a beep, then test if point has been placed correctly.

2870-2900	For an incorrect answer, play music, plot the correct point, and branch to line 2430 for another point.
2910-2970	For a correct answer, play music, print the next option, and branch appropriately. Return to the main menu screen.
2980-3860	Subroutine for main option 4, *Distance between Points.*
2980-3080	Print review information and illustration.
3090-3210	Subroutine to draw the illustration for distance.
3220-3480	Presents the instruction for distance.
3490-3510	Clear the screen; randomly choose one of six problems.
3520-3690	RESTORE proper DATA and READ numbers A, B, and C, which are numbers in a Pythagorean triple.
3700-3750	Print the problem, with random coordinates for the first point, and coordinates depending on the DATA for the second point.
3760-3770	Student INPUTs an answer; test for the correct distance.
3780-3800	For an incorrect answer, play music, show the correct answer, and branch to line 3480 for another problem.
3810-3860	For a correct answer, play music, print the next option, and branch appropriately. Return to the main menu screen.
3870-4640	Subroutine for main option 3, *Slope of a Line between Two Points.*
3870-4090	Present the instruction for slope.
4100-4220	Randomly choose two points and ask for the difference in y-coordinates.
4230-4270	Print the key that the student presses for delta-y.
4280-4350	Ask the student for delta-x and print the response.
4360-4370	Ask the student for slope and receive the student's INPUT.
4380-4390	Calculate the correct answer and compare it with the student's answer.
4400-4420	For a correct answer, play music, pause, and branch to line 4460.

4430-4450	For an incorrect answer, play music, give the correct slope, and branch to line 4090 for another problem.
4460-4580	Present another problem for slope without the intermediate steps.
4590-4640	For a correct answer, play music, print the next option, and branch appropriately. Return to main menu screen.
4650	End.

Program 4-4. Coordinate Geometry

```
100 DEF R(N)=INT(N*RND+1)
110 CALL CLEAR
120 PRINT "  ************************
    ":"  *";TAB(25);"*":"  * COORDIN
    ATE GEOMETRY *"
130 PRINT "  *";TAB(25);"*":"  *****
    *******************":::TAB(11);"
    POINTS":::
140 A$="1818181818181818"
150 B$="181818FFFF181818"
160 C$="000000FFFF"
170 FOR C=96 TO 112 STEP 8
180 CALL CHAR(C,A$)
190 CALL CHAR(C+1,B$)
200 CALL CHAR(C+2,C$)
210 NEXT C
220 CALL CHAR(120,"183C7EFFFF7E3C18"
    )
230 CALL CHAR(128,"183C7EFFFF7E3C18"
    )
240 CALL CHAR(129,"000000000030C30C")
250 CALL CHAR(130,"030C30C")
260 CALL CHAR(64,"3C4299A1A199423C")
270 CALL CHAR(94,"0010282844448 2FE")
280 CALL COLOR(10,5,1)
290 CALL COLOR(11,10,1)
300 CALL COLOR(12,11,1)
310 CALL COLOR(13,7,1)
320 CALL CHAR(140,"1010101010101010")
```

```
330 CALL CHAR(141,"000000FF")
340 CALL CHAR(142,"101010F")
350 CALL COLOR(14,13,1)
360 A$="`  h  h  h  h  h  h  h"
370 B$="ajjijjijjijjijjijjijji"
380 C$="abbabbabbabbabbabbabbabb"
390 PRINT "CHOOSE:"::"1 PLOTTING POI
    NTS":"2 + AND - COORDINATES":"3
    SLOPE OF A LINE"
400 PRINT "4 DISTANCE BETWEEN POINTS
    ":"5 END PROGRAM"
410 PRINT :::::
420 CALL KEY(0,K,S)
430 IF (K<49)+(K>53)THEN 420
440 CALL CLEAR
450 CALL COLOR(2,2,1)
460 ON K-48 GOSUB 720,2050,3870,2980
    ,4650
470 CALL CLEAR
480 GOTO 390
490 CALL SOUND(100,330,2)
500 CALL SOUND(100,262,2)
510 RETURN
520 CALL SOUND(100,262,2)
530 CALL SOUND(100,330,2)
540 CALL SOUND(100,392,2)
550 CALL SOUND(200,523,2)
560 RETURN
570 CALL CLEAR
580 PRINT "{4 SPACES}Y":"{4 SPACES}"
    ;A$:"{4 SPACES}";A$:"{3 SPACES}4
    ";B$:"{4 SPACES}";A$:"
    {4 SPACES}";A$:"{3 SPACES}3";B$:
    "{3 SPACES}";A$:"{4 SPACES}";A$:"
    {3 SPACES}2";B$
590 PRINT "{4 SPACES}";A$:"
    {4 SPACES}";A$:"{3 SPACES}1";B$:
    "{4 SPACES}";A$:"{4 SPACES}";A$:
    "{3 SPACES}0";C$:"{4 SPACES}0  1
    2  3  4  5  6  7":::
600 CALL HCHAR(20,31,88)
```

```
610 RETURN
620 PRINT TAB(16);"PRESS <ENTER>";
630 CALL KEY(Ø,K,S)
640 IF K<>13 THEN 63Ø
650 CALL HCHAR(24,18,32,13)
660 RETURN
670 FOR I=1 TO N
680 READ A,B,C
690 CALL HCHAR(A,B,C)
700 NEXT I
710 RETURN
720 GOSUB 57Ø
730 PRINT "THE LOCATION OF A POINT I
    S":"GIVEN BY ITS X-COORDINATE":"
    AND Y-COORDINATE  (X,Y)"
740 RANDOMIZE
750 X=R(5)
760 GOSUB 186Ø
770 GOSUB 191Ø
780 CALL HCHAR(Y1,X1+2,4Ø)
790 CALL HCHAR(Y1,X1+3,48+X)
800 CALL HCHAR(Y1,X1+4,44)
810 GOSUB 195Ø
820 CALL HCHAR(Y1,X1+5,48+Y)
830 CALL HCHAR(Y1,X1+6,41)
840 PRINT :"WANT ANOTHER EXAMPLE? (Y
    /N)";
850 CALL KEY(Ø,K,S)
860 IF K=89 THEN 72Ø
870 IF K<>78 THEN 85Ø
880 CALL CLEAR
890 PRINT "YOU WILL BE SHOWN A POINT
    .":"PRESS THE NUMBER OF THE"::"
    X-COORDINATE THEN THE"
900 PRINT :"NUMBER OF THE Y-COORDINA
    TE.":::::::
910 GOSUB 62Ø
920 CALL CLEAR
930 GOSUB 57Ø
940 PRINT :::
950 RANDOMIZE
960 GOSUB 185Ø
```

```
970 CALL HCHAR(21,7,40)
980 CALL HCHAR(21,9,44)
990 CALL HCHAR(21,11,41)
1000 CALL KEY(0,K,S)
1010 CALL HCHAR(21,8,63)
1020 CALL HCHAR(21,8,32)
1030 IF S<1 THEN 1000
1040 CALL HCHAR(21,8,K)
1050 X2=K
1060 CALL KEY(0,K,S)
1070 CALL HCHAR(21,10,63)
1080 CALL HCHAR(21,10,32)
1090 IF S<1 THEN 1060
1100 CALL HCHAR(21,10,K)
1110 Y2=K
1120 IF X2<>X+48 THEN 1190
1130 IF Y2<>Y+48 THEN 1190
1140 GOSUB 520
1150 PRINT "PRESS":"1 FOR SAME TYPE
     PROBLEM":"2 TO CONTINUE PROGRAM
     ";
1160 CALL KEY(0,K,S)
1170 IF K=49 THEN 920
1180 IF K=50 THEN 1250 ELSE 1160
1190 GOSUB 490
1200 GOSUB 1910
1210 GOSUB 1950
1220 PRINT "THE CORRECT ANSWER IS ("
     ;STR$(X);",";STR$(Y);")"
1230 GOSUB 620
1240 GOTO 920
1250 CALL CLEAR
1260 PRINT "NOW YOU WILL BE GIVEN TH
     E"::"COORDINATES."::"USE THE AR
     ROW KEYS TO MOVE"::"THE POINT TO
     THE CORRECT"
1270 PRINT :"PLACE, THEN PRESS <ENTE
     R>.":::::
1280 GOSUB 620
1290 CALL CLEAR
1300 GOSUB 570
1310 RANDOMIZE
```

```
1320 X=R(7)
1330 Y=R(4)
1340 X1=7+3*X
1350 Y1=17-3*Y
1360 PRINT :"PLOT (";STR$(X);",";STR
     $(Y);")"::
1370 C1=97
1380 A=17
1390 A1=A
1400 B=7
1410 B1=B
1420 CALL HCHAR(A,B,120)
1430 CALL KEY(0,K,S)
1440 IF S<1 THEN 1430
1450 IF K=13 THEN 1700
1460 IF K<>69 THEN 1510
1470 IF A=5 THEN 1430
1480 CALL GCHAR(A-3,B,C)
1490 A=A-3
1500 GOTO 1650
1510 IF K<>88 THEN 1560
1520 IF A=17 THEN 1430
1530 CALL GCHAR(A+3,B,C)
1540 A=A+3
1550 GOTO 1650
1560 IF K<>83 THEN 1610
1570 IF B=7 THEN 1430
1580 CALL GCHAR(A,B-3,C)
1590 B=B-3
1600 GOTO 1650
1610 IF K<>68 THEN 1430
1620 IF B=28 THEN 1430
1630 CALL GCHAR(A,B+3,C)
1640 B=B+3
1650 CALL HCHAR(A1,B1,C1)
1660 A1=A
1670 B1=B
1680 C1=C
1690 GOTO 1420
1700 CALL SOUND(150,1397,2)
1710 CALL GCHAR(Y1,X1,C)
1720 IF C=120 THEN 1790
```

```
173Ø GOSUB 49Ø
174Ø CALL HCHAR(Y1,X1,128)
175Ø GOSUB 191Ø
176Ø GOSUB 195Ø
177Ø GOSUB 62Ø
178Ø GOTO 129Ø
179Ø GOSUB 52Ø
18ØØ PRINT "PRESS":"1 FOR SAME TYPE
     PROBLEM":"2 TO CONTINUE PROGRAM
     ";
181Ø CALL KEY(Ø,K,S)
182Ø IF K=49 THEN 129Ø
183Ø IF K<>5Ø THEN 181Ø
184Ø RETURN
185Ø X=R(7)
186Ø Y=R(4)
187Ø X1=7+3*X
188Ø Y1=17-3*Y
189Ø CALL HCHAR(Y1,X1,128)
19ØØ RETURN
191Ø FOR I=Y1+1 TO 17
192Ø CALL HCHAR(I,X1,112)
193Ø NEXT I
194Ø RETURN
195Ø FOR I=X1-1 TO 7 STEP -1
196Ø CALL HCHAR(Y1,I,114)
197Ø NEXT I
198Ø RETURN
199Ø CALL CLEAR
2ØØØ PRINT TAB(14);"Y":"jijjijjijjij
     3ajjijjijjijjijj":D$:D$;"jijjij
     jijjij2ajjijjijjijjijj":D$:D$
2Ø1Ø PRINT "jijjijjijjijlajjijjijjij
     jijj":D$:D$:"babbabbabbabØabbab
     babbabbabX"
2Ø2Ø PRINT "-4 -3 -2 -1  Ø  1  2  3
      4":D$:"jijjijjijji-1ajjijjijji
     jjijj":D$:D$:"jijjijjijji-2ajjij
     jijjijjijj"
2Ø3Ø PRINT D$:D$:"jijjijjijji-3ajjij
     jijjijjijj":D$
2Ø4Ø RETURN
```

```
2050 D$=" h  h  h  h  `  h  h  h  h"
2060 GOSUB 2000
2070 PRINT "HERE ARE EXAMPLES PLOTTI
     NG{3 SPACES}+ AND - COORDINATES
     ."
2080 RESTORE 2090
2090 DATA 5,25,128,6,26,40,6,27,51,6
     ,28,44,6,29,50,6,30,41,5,7,128,
     6,5,40,6,6,45,6,7,51,6,8,44
2100 DATA 6,9,50,6,10,41,14,22,128,1
     5,23,40,15,24,50,15,25,44,15,26
     ,45,15,27,49,15,28,41
2110 DATA 17,4,128,18,5,40,18,6,45,1
     8,7,52,18,8,44,18,9,45,18,10,50
     ,18,11,41,18,12,32
2120 N=29
2130 GOSUB 670
2140 GOSUB 620
2150 GOSUB 1990
2160 RANDOMIZE
2170 X=R(9)-5
2180 Y=R(7)-4
2190 X1=3*X+16
2200 Y1=13-3*Y
2210 CALL HCHAR(Y1,X1,128)
2220 PRINT "COORDINATES:"
2230 INPUT "{4 SPACES}X = ":X2$
2240 INPUT "{4 SPACES}Y = ":Y2$
2250 IF STR$(X)<>X2$ THEN 2320
2260 IF STR$(Y)<>Y2$ THEN 2320
2270 GOSUB 520
2280 PRINT :"PRESS":"1 FOR ANOTHER P
     OINT":"2 TO CONTINUE PROGRAM";
2290 CALL KEY(0,K,S)
2300 IF K=49 THEN 2150
2310 IF K=50 THEN 2390 ELSE 2290
2320 GOSUB 490
2330 P$="("&STR$(X)&","&STR$(Y)&")"
2340 FOR I=1 TO LEN(P$)
2350 CALL HCHAR(Y1-2,I+X1-2,ASC(SEG$
     (P$,I,1)))
2360 NEXT I
```

```
2370 GOSUB 620
2380 GOTO 2150
2390 CALL CLEAR
2400 PRINT "PLOT THE GIVEN POINT.":: 
     "USE THE ARROW KEYS TO MOVE"::"
     THE YELLOW SPOT TO THE"
2410 PRINT :"CORRECT POSITION, THEN"
     ::"PRESS <ENTER>.":::::::
2420 GOSUB 620
2430 CALL CLEAR
2440 RANDOMIZE
2450 X=R(9)-5
2460 Y=R(7)-4
2470 X1=16+3*X
2480 Y1=11-3*Y
2490 GOSUB 1990
2500 A1=11
2510 A=A1
2520 B1=16
2530 B=B1
2540 C1=97
2550 PRINT :"PLOT POINT ("&STR$(X)&"
     ,"&STR$(Y)&")"
2560 CALL HCHAR(A,B,120)
2570 CALL KEY(0,K,S)
2580 IF S<1 THEN 2570
2590 IF K=13 THEN 2840
2600 IF K<>69 THEN 2650
2610 IF A=2 THEN 2570
2620 CALL GCHAR(A-3,B,C)
2630 A=A-3
2640 GOTO 2790
2650 IF K<>88 THEN 2700
2660 IF A=20 THEN 2570
2670 CALL GCHAR(A+3,B,C)
2680 A=A+3
2690 GOTO 2790
2700 IF K<>68 THEN 2750
2710 IF B=28 THEN 2570
2720 CALL GCHAR(A,B+3,C)
2730 B=B+3
2740 GOTO 2790
```

```
2750 IF K<>83 THEN 2570
2760 IF B=4 THEN 2570
2770 CALL GCHAR(A,B-3,C)
2780 B=B-3
2790 CALL HCHAR(A1,B1,C1)
2800 A1=A
2810 B1=B
2820 C1=C
2830 GOTO 2560
2840 CALL SOUND(150,1397,2)
2850 CALL GCHAR(Y1,X1,C)
2860 IF C=120 THEN 2910
2870 GOSUB 490
2880 CALL HCHAR(Y1,X1,128)
2890 GOSUB 620
2900 GOTO 2430
2910 GOSUB 520
2920 PRINT :"PRESS":"1 TO PLOT ANOTH
     ER POINT":"2 TO CONTINUE PROGRA
     M";
2930 CALL KEY(0,K,S)
2940 IF K=49 THEN 2430
2950 IF K<>50 THEN 2930
2960 CALL CLEAR
2970 RETURN
2980 CALL SCREEN(8)
2990 CALL CHAR(136,"3D4381818181433D
     ")
3000 CALL CHAR(137,"0000808080808008"
     )
3010 CALL CHAR(138,"BCC2818181C2BC
     ")
3020 CALL CHAR(139,"3C4280808080423C
     ")
3030 PRINT "REVIEW: FIND THE DISTANC
     E":TAB(9);"BETWEEN A AND B.":::
     :::::::::
3040 PRINT "IN A RIGHT TRIANGLE,"::T
     AB(8);"2{3 SPACES}"&CHR$(137)&"
     2{5 SPACES}2"
3050 PRINT TAB(7);CHR$(136);"  + ";C
     HR$(138);"  =  ";CHR$(139)::"OR
```

```
     LENGTH  ";CHR$(139);" = SQUARE ROOT"
3060 PRINT TAB(17);CHR$(137):"OF  ";
     CHR$(136);" SQUARED + ";CHR$(13
     8);" SQUARED.":: 
3070 GOSUB 3090
3080 GOTO 3220
3090 CALL HCHAR(11,8,128)
3100 CALL HCHAR(11,7,65)
3110 CALL HCHAR(5,20,128)
3120 CALL HCHAR(5,21,66)
3130 CALL VCHAR(6,20,104,5)
3140 CALL HCHAR(11,9,106,11)
3150 CALL HCHAR(11,20,105)
3160 RESTORE 3170
3170 DATA 10,9,129,10,10,130,9,11,12
     9,9,12,130,8,13,129,8,14,130,7,
     15,129,7,16,130,6,17,129
3180 DATA 6,18,130,5,19,129,7,13,139
     ,8,22,136,12,14,137,13,14,138,1
     ,1,32
3190 N=16
3200 GOSUB 670
3210 RETURN
3220 GOSUB 620
3230 CALL CLEAR
3240 PRINT "FIND THE DISTANCE ";CHR$
     (139);"BETWEEN POINT 1 AND POIN
     T 2."::::::::::::::
3250 PRINT CHR$(136);" = Y2 - Y1"::C
     HR$(137):CHR$(138);" = X2 - X1"
     :::
3260 CALL CHAR(92,"0102022414180808"
     )
3270 CALL CHAR(95,"00000000000000FF"
     )
3280 CALL CHAR(91,"380418203C")
3290 PRINT CHR$(139);" = \(X2-X1)[ +
      (Y2-Y1)[":: 
3300 CALL HCHAR(21,8,95,19)
3310 GOSUB 3090
3320 M1$="(X1,Y1)"
```

```
3330 M$="(X2,Y2)"
3340 FOR I=1 TO 7
3350 CALL HCHAR(5,20+I,ASC(SEG$(M$,I
     ,1)))
3360 CALL HCHAR(12,2+I,ASC(SEG$(M1$,
     I,1)))
3370 NEXT I
3380 CALL HCHAR(11,7,32)
3390 GOSUB 620
3400 RESTORE 3410
3410 DATA 5,22,53,5,23,44,5,24,52,5,
     25,41,12,5,40,12,6,49,12,7,44,1
     2,8,49,16,16,61,16,18,51,19,16,61
3420 DATA 19,18,52,22,27,61,22,29,53
3430 CALL HCHAR(5,22,32,6)
3440 CALL HCHAR(12,3,32,6)
3450 N=14
3460 GOSUB 670
3470 CALL HCHAR(2,3,32,28)
3480 GOSUB 620
3490 CALL CLEAR
3500 I=R(6)
3510 ON I GOTO 3520,3550,3580,3610,3
     640,3670
3520 RESTORE 3530
3530 DATA 3,4,5
3540 GOTO 3690
3550 RESTORE 3560
3560 DATA 4,3,5
3570 GOTO 3690
3580 RESTORE 3590
3590 DATA 5,12,13
3600 GOTO 3690
3610 RESTORE 3620
3620 DATA 12,5,13
3630 GOTO 3690
3640 RESTORE 3650
3650 DATA 8,15,17
3660 GOTO 3690
3670 RESTORE 3680
3680 DATA 15,8,17
3690 READ A,B,C
```

```
3700 X1=R(5)-1
3710 Y1=R(5)-1
3720 PRINT "POINT 1 = (";STR$(X1);",
     ";STR$(Y1);")"
3730 X2=X1+A
3740 Y2=Y1+B
3750 PRINT :"POINT 2 = (";STR$(X2);"
     ,";STR$(Y2);")":::"WHAT IS THE
     DISTANCE"::"BETWEEN THE POINTS?"::
3760 INPUT C1
3770 IF C=C1 THEN 3810
3780 GOSUB 490
3790 PRINT :CHR$(136);" = ";B:CHR$(1
     37):CHR$(138);" = ";A::"DISTANC
     E "&CHR$(139);" = ";C::
3800 GOTO 3480
3810 GOSUB 520
3820 PRINT :"PRESS":"1 TRY ANOTHER P
     ROBLEM":"2 CONTINUE PROGRAM"
3830 CALL KEY(0,K,S)
3840 IF K=49 THEN 3490
3850 IF K<>50 THEN 3830
3860 RETURN
3870 PRINT "THE SLOPE OF A LINE BETW
     EEN":"TWO POINTS IS DEFINED AS
     THE":"RATIO OF THE CHANGE IN Y TO"
3880 PRINT "THE CHANGE IN X."
3890 GOSUB 580
3900 RESTORE 3910
3910 DATA 17,10,128,11,22,128,16,11,
     129,16,12,130,15,13,129,15,14,1
     30,14,15,129,14,16,130,13,17,129
3920 DATA 13,18,130,12,19,129,12,20,
     130,11,21,129,1,1,32
3930 N=14
3940 GOSUB 670
3950 CALL VCHAR(12,21,140,5)
3960 CALL HCHAR(16,13,141,8)
3970 CALL HCHAR(16,21,142)
3980 GOSUB 620
3990 CALL HCHAR(14,22,94)
```

```
4000 CALL HCHAR(14,23,89)
4010 CALL HCHAR(17,17,94)
4020 CALL HCHAR(17,18,88)
4030 PRINT "^Y = 2":"^X = 4":"SLOPE
     M = 2/4 = 1/2 = .5"
4040 GOSUB 620
4050 CALL CLEAR
4060 PRINT "^Y IS THE DIFFERENCE BET
     WEEN"::"THE Y-COORDINATES."
4070 PRINT ::"^X IS THE DIFFERENCE B
     ETWEEN"::"THE X-COORDINATES."
4080 PRINT :::TAB(11);"^Y":"SLOPE M
     = --":TAB(11);"^X"::::
4090 GOSUB 620
4100 CALL CLEAR
4110 PRINT "GIVEN TWO POINTS:"
4120 RANDOMIZE
4130 X1=R(3)-1
4140 X2=R(8)-1
4150 IF X2<=X1 THEN 4140
4160 Y1=R(3)-1
4170 Y2=R(5)-1
4180 IF Y2<=Y1 THEN 4170
4190 PRINT :"("&STR$(X1)&","&STR$(Y1
     )&"){3 SPACES}AND{3 SPACES}("&S
     TR$(X2)&","&STR$(Y2)&")":::
4200 PRINT "^Y =";
4210 DY=Y2-Y1
4220 CALL SOUND(150,1397,2)
4230 CALL KEY(0,K,S)
4240 CALL HCHAR(24,8,63)
4250 CALL HCHAR(24,8,32)
4260 IF K<>DY+48 THEN 4230
4270 PRINT DY
4280 PRINT :"^X =";
4290 DX=X2-X1
4300 CALL SOUND(150,1397,2)
4310 CALL KEY(0,K1,S)
4320 CALL HCHAR(24,8,63)
4330 CALL HCHAR(24,8,32)
4340 IF K1<>DX+48 THEN 4310
4350 PRINT DX
```

```
4360 PRINT :::"SLOPE M = ^Y/^X"::"EX
     PRESS M AS A DECIMAL.":::
4370 INPUT "M = ":M
4380 M1=DY/DX
4390 IF ABS(M-M1)>.005 THEN 4430
4400 GOSUB 520
4410 GOSUB 620
4420 GOTO 4460
4430 GOSUB 490
4440 PRINT :"THE CORRECT SLOPE IS";I
     NT(100*(M1+.005))/100::
4450 GOTO 4090
4460 CALL CLEAR
4470 PRINT :"GIVEN TWO POINTS:"
4480 X1=R(3)-1
4490 X2=R(8)-1
4500 IF X2<=X1 THEN 4490
4510 Y1=R(2)-1
4520 Y2=R(5)-1
4530 IF Y2<=Y1 THEN 4520
4540 PRINT :"("&STR$(X1)&","&STR$(Y1
     )&"){3 SPACES}AND{3 SPACES}("&S
     TR$(X2)&","&STR$(Y2)&")":::
4550 PRINT "WHAT IS THE SLOPE M?":::
4560 M1=(Y2-Y1)/(X2-X1)
4570 INPUT "M = ":M
4580 IF ABS(M-M1)>.005 THEN 4430
4590 GOSUB 520
4600 PRINT :"PRESS":"1 FOR SAME TYPE
      PROBLEM":"2 TO CONTINUE PROGRA
     M";
4610 CALL KEY(0,K,S)
4620 IF K=49 THEN 4460
4630 IF K<>50 THEN 4610
4640 RETURN
4650 END
```

Chapter 5

Built-in Functions

Built-in Functions

Commands, like RUN, GOSUB, ON, and LET, stand at the beginning of a statement and control everything else that happens in the line. Functions, on the other hand, are like small subroutines within a statement. They can never stand alone, and always return a value or a string.

You have been using some numeric functions all along. The symbols +, –, /, *, and ^ all require the computer to leave the current command, perform an operation, and return with a value. In the command line LET A=3, the computer simply follows the command LET. But in the command line LET A=3*C, the computer has to perform the function of multiplying 3 by the value of C before carrying out the command LET.

Functions cannot stand alone. Your computer does not know what to do with a statement like 4+4. Not until the statement includes a command can the computer tell what you want it to do: PRINT 4+4. Now the computer knows that it is required to PRINT, and performs the function + on the way to carrying out the command.

Mathematical Functions

TI BASIC can do difficult calculations just like big computers, and it has many of the functions that are on the more expensive calculators.

The mathematical functions, aside from the one-symbol arithmetic operations already mentioned, consist of three-letter abbreviations and an argument or numeric expression in parentheses. The name of the function is a reserved word, which means you cannot use it alone as a variable name in your programs. However, the word can be embedded within a variable name.

ABS

ABS(*x*) gives the absolute value of a numeric expression or number *x*. The absolute value of a number is the number itself without a negative sign.

Command	Result
PRINT ABS(-3)	3
PRINT ABS(3)	3
PRINT ABS(0)	0

ATN

ATN(*x*) gives the arctangent of the expression *x*. The arctangent of *x* is the angle whose tangent is *x*. The value is in radians; if you want the equivalent angle in degrees, multiply your answer by 180/pi or 180/(4*ATN(1)) or 57.295779513079.

Here is a short program that illustrates ATN (*x*). Values for *x* are read in as numbers from DATA. The angle whose tangent is *x* is printed first in radians, then in degrees.

```
100 REM ATN
110 CALL CLEAR
120 PRINT " X","ATN(X)"
130 FOR C=1 TO 10
140 READ X
150 PRINT :X,"R ";ATN(X)
160 PRINT TAB(15);"D ";ATN(X)*(180/(4*ATN(
    1)))
170 NEXT C
180 DATA .10,.22,.44,.50,1,0
190 DATA -.33,-10,-50,1E35
200 END
```

The value for pi can be obtained by the command:

PRINT 4*ATN(1)

It is given as 3.141592654.

COS

COS(*x*) gives the cosine of an angle *x*, where the angle is expressed in radians. If your angle is in degrees, you may convert by multiplying by pi/180 or (4*ATN(1))/180 or 0.01745329251994.

EXP

EXP(x) gives the exponential function, or the value of e^x, where e is approximately 2.718281828.

INT

INT(x) gives the integer function of a number x, which is the whole number part of the number x if x is positive, and the next smaller whole number if the number x is negative. Another way to think of the integer function is that the result is the closest integer (whole number) which is to the left of the number x on a number line. The value returned by INT(3.5) is 3, and INT(-7.4) returns -8.

LOG

LOG(x) gives the natural logarithm of x, or $\log_e(x)$. Remember that the argument or expression x must be greater than zero. The logarithm function is the inverse of exponential function, so

X=LOG(EXP(X))   and   X=EXP(LOG(X))

Here are formulas to keep in mind for logarithms:
If you want to find the logarithm of a number in base N:

$$\log_N(X)=\log_e(X)/\log_e(N)$$

Probably the most common base you would need is base 10:

$$\log_{10}(X)=\log_e(X)/\log_e(10)$$

SGN

SGN(x) gives the sign of a number x. If x is negative, SGN(x) is equal to -1. If x is positive, SGN(x) is equal to 1. If x is zero, SGN(x) is equal to 0.

SIN

SIN(x) gives the sine of an angle x, where x is expressed in radians. Multiply degrees by pi/180 or (4*ATN(1))/180 to get radians.

SQR

SQR(x) gives the positive square root of the expression x. You cannot evaluate the square root of a negative number.

TAN

TAN(x) gives the tangent of the angle x, where x is expressed in radians.

The functions SIN(x), COS(x), and TAN(x) all have the limit that the angle x must be between positive and negative $1.5707963266375 * 10^{10}$ or you will get a "BAD ARGUMENT" message and the program will stop.

DEFining Your Own Functions

If you wish to use a function that is not listed here, a combination of these functions, or any sort of formula or equation, you may define your own functions with a DEF statement. The main stipulation is that the DEF statement line must be numbered lower than any line number that uses the function. It's usually simplest to put DEF statements at the beginning of a program.

You may use the DEF statement any time you have an expression that you would rather not type several times. For example, in "Coordinate Geometry" (Program 4-4), line 100 defines a function R(N) to be a random integer number from 1 to N. The statement is:

100 DEF R(N)=INT(N*RND+1)

This function is executed in several places later in the program. Line 750 is X=R(5), which chooses a random x-coordinate from 1 to 5. Lines 1320 and 1330 have X=R(7) and Y=R(4), and lines 1850 and 1870 have X=R(7) and Y=R(4). These lines choose a point within the limits of the graph. Nearly all the problems and examples in the program use the defined random function.

The following program defines some functions, then prints the results for the answers to some algebra homework.

```
100 REM FUNCTIONS
110 DEF F(X)=X^3+2*X^2-5*X
120 DEF G(X)=X^3+X^2+X
130 DEF H(X)=F(X)+G(X)
140 CALL CLEAR
150 INPUT "X = ":X
160 PRINT :"F(X) =";F(X)
170 PRINT :"G(X) =";G(X)
180 PRINT :"H(X) =";H(X)
190 PRINT :::
200 GOTO 150
210 END
```

The user INPUTs a value for *X*. The computer then evaluates each function and prints the answers. Of course, you can add printer statements and have the computer print out the homework.

The computer allows rapid iterations of numerical combinations that would otherwise be time consuming because of the mathematics involved. Following are two programs on electrical engineering circuit design that illustrate how much easier it is to make complex calculations with the computer than by hand.

Electrical Engineering Circuit Design

These two programs are designed to help the electrical engineering professional or student analyze and design basic electrical circuits. Elementary circuits are illustrated on the screen and can be evaluated or converted quickly without the user having to work with tedious mathematical equations.

The professional engineer may use these programs to design several circuits with varying data to quickly optimize a solution. An electrical engineering student can learn about circuits more easily by using this computer program. By trial and error you can enter many combinations of data to study the corresponding results.

The user INPUTs numerical data for the given circuit, and an equivalent or resultant circuit design is printed on the screen. All elements of complex numbers are rounded to the third decimal place.

If you have an RS-232 Interface and a printer, you can alter the programs to get a printed copy of each problem you enter. The necessary statements for using the printer are listed in the programs as REMarks. Simply delete the REM at the beginning of each statement that has "#1" in it, and delete line 530 FLAG=1 in Part Two. You may also need to adjust the OPEN #1 statements in each program for your particular printer configuration (stop bits and baud rate).

The first program includes the following circuits:

1. *Series resistance* is the sum of the values of all the resistors in series. Enter the numerical values of the resistors, R1, R2, R3, etc., one at a time, and the total will be calculated.

2. *Parallel resistance* is the reciprocal of the sum of the reciprocal values of all the parallel resistors.

$$\frac{1}{RT} = \frac{1}{R1} + \frac{1}{R2} + \frac{1}{R3} + \cdots$$

For both the series and the parallel resistance circuits, any number of resistors may be used. However, in this program, to avoid an INPUT error of a very large number, a maximum of 50 resistors is allowed. You may solve problems with more resistors by solving 50 at a time and combining the solutions.

Linear electrical networks can be represented by equivalent networks which are more readily analyzed. The next four sections convert one circuit to an equivalent one for a more desirable circuit analysis.

3. Converting a resistive T-section to an equivalent Pi-section, also known as *Y-Delta conversions*. Given elements R_1, R_2, and R_3 of a resistive T-section, the corresponding elements of the Pi-section are R_A, R_B, and R_C. They are calculated using the following formulas.

$$\frac{1}{R_A} = \frac{R_2}{R_1R_2+R_1R_3+R_2R_3}$$

$$\frac{1}{R_B} = \frac{R_3}{R_1R_2+R_1R_3+R_2R_3}$$

$$\frac{1}{R_C} = \frac{R_1}{R_1R_2+R_1R_3+R_2R_3}$$

4. Converting a resistive Pi-section to an equivalent T-section, also known as a *Delta-Y conversion*. This is just the converse of the previous calculation (involving more fractions because of reciprocals).

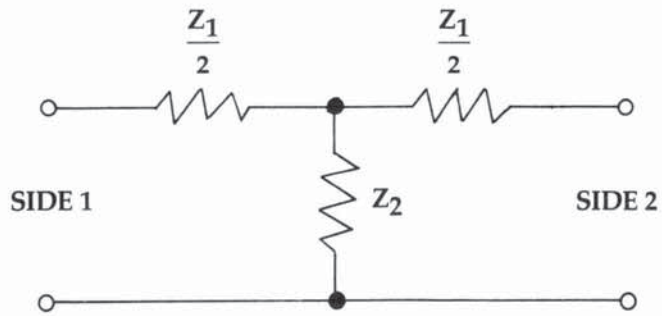
5. Converting a complex admittance Pi-section to an equivalent complex impedance T-section. This extends the previous T-Pi or Pi-T conversions to the sinusoidal steady state frequency domain. The T network is driven by the two sinusoidal current generators I_1 and I_2. All voltages, currents, and admittances are complex numbers.

6. Converting a complex impedance T-section to an equivalent complex admittance Pi-section. The calculations are similar to those for the previous conversion.

In each section, the circuit is drawn on the screen with all parts labeled.

Part Two of Electrical Engineering Circuit Design consists of the following circuits for analysis:

1. Symmetrical T-Section:



It is possible to represent a reciprocal two-port network by an equivalent T network. This part of the program is the calculation of an equivalent symmetrical T network and the corresponding characteristic impedance from short-circuit and open-circuit laboratory measurements. The impedance at either side with the opposite side open circuited is:

$$Z_{OC} = R_{OC} + j\,X_{OC} = \frac{Z_1}{2} + Z_2$$

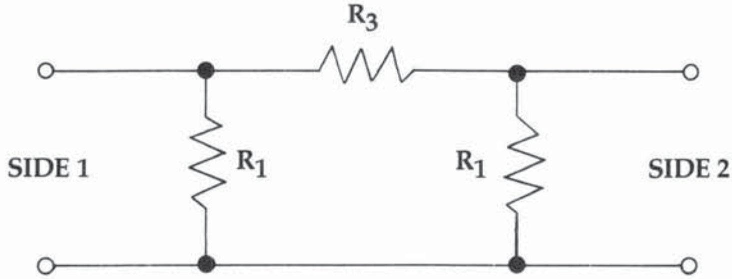
The impedance seen at either side with the opposite side short circuited is:

$$Z_{SC} = R_{SC} + j\,X_{SC} = \frac{Z_1}{2} + \frac{\frac{Z_1}{2}\quad Z_2}{\frac{Z_1}{2} + Z_2}$$

In general, these equations involve complex quantities.

The user enters R_{OC}, X_{OC}, R_{SC}, and X_{SC}, the real and imaginary parts of the open-circuit and short-circuit impedances. The characteristic impedance Z_0 for the symmetrical network is calculated and printed in polar form. Then the computed values of $Z_1/2$ and Z_2 for the equivalent T-section are calculated and printed.

2. Symmetrical Pi-Section:



This section of the program is the design of a symmetrical resistive pi attenuator. It is used to reduce the power (voltage) that will be supplied to a given laboratory load or measuring instrument, and at the same time preserve a matched load condition. The pi attenuator can be inserted to introduce a fixed amount of decibel (DB) loss.

The user INPUTs the numerical value for the characteristic resistance, R_O. This resistance, if connected at either port, will also be seen at the other port as the same resistance. This permits the insertion of one or more sections without affecting the matched load condition.

The user next specifies the required attenuation of DB. The numerical values of R1 and R3 are then calculated and printed. The formulas are:

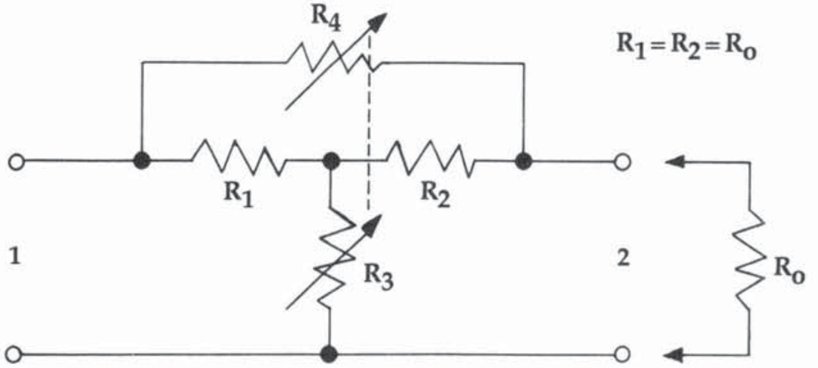
$$R1 = \frac{R_O}{\tanh \frac{\theta}{2}}$$

$$R3 = R_O \sinh \theta$$

where θ is the loss in nepers (1 neper = 8.686 decibels).

The user continues to enter different values of DB, and the corresponding values of R1 and R3 are returned. To end this process, you must enter 0 for the value of DB. You may then enter a different value for R_O; then continue or return to the menu screen.

3. Bridged T attenuator:



The loss using the symmetrical bridged T attenuator may be adjusted by varying the values of R_3 and R_4. Two special cases are: when $R_3 = 0$ and $R_4 =$ infinity, the input resistance equals R_O and the attenuation is infinite; and when $R_3 =$ infinity and $R_4 = 0$, the input resistance will equal R_O, but the loss will be equal to zero. Between these two cases, the input resistance can be kept at the value of R_O and the loss adjusted to any desirable value by adjusting R_3 and R_4 using these equations:

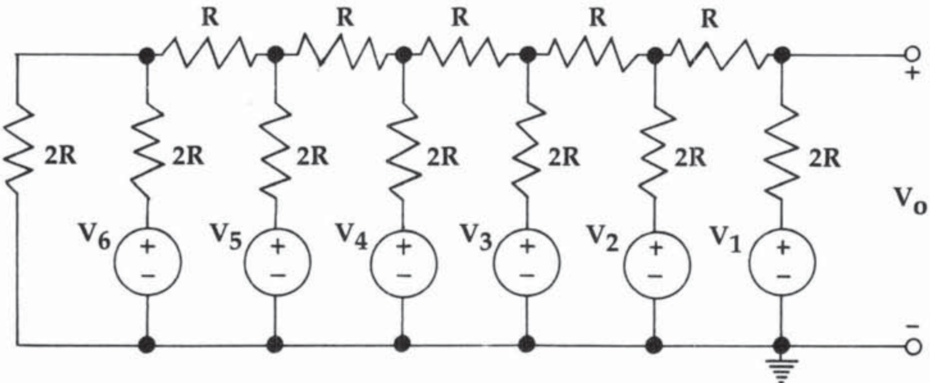
$$R_3 = \frac{R_O}{\frac{1}{VR} - 1}$$

$$R_4 = R_O \left(\frac{1}{VR} - 1\right)$$

where VR is the desired ratio of V2/V1 and the characteristic resistance R_O is specified.

Enter a numerical value for the characteristic resistance, R_O. Then enter varying values for VR, the voltage ratio. The corresponding values for R3 and R4 will be calculated and printed. To end this process, enter –1. You will then have the choice of doing another problem of the same type (entering a different value for R_O) or returning to the menu screen.

4. Digital-to-analog conversion:



In an electrical system it may be necessary to feed information that has been stored digitally into analog computers or analog readout equipment. This program involves a six-bit digital-to-analog converter. The input voltages are restricted to the binary ''1'' state or the binary ''0'' state, which may in practice correspond to ten volts and zero volts. By combining resistors that are in series or parallel, the output voltage is given by:

$$V_O = \frac{1}{2}V_1 + \frac{1}{4}V_2 + \frac{1}{8}V_3 + \frac{1}{16}V_4 + \frac{1}{32}V_5 + \frac{1}{64}V_6$$

This program determines the actual analog output voltage for a given binary number stored in the counter. Enter the six input voltages corresponding to the binary stored in the counter. The actual output voltage is calculated and printed. You may then enter another set of input voltages, and the corresponding output voltage will be determined. This process continues until you enter a value of –1. The program then returns to the menu screen.

Program 5-1. Electrical Engineering Circuit Design 1

```
100 CALL CLEAR
110 CALL CHAR(96,"000000FFFF")
120 CALL CHAR(97,"2070D0880905060 2")
130 PRINT "{3 SPACES}ELECTRICAL ENGINEERING
    ":::TAB(7);"CIRCUIT DESIGN"::::::::::::
```

```
140 CALL HCHAR(16,11,96,9)
150 CALL HCHAR(16,14,97,3)
160 FOR C=98 TO 121
170 READ C$
180 CALL CHAR(C,C$)
190 NEXT C
200 DATA 000000F8F8080808,0000000F0F080808,
    0808080808080808,080808F8F8,0808080F0
    F,000000FFFF080808,080808FFFF
210 DATA 0C06030618603000C,06030618603000C06,
    0306186030180808,000E11F1F1110E,00384
    4C7C74438,0070888F8F887
220 DATA 000C18FFFF180C,006030FFFF306,07182
    0404081838,E018048282C1E181,818181828
    20418E,8080804040201807
230 DATA 0718204043808080,E0188482E2818101,0
    10101E2020418E,8080804340201807,00000
    01818
240 REM  OPEN #1:"RS232.BA=600"
250 CALL SCREEN(2)
260 PRINT :::::::"1 SERIES RESISTANCE"::"2
    PARALLEL RESISTANCE"::"3 T-PI OR Y-DE
    LTA CONVERSION"
270 PRINT :"4 PI-T OR DELTA-Y CONVERSION"::
    "5 COMPLEX PI TO T"::"6 COMPLEX T TO
    PI"::"7 END PROGRAM"::::
280 CALL SCREEN(8)
290 CALL KEY(0,K,S)
300 IF (K<49)+(K>55)THEN 290
310 CALL CLEAR
320 ON K-48 GOTO 390,830,1230,1850,2450,306
    0,3670
330 READ N
340 FOR I=1 TO N
350 READ X,Y,GR
360 CALL HCHAR(X,Y,GR)
370 NEXT I
380 RETURN
390 PRINT " ** SERIES RESISTANCE **":::::::
    :
400 REM  PRINT #1::::"** SERIES RESISTANCE
    **":::
```

```
410 GOSUB 610
420 PRINT ::"TOTAL R = R1+R2+R3 ...":::"YOU
    R PROBLEM:"::
430 INPUT "HOW MANY RESISTORS? ":N
440 IF N<1 THEN 570
450 IF N>50 THEN 590
460 PRINT
470 RT=0
480 FOR I=1 TO N
490 INPUT " R"&STR$(I)&" = ":R
500 REM  PRINT #1:" R"&STR$(I)&" =";R
510 RT=RT+R
520 NEXT I
530 PRINT :" RT =";RT:::
540 REM  PRINT #1::" RT =";RT:::
550 GOSUB 3610
560 GOTO 430
570 PRINT :"YOU HAVE TO HAVE ONE OR MORE FO
    R A DECENT PROBLEM."::
580 GOTO 430
590 PRINT :"ARE YOU SURE?":"FOR >50 SOLVE I
    N STEPS."::
600 GOTO 430
610 FOR X=19 TO 23 STEP 4
620 CALL HCHAR(X,9,110)
630 CALL HCHAR(X,10,96,12)
640 CALL HCHAR(X,12,121,3)
650 CALL HCHAR(X,18,97,3)
660 NEXT X
670 X=20
680 Y=22
690 GOSUB 790
700 CALL HCHAR(19,22,98)
710 CALL HCHAR(23,Y,101)
720 CALL HCHAR(18,19,82)
730 CALL HCHAR(18,20,49)
740 CALL HCHAR(X+1,Y+1,82)
750 CALL HCHAR(X+1,Y+2,50)
760 CALL HCHAR(24,19,82)
770 CALL HCHAR(24,20,51)
780 RETURN
790 CALL VCHAR(X,Y,105)
```

```
800 CALL VCHAR(X+1,Y,106)
810 CALL VCHAR(X+2,Y,107)
820 RETURN
830 PRINT "** PARALLEL RESISTANCE **"::::::
    ::
840 REM  PRINT #1::::"** PARALLEL RESISTAN
    CE **":::
850 GOSUB 1090
860 PRINT ::"  1{4 SPACES}1{4 SPACES}1
    {4 SPACES}1":" -- = -- + -- + -- + ..."
870 PRINT " RT{3 SPACES}R1{3 SPACES}R2
    {3 SPACES}R3":::"YOUR PROBLEM:"::
880 INPUT "HOW MANY RESISTORS? ":N
890 IF N<1 THEN 1050
900 IF N>50 THEN 1070
910 PRINT
920 RTD=0
930 FOR I=1 TO N
940 INPUT " R"&STR$(I)&" = ":R
950 IF R<>0 THEN 980
960 PRINT :"SORRY - ZERO IS NOT ALLOWED"::
970 GOTO 940
980 RTD=RTD+1/R
990 REM  PRINT #1:" R"&STR$(I)&" =";R
1000 NEXT I
1010 PRINT :" RT =";1/RTD:::
1020 REM  PRINT #1::" RT =";1/RTD:::
1030 GOSUB 3610
1040 GOTO 880
1050 PRINT :"ONE OR MORE PLEASE."::
1060 GOTO 880
1070 PRINT :"REALLY?  FOR >50 RESISTORS  SO
     LVE IN SEVERAL STEPS."::
1080 GOTO 880
1090 FOR X=19 TO 23 STEP 4
1100 CALL HCHAR(X,10,110)
1110 CALL HCHAR(X,11,96,12)
1120 CALL HCHAR(X,12,121,3)
1130 NEXT X
1140 X=20
1150 FOR Y=17 TO 23 STEP 3
1160 GOSUB 790
```

```
1170 NEXT Y
1180 DATA 13,19,17,103,19,20,103,23,17,104,
     23,20,104,19,23,98,23,23,101
1190 DATA 24,16,82,24,17,49,24,19,82,24,20,
     50,24,22,82,24,23,51,1,1,32
1200 RESTORE 1180
1210 GOSUB 330
1220 RETURN
1230 PRINT "CONVERTING A RESISTIVE
     {6 SPACES}T-SECTION TO AN EQUIALENT"
1240 PRINT "PI-SECTION (ALSO KNOWN AS
     {3 SPACES}Y-DELTA CONVERSION)":::::::
1250 REM  PRINT #1::::"CONVERTING A RESISTI
     VE T-SECTION TO AN EQUIVALENT"
1260 REM  PRINT #1:"PI-SECTION (ALSO KNOWN
     AS Y-DELTA CONVERSION)"::::
1270 XO=19
1280 GOSUB 1470
1290 PRINT "YOUR PROBLEM:":::
1300 INPUT " R1 = ":R1
1310 INPUT " R2 = ":R2
1320 INPUT " R3 = ":R3
1330 SUM=R1+R2+R3
1340 IF SUM<>0 THEN 1370
1350 PRINT ::"SORRY - THE SUM OF THE THREE
     VALUES CANNOT BE ZERO."::
1360 GOTO 1300
1370 RA=R2*R3/SUM
1380 PRINT ::" RA =";RA
1390 RB=R1*R3/SUM
1400 PRINT " RB =";RB
1410 RC=R1*R2/SUM
1420 PRINT " RC =";RC::
1430 REM  PRINT #1:" R1 =";R1:" R2 = ";R2:"
     R3 =";R3::" RA =";RA:" RB =";RB:" RC
     =";RC:::
1440 GOSUB 3610
1450 GOSUB 1470
1460 GOTO 1300
1470 FOR X=XO TO XO+4 STEP 4
1480 CALL HCHAR(X,3,110)
1490 CALL HCHAR(X,4,96,11)
```

```
1500 CALL HCHAR(X,15,108)
1510 CALL HCHAR(X,20,110)
1520 CALL HCHAR(X,21,96,9)
1530 CALL HCHAR(X,30,108)
1540 NEXT X
1550 CALL HCHAR(XO,5,97,3)
1560 CALL HCHAR(XO,11,97,3)
1570 CALL HCHAR(XO,9,103)
1580 X=XO+1
1590 Y=9
1600 GOSUB 790
1610 CALL HCHAR(XO+4,9,104)
1620 CALL HCHAR(XO,24,97,3)
1630 CALL HCHAR(XO,22,103)
1640 Y=22
1650 GOSUB 790
1660 CALL HCHAR(XO+4,22,104)
1670 CALL HCHAR(XO,28,103)
1680 Y=28
1690 GOSUB 790
1700 CALL HCHAR(XO+4,28,104)
1710 CALL HCHAR(XO-1,6,82)
1720 CALL HCHAR(XO-1,7,49)
1730 CALL HCHAR(XO-1,12,82)
1740 CALL HCHAR(XO-1,13,50)
1750 CALL HCHAR(XO+2,10,82)
1760 CALL HCHAR(XO+2,11,51)
1770 CALL HCHAR(XO+2,23,82)
1780 CALL HCHAR(XO+2,24,65)
1790 CALL HCHAR(XO-1,25,82)
1800 CALL HCHAR(XO-1,26,66)
1810 CALL HCHAR(XO+2,29,82)
1820 CALL HCHAR(XO+2,30,67)
1830 PRINT :::
1840 RETURN
1850 PRINT "CONVERTING A RESISTIVE
     {6 SPACES}PI-SECTION TO AN EQUIVALENT"
1860 PRINT "T-SECTION (ALSO KNOWN AS
     {4 SPACES}DELTA-Y CONVERSION)"::::::::
1870 REM  PRINT #1::::"CONVERTING A RESISTI
     VE PI-SECTION TO AN EQUIVALENT"
1880 REM  PRINT #1:"T-SECTION (ALSO KNOWN A
     S DELTA-Y CONVERSION)"::::
```

```
1890 XO=19
1900 GOSUB 2150
1910 PRINT "YOUR PROBLEM:":::
1920 INPUT " RA = ":RA
1930 IF RA<>0 THEN 1960
1940 PRINT :"SORRY, RA CANNOT BE ZERO"::
1950 GOTO 1920
1960 INPUT " RB = ":RB
1970 IF RB<>0 THEN 2000
1980 PRINT :"SORRY, RB CANNOT BE ZERO"::
1990 GOTO 1960
2000 INPUT " RC = ":RC
2010 IF RC<>0 THEN 2040
2020 PRINT :"SORRY, RC CANNOT BE ZERO"::
2030 GOTO 2000
2040 SUM=RA*RB+RA*RC+RB*RC
2050 R1=SUM/RA
2060 PRINT ::" R1 =";R1
2070 R2=SUM/RB
2080 PRINT " R2 =";R2
2090 R3=SUM/RC
2100 PRINT " R3 =";R3::
2110 REM  PRINT #1:::" RA =";RA:" RB =";RB:
     " RC =";RC:::" R1 =";R1:" R2 =";R2:"
     R3 =";R3:::
2120 GOSUB 3610
2130 GOSUB 2150
2140 GOTO 1920
2150 FOR X=XO TO XO+4 STEP 4
2160 CALL HCHAR(X,3,110)
2170 CALL HCHAR(X,4,96,9)
2180 CALL HCHAR(X,13,108)
2190 CALL HCHAR(X,17,110)
2200 CALL HCHAR(X,18,96,11)
2210 CALL HCHAR(X,29,108)
2220 NEXT X
2230 CALL HCHAR(XO,7,97,3)
2240 CALL HCHAR(XO,5,103)
2250 X=XO+1
2260 Y=5
2270 GOSUB 790
2280 CALL HCHAR(XO+4,5,104)
```

```
2290 CALL HCHAR(XO,11,103)
2300 Y=11
2310 GOSUB 790
2320 CALL HCHAR(XO+4,11,104)
2330 CALL HCHAR(XO,19,97,3)
2340 CALL HCHAR(XO,25,97,3)
2350 CALL HCHAR(XO,23,103)
2360 Y=23
2370 GOSUB 790
2380 CALL HCHAR(XO+4,Y,104)
2390 DATA 13,21,6,82,21,7,65,18,8,82,18,9,6
     6,21,12,82,21,13,67
2400 DATA 18,20,82,18,21,49,18,26,82,18,27,
     50,21,24,82,21,25,51,1,1,32
2410 RESTORE 2390
2420 GOSUB 330
2430 PRINT :::
2440 RETURN
2450 PRINT "CONVERTING A COMPLEX{8 SPACES}A
     DMITTANCE PI-SECTION TO"
2460 PRINT "AN EQUIVALENT COMPLEX
     {7 SPACES}IMPEDANCE T-SECTION"::::::::
2470 REM  PRINT #1::::"CONVERTING A COMPLEX
      ADMITTANCE PI-SECTION TO"
2480 REM  PRINT #1:"AN EQUIVALENT COMPLEX I
     MPEDANCE T-SECTION"::::
2490 XO=19
2500 GOSUB 2840
2510 INPUT "  AA = ":AA
2520 INPUT "J BA = ":BA
2530 INPUT "  AB = ":AB
2540 INPUT "J BB = ":BB
2550 INPUT "  AC = ":AC
2560 INPUT "J BC = ":BC
2570 API=AA*AB-BA*BB+AA*AC-BA*BC+AB*AC-BB*B
     C
2580 BPI=BA*AB+AA*BB+BA*AC+AA*BC+BB*AC+AB*B
     C
2590 D=API*API+BPI*BPI
2600 IF D<>0 THEN 2640
2610 PRINT :"DENOMINATOR CANNOT = 0"::
2620 GOSUB 3610
```

```
2630 GOTO 2500
2640 PRINT ::"GIVEN PI-SECTION:"::
2650 PRINT " YA =";AA;" + J (";BA;")"
2660 PRINT " YB =";AB;" + J (";BB;")"
2670 PRINT " YC =";AC;" + J (";BC;")"
2680 R1=(INT(1000*((AC*API+BC*BPI)/D+.0005)
     ))/1000
2690 X1=(INT(1000*((BC*API-AC*BPI)/D+.0005)
     ))/1000
2700 PRINT ::"EQUIVALENT T-SECTION:"::
2710 PRINT " Z1 =";R1;" + J (";X1;")"
2720 R2=(INT(1000*((AA*API+BA*BPI)/D+.0005)
     ))/1000
2730 X2=(INT(1000*((BA*API-AA*BPI)/D+.0005)
     ))/1000
2740 PRINT " Z2 =";R2;" + J (";X2;")"
2750 R3=(INT(1000*((AB*API+BB*BPI)/D+.0005)
     ))/1000
2760 X3=(INT(1000*((BB*API-AB*BPI)/D+.0005)
     ))/1000
2770 PRINT " Z3 =";R3;" + J (";X3;")":::
2780 REM  PRINT #1:::"GIVEN COMPLEX ADMITTA
     NCE PI-SECTION:"::" YA =";AA;" + J ("
     ;BA;")"
2790 REM  PRINT #1:" YB =";AB;" + J (";BB;"
     )":" YC =";AC;" + J (";BC;")"
2800 REM  PRINT #1:::"EQUIVALENT COMPLEX IM
     PEDANCE T-SECTION:"::" Z1 =";R1;" + J
     (";X1;")"
2810 REM  PRINT #1:" Z2 =";R2;" + J (";X2;"
     )":" Z3 =";R3;" + J (";X3;")"::::
2820 GOSUB 3610
2830 GOTO 2500
2840 CALL HCHAR(19,5,96,9)
2850 CALL HCHAR(19,8,97,3)
2860 CALL HCHAR(24,5,96,9)
2870 CALL VCHAR(20,3,100,4)
2880 CALL VCHAR(20,15,100,4)
2890 CALL VCHAR(20,19,100,4)
2900 CALL VCHAR(20,31,100,4)
2910 CALL HCHAR(24,20,96,11)
2920 CALL HCHAR(19,22,97,3)
```

```
2930 CALL HCHAR(19,26,97,3)
2940 DATA 65,21,6,105,22,6,106,23,6,107,21,
     12,105,22,12,106,23,12,107,21,25,105,
     22,25,106,23,25,107
2950 DATA 19,3,99,19,4,109,19,6,103,20,6,10
     0,19,12,103,20,12,100,19,14,109,19,15
     ,98,24,3,102
2960 DATA 24,4,109,24,14,109,24,15,101,24,6
     ,104,24,12,104,21,2,113,21,3,114,22,3
     ,115,22,2,116
2970 DATA 21,14,113,21,15,114,22,15,115,22,
     14,116,19,19,99,19,20,112,19,21,109,1
     9,25,103,20,25,100
2980 DATA 19,29,109,19,30,111,19,31,98,24,1
     9,102,24,21,109,24,25,104,24,29,109,2
     4,31,101,21,18,117
2990 DATA 21,19,118,22,19,119,22,18,120,21,
     30,117,21,31,118,22,31,119,22,30,120,
     22,7,89,22,8,65,18,8
3000 DATA 89,18,9,66,22,10,89,22,11,67,18,2
     2,90,18,23,49,18,27,90,18,28,50,22,26
     ,90,22,27,51,1,1,32
3010 RESTORE 2940
3020 GOSUB 330
3030 PRINT :::"YA=AA + J BA{4 SPACES}Z1=R1
     + J X1YB=AB + J BB{4 SPACES}Z2=R2 + J
     X2"
3040 PRINT "YC=AC + J BC{4 SPACES}Z3=R3 + J
      X3":::
3050 RETURN
3060 PRINT "CONVERTING A COMPLEX{8 SPACES}I
     MPEDANCE T-SECTION TO"
3070 PRINT "AN EQUIVALENT COMPLEX
     {7 SPACES}ADMITTANCE PI-SECTION":::::
     ::
3080 REM  PRINT #1:"CONVERTING A COMPLEX IM
     PEDANCE T-SECTION TO"
3090 REM  PRINT #1:"AN EQUIVALENT COMPLEX A
     DMITTANCE PI-SECTION"::::
3100 GOSUB 3400
3110 INPUT " R1 = ":R1
3120 INPUT " X1 = ":X1
```

```
3130 INPUT " R2 = ":R2
3140 INPUT " X2 = ":X2
3150 INPUT " R3 = ":R3
3160 INPUT " X3 = ":X3
3170 RT=R1*R2-X2*X2+R1*R3-X1*X3+R2*R3-X2*X3
3180 XT=R1*X2+R2*X1+R1*X3+R3*X1+R2*X3+R3*X2
3190 D=RT*RT+XT*XT
3200 IF D<>0 THEN 3240
3210 PRINT :"SORRY, DENOMINATOR CANNOT
     {4 SPACES}EQUAL ZERO."::
3220 GOSUB 3610
3230 GOTO 3100
3240 PRINT ::"ELEMENTS OF T-SECTION:":" Z1
     =";R1;" + J (";X1;")":" Z2 =";R2;" +
      J (";X2;")"
3250 PRINT " Z3 =";R3;" + J (";X3;")":::"E
     QUIVALENT PI-SECTION:"
3260 AA=(INT(1000*((R2*RT+X2*XT)/D+.0005)))
     /1000
3270 BA=(INT(1000*((X2*RT-R2*XT)/D+.0005)))
     /1000
3280 PRINT :" YA =";AA;" + J (";BA;")"
3290 AB=(INT(1000*((R3*RT+X3*XT)/D+.0005)))
     /1000
3300 BB=(INT(1000*((X3*RT-R3*XT)/D+.0005)))
     /1000
3310 PRINT " YB =";AB;" + J (";BB;")"
3320 AC=(INT(1000*((R1*RT+X1*XT)/D+.0005)))
     /1000
3330 BC=(INT(1000*((X1*RT-R1*XT)/D+.0005)))
     /1000
3340 PRINT " YC =";AC;" + J (";BC;")"::::
3350 REM  PRINT #1:"ELEMENTS OF T-SECTION:"
     :::" Z1 =";R2;" + J (";X1;")":" Z2 ="
     ;R2;" ++ J (";X2;")"
3360 REM  PRINT #1:" Z3 =";R3;" + J (";X3;"
     )":::"EQUIVALENT PI-SECTION:"::" YA =
     ";AA;" + J (";BA;")"
3370 REM  PRINT #1:" YB =";AB;" + J (";BB;"
     )":" YC =";AC;" + J (";BC;")"::::
3380 GOSUB 3610
3390 GOTO 3100
```

```
3400 CALL HCHAR(24,4,96,11)
3410 CALL HCHAR(19,6,97,3)
3420 CALL HCHAR(19,10,97,3)
3430 CALL HCHAR(19,21,96,9)
3440 CALL HCHAR(24,21,96,9)
3450 CALL VCHAR(20,19,100,4)
3460 CALL VCHAR(20,31,100,4)
3470 CALL HCHAR(19,24,97,3)
3480 DATA 69,19,3,99,19,4,112,19,5,109,19,9
     ,103,19,13,109,19,14,111,19,15,98,20,
     3,100,20,9,100
3490 DATA 20,15,100,23,3,100,23,15,100,24,3
     ,102,24,15,101,21,2,117,21,3,118,22,3
     ,119,22,2,120
3500 DATA 21,14,117,21,15,118,22,15,119,22,
     14,120,24,5,109,24,13,109,21,9,105,22
     ,9,106,23,9,107
3510 DATA 24,9,104,19,19,99,19,20,109,19,22
     ,103,19,28,103,19,30,109,19,31,98,24,
     19,102,24,20,109
3520 DATA 24,30,109,24,31,101,24,22,104,24,
     28,104,21,18,113,21,19,114,22,19,115,
     22,18,116,21,30,113
3530 DATA 21,31,114,22,31,115,22,30,116,21,
     22,105,22,22,106,23,22,107,21,28,105,
     22,28,106,23,28,107
3540 DATA 20,22,100,20,28,100,18,7,90,18,8,
     49,18,11,90,18,12,50,22,10,90,22,11,51,
     22,23,89
3550 DATA 22,24,65,18,24,89,18,25,66,22,26,
     89,22,27,67,1,1,32
3560 RESTORE 3480
3570 GOSUB 330
3580 PRINT :::"Z1=R1 + J X1{4 SPACES}YA=AA
     + J BAZ2=R2 + J X2{4 SPACES}YB=AB + J
     BB"
3590 PRINT "Z3=R3 + J X3{4 SPACES}YC=AC + J
      BC":::
3600 RETURN
3610 PRINT :"DO YOU HAVE MORE PROBLEMS
     {3 SPACES}OF THIS TYPE? (Y/N)"
```

```
3620 CALL KEY(0,K,S)
3630 IF K=78 THEN 250
3640 IF K<>89 THEN 3620
3650 CALL CLEAR
3660 RETURN
3670 REM  CLOSE #1
3680 END
```

Program 5-2. Electrical Engineering Circuit Design 2

```
110 REM EE CIRCUIT DESIGN PART 2
120 CALL CLEAR
130 CALL CHAR(96,"000000FFFF")
140 CALL CHAR(97,"2070D08809050602")
150 PRINT "{3 SPACES}ELECTRICAL ENGINEERING
    "
160 CALL CHAR(98,"000000F8F8080808")
170 CALL CHAR(99,"0000000F0F080808")
180 CALL CHAR(100,"0808080808080808")
190 PRINT ::TAB(7);"CIRCUIT DESIGN":::::::
    ::::
200 CALL CHAR(101,"080808F8F8")
210 CALL CHAR(102,"0808080F0F")
220 CALL CHAR(103,"000000FFFF080808")
230 CALL HCHAR(16,11,96,9)
240 CALL HCHAR(16,14,97,3)
250 CALL CHAR(104,"080808FFFF")
260 CALL CHAR(105,"0C0603061860300C")
270 CALL CHAR(106,"0603061860300C06")
280 CALL CHAR(107,"0306186030180808")
290 CALL CHAR(108,"000E11F1F1110E")
300 CALL CHAR(110,"0070888F8F887")
310 PRINT TAB(9);"PART TWO"::
320 CALL CHAR(111,"061E7CFFFF7C1E06")
330 CALL CHAR(117,"0718204043808080")
340 CALL CHAR(118,"E0188482E2818101")
350 CALL CHAR(119,"0101011E2020418E")
360 CALL CHAR(120,"8080804340201807")
370 CALL CHAR(122,"080808FFFF0808FF")
380 CALL CHAR(123,"007E003C0018")
390 CALL CHAR(147,"0010301010101")
400 CALL CHAR(148,"003844081020 7C")
```

```
410 CALL COLOR(15,7,1)
420 CALL CHAR(124,"Ø1Ø2Ø4Ø81Ø2Ø4Ø8")
430 CALL CHAR(125,"2172D48819254682")
440 CALL CHAR(126,"Ø7Ø3Ø5Ø81Ø244484")
450 CALL CHAR(127,"2Ø64D494Ø9ØDØ6Ø6")
460 CALL CHAR(128,"ØØØ4Ø4Ø4ØØØ4Ø4Ø4")
470 CALL CHAR(129,"Ø7Ø3Ø61870304C86")
480 CALL CHAR(130,"Ø7Ø3Ø5Ø81Ø2Ø4Ø8")
490 CALL CHAR(131,"ØØØ4ØØFFFFØØØ4Ø4")
500 CALL CHAR(132,"ØØØØ36494936")
510 REM  OPEN #1:"RS232.TW.BA=11Ø"
520 FLAG=Ø
530 REM  FLAG=1
540 CALL CLEAR
550 CALL SCREEN(2)
560 PRINT ::::"1 SYMMETRICAL T-SECTION"::"2
     SYMMETRICAL PI-SECTION"
570 PRINT :"3 BRIDGED T ATTENUATOR"::"4 DIG
    ITAL TO ANALOG"
580 PRINT ::"5 END PROGRAM"::::::::
590 CALL SCREEN(8)
600 CALL KEY(Ø,K,S)
610 IF K<49 THEN 6ØØ
620 IF K>53 THEN 6ØØ
630 CALL CLEAR
640 ON K-48 GOTO 65Ø,159Ø,199Ø,256Ø,306Ø
650 PRINT "CALCULATION OF AN EQUIVALENTSYMM
    ETRICAL NETWORK AND THE"
660 PRINT "CORRESPONDING CHARACTERISTIC IMPE
    DANCE FROM SHORT-CIRCUIT"
670 PRINT "AND OPEN-CIRCUIT TESTS"::"SYMMET
    RICAL T-SECTION"::::::::::
680 REM  PRINT #1::::"CALCULATION OF AN EQU
    IVALENT"
690 REM  PRINT #1:"SYMMETRICAL NETWORK AND
    THE"
700 REM  PRINT #1:"CORRESPONDING CHARACTERI
    STIC"
710 REM  PRINT #1:"IMPEDANCE FROM SHORT-CIR
    CUIT"
720 REM  PRINT #1:"AND OPEN-CIRCUIT TESTS"
```

```
730 REM  PRINT #1::"SYMMETRICAL T-SECTION":
    ::
740 IF FLAG=1 THEN 770
750 FOR DELAY=1 TO 400
760 NEXT DELAY
770 GOSUB 1260
780 PRINT "IMPEDANCE WITH OPPOSITE SIDE OPE
    N CIRCUITED:"
790 PRINT :"{3 SPACES}ZOC = ROC + J XOC":"
    {7 SPACES}= Z1/2 + Z2"
800 PRINT :"IMPEDANCE WITH OPPOSITE SIDE SH
    ORT CIRCUITED:"
810 PRINT :"{3 SPACES}ZSC = RSC + J XSC"
820 PRINT "{7 SPACES}= ZT +(ZT*Z2)/(ZT+Z2)"
830 PRINT TAB(10);"WHERE ZT=Z1/2"::
840 INPUT " ROC = ":ROC
850 INPUT " XOC = ":XOC
860 INPUT " RSC = ":RSC
870 INPUT " XSC = ":XSC
880 A=ROC*(ROC-RSC)-XOC*(XOC-XSC)
890 PRINT :"SYMMETRICAL T EQUIVALENT"
900 B=XOC*(ROC-RSC)+ROC*(XOC-XSC)
910 PRINT "FROM SHORT CIRCUIT AND":"OPEN CI
    RCUIT TESTS":::
920 C=ROC*RSC-XOC*XSC
930 PRINT "ZOC =";ROC;"+ J (";XOC;")"
940 D=ROC*XSC+RSC*XOC
950 PRINT "ZSC =";RSC;"+ J (";XSC;")"
960 ZOM=SQR(SQR(C*C+D*D))
970 IZOM=(INT(1000*(ZOM+.0005)))/1000
980 ZOA=0.5*ATN(D/C)
990 IZOA=(INT(1000*(ZOA+.0005)))/1000
1000 PRINT :" ZO =";IZOM;" EXP J (";IZOA;")
     "
1010 RO=(INT(1000*(ZOM*COS(ZOA)+.0005)))/10
     00
1020 XO=(INT(1000*(ZOM*SIN(ZOA)+.0005)))/10
     00
1030 PRINT "{4 SPACES}=";RO;"+ J (";XO;")"
1040 Z2M=SQR(SQR(A*A+B*B))
1050 Z2A=.5*ATN(B/A)
1060 PRINT ::"EQUIVALENT T-SECTION"::
```

```
1070 R2=Z2M*COS(Z2A)
1080 IR2=(INT(1000*(R2+.0005)))/1000
1090 X2=Z2M*SIN(Z2A)
1100 IX2=(INT(1000*(X2+.0005)))/1000
1110 PRINT "  Z2 =";IR2;"+ J (";IX2;")"
1120 R12=(INT(1000*(ROC-R2+.0005)))/1000
1130 X12=(INT(1000*(XOC-X2+.0005)))/1000
1140 PRINT "Z1/2 =";R12;"+ J (";X12;")"
1150 REM  PRINT #1:::"SYMMETRICAL T EQUIVAL
     ENT FROM"
1160 REM  PRINT #1:"SHORT CIRCUIT AND OPEN
     CIRCUIT TESTS"::
1170 REM  PRINT #1:"ZOC =";ROC;"+ J (";XOC;
     ")"
1180 REM  PRINT #1:"ZSC =";RSC;"+ J (";XSC;
     ")"
1190 REM  PRINT #1::" ZO =";IZOM;" EXP J ("
     ;IZOA;")"
1200 REM  PRINT #1:"{4 SPACES}=";RO;"+ J ("
     ;XO;")"
1210 REM  PRINT #1::"EQUIVALENT T SECTION:"
1220 REM  PRINT #1::"Z1/2 =";R12;"+ J (";X1
     2;")"
1230 REM  PRINT #1:"  Z2 =";IR2;"+ J (";IX2
     ;")"::::
1240 GOSUB 3080
1250 GOTO 770
1260 CALL HCHAR(18,6,96,19)
1270 CALL HCHAR(24,6,96,19)
1280 CALL HCHAR(18,9,97,3)
1290 CALL HCHAR(18,18,97,3)
1300 CALL VCHAR(19,15,100,5)
1310 X=18
1320 Y=15
1330 GOSUB 1440
1340 DATA 18,5,110,24,5,110,18,25,108,24,25
     ,108,17,9,90
1350 DATA 17,10,49,17,11,47,17,12,50,17,18,
     90,17,19,49
1360 DATA 17,20,47,17,21,50,21,16,90,21,17,
     50,1,1,32
1370 RESTORE 1340
```

```
1380 FOR I=1 TO 15
1390 READ X,Y,G
1400 CALL HCHAR(X,Y,G)
1410 NEXT I
1420 GOSUB 1510
1430 RETURN
1440 CALL VCHAR(X+6,Y,104)
1450 CALL VCHAR(X,Y,103)
1460 CALL VCHAR(X+1,Y,100,5)
1470 CALL VCHAR(X+2,Y,105)
1480 CALL VCHAR(X+3,Y,106)
1490 CALL VCHAR(X+4,Y,107)
1500 RETURN
1510 RESTORE 1520
1520 DATA 2,83,3,73,4,68,5,69,7,147,24,83,2
     5,73,26,68,27,69,29,148,1,32
1530 FOR I=1 TO 11
1540 READ Y,G
1550 CALL HCHAR(21,Y,G)
1560 NEXT I
1570 PRINT :::
1580 RETURN
1590 PRINT "DESIGN OF A SYMMETRICAL":"RESIS
     TIVE PI ATTENUATOR"::::::::::
1600 REM  PRINT #1::::"DESIGN OF A SYMMETRI
     CAL"
1610 REM  PRINT #1:"RESISTIVE PI ATTENUATOR
     "::::
1620 GOSUB 1800
1630 PRINT :"GIVEN CHARACTERISTIC"
1640 INPUT "RESISTANCE RO = ":RO
1650 REM  PRINT #1:::"CHARACTERISTIC RESIST
     ANCE =";RO
1660 PRINT :"INPUT ATTENUATION VALUES
     {4 SPACES}TO STOP, ENTER  0":::
1670 INPUT "DB = ":DB
1680 IF DB=0 THEN 1780
1690 TA=DB/8.686
1700 R3=RO*(EXP(TA)-EXP(-TA))/2
1710 IR3=(INT(1000*(R3+.0005)))/1000
1720 TA2=TA/2
1730 TANHT=(EXP(TA2)-EXP(-TA2))/(EXP(TA2)+E
     XP(-TA2))
```

```
1740 R1=(INT(1000*(RO/TANHT+.0005)))/1000
1750 PRINT :"R1 =";R1,"R3 =";IR3:::
1760 REM  PRINT #1::"DB =";DB,"R1 =";R1,"R3
     =";R3
1770 GOTO 1670
1780 GOSUB 3110
1790 GOTO 1620
1800 CALL HCHAR(18,6,96,19)
1810 CALL HCHAR(24,6,96,19)
1820 CALL HCHAR(18,14,97,3)
1830 X=18
1840 Y=10
1850 GOSUB 1440
1860 Y=20
1870 GOSUB 1440
1880 DATA 18,5,110,18,25,108,24,5,110,24,25
     ,108,21,11,82,21,12,49
1890 DATA 21,18,82,21,19,49,17,15,82,17,16,
     51,1,1,32
1900 RESTORE 1880
1910 FOR I=1 TO 11
1920 READ X,Y,G
1930 CALL HCHAR(X,Y,G)
1940 NEXT I
1950 GOSUB 1510
1960 PRINT "R1 = RO/TANH(N/2)"
1970 PRINT "R3 = RO*SINH(N)":"{5 SPACES}WHE
     RE N=LOSS IN NEPERS":::
1980 RETURN
1990 PRINT "DESIGN OF SYMMETRICAL":"BRIDGED
      T ATTENUATOR"::::::::::::
2000 REM  PRINT #1:::"DESIGN OF SYMMETRICA
     L"
2010 REM  PRINT #1:"BRIDGED T ATTENUATOR"::
2020 GOSUB 2260
2030 INPUT "INPUT RESISTANCE RO = ":RO
2040 REM  PRINT #1::"INPUT RESISTANCE RO =
     ";RO::
2050 PRINT ::"ENTER VARIOUS RATIOS OF":"VR=
     V2/V1; ENTER -1 TO STOP."::
2060 INPUT "VR = ":VR
2070 IF VR<0 THEN 2240
```

```
2080 IF VR<>Ø THEN 212Ø
2090 PRINT :"R3 = Ø","R4 = ";CHR$(132):::
2100 REM  PRINT #1:"VR =";VR,"R3 = Ø
     {8 SPACES}R4 = INFINITY"
2110 GOTO 2060
2120 IF VR<>1 THEN 2160
2130 PRINT :"R3 = ";CHR$(132),"R4 = Ø":::
2140 REM  PRINT #1:"VR =";VR,"R3 = INFINITY
      R4 = Ø"
2150 GOTO 2060
2160 IF VR<1 THEN 2190
2170 PRINT "Ø<VR<1 PLEASE"::
2180 GOTO 2060
2190 R3=(INT(1000*((RO/(1/VR-1))+.0005)))/1
     000
2200 R4=(INT(1000*((RO*(1/VR-1))+.0005)))/1
     000
2210 PRINT :"R3 =";R3,"R4 =";R4:::
2220 REM  PRINT #1:"VR =";VR,"R3 =";R3,"R4
     =";R4
2230 GOTO 2060
2240 GOSUB 3110
2250 GOTO 2020
2260 CALL HCHAR(15,10,96,11)
2270 CALL HCHAR(18,6,96,18)
2280 CALL HCHAR(24,6,96,18)
2290 CALL VCHAR(16,9,100,2)
2300 CALL VCHAR(16,21,100,2)
2310 CALL HCHAR(15,14,97,3)
2320 CALL HCHAR(18,11,97,3)
2330 CALL HCHAR(18,17,97,3)
2340 X=18
2350 Y=15
2360 GOSUB 1440
2370 DATA 18,5,110,24,5,110,18,24,108,24,24
     ,108,18,9,104,18,21,104
2380 DATA 15,9,99,15,21,98,16,14,124,15,15,
     125,14,16,126,15,16,127
2390 DATA 16,16,128,17,16,128,18,16,131,19,
     16,128,22,14,124,21,15,129
2400 DATA 20,16,130,20,5,147,20,24,148,14,1
     4,82,14,15,52,19,11,82
```

```
2410 DATA 19,12,49,19,18,82,19,19,50,21,16,
     82,21,17,51,14,24,82
2420 DATA 14,25,49,14,26,61,14,27,82,14,28,
     50,14,29,61,14,30,82
2430 DATA 14,31,79,21,29,82,21,30,79,18,26,
     111,24,26,111,18,27,96
2440 DATA 24,27,96,18,28,98,24,28,101,1,1,3
     2
2450 RESTORE 2370
2460 FOR I=1 TO 46
2470 READ X,Y,G
2480 CALL HCHAR(X,Y,G)
2490 NEXT I
2500 X=18
2510 Y=28
2520 GOSUB 1460
2530 PRINT :::"R3 = RO/((1/VR)-1)"
2540 PRINT "R4=RO*((1/VR)-1)":::
2550 RETURN
2560 PRINT "SIX-BIT DIGITAL TO":"ANALOG CON
     VERTER"::::::::::::::
2570 REM  PRINT #1::::"SIX-BIT DIGITAL TO A
     NALOG CONVERTER":::
2580 GOSUB 2720
2590 PRINT :::"ENTER SIX INPUT VOLTAGES.":"
     TO STOP, ENTER -1"::::
2600 F=1
2610 VO=0
2620 FOR J=1 TO 6
2630 INPUT " V"&STR$(J)&" = ":V
2640 IF V<0 THEN 540
2650 REM  PRINT #1:" V"&STR$(J)&" =";V
2660 F=.5*F
2670 VO=VO+F*V
2680 NEXT J
2690 PRINT :"V OUT =";VO:::
2700 REM  PRINT #1::" V OUT =";VO:::
2710 GOTO 2600
2720 CALL HCHAR(14,4,96,3)
2730 CALL HCHAR(23,4,96,27)
2740 CALL VCHAR(20,3,100,3)
2750 X=14
```

```
2760 Y=3
2770 GOSUB 1460
2780 CALL HCHAR(14,3,99)
2790 CALL HCHAR(23,3,102)
2800 CALL HCHAR(14,8,97,19)
2810 I=0
2820 FOR Y=7 TO 27 STEP 4
2830 GOSUB 1450
2840 CALL VCHAR(22,Y,100)
2850 CALL VCHAR(23,Y,104)
2860 CALL HCHAR(20,Y-1,117)
2870 CALL HCHAR(20,Y,118)
2880 CALL HCHAR(21,Y,119)
2890 CALL HCHAR(21,Y-1,120)
2900 CALL HCHAR(13,Y+2,82)
2910 CALL HCHAR(17,Y+1,50)
2920 CALL HCHAR(17,Y+2,82)
2930 CALL HCHAR(19,Y-2,86)
2940 CALL HCHAR(19,Y-1,54-I)
2950 I=I+1
2960 NEXT Y
2970 CALL HCHAR(14,28,96,3)
2980 DATA 14,31,108,15,31,43,23,31,108,22,3
     1,45,23,27,122,24,27,123
2990 DATA 18,30,86,18,31,79,17,4,50,17,5,82
     ,1,1,32
3000 RESTORE 2980
3010 FOR I=1 TO 11
3020 READ X,Y,G
3030 CALL HCHAR(X,Y,G)
3040 NEXT I
3050 RETURN
3060 REM  CLOSE #1
3070 STOP
3080 PRINT :"PRESS <ENTER> TO CONTINUE"
3090 CALL KEY(0,K,S)
3100 IF K<>13 THEN 3090
3110 PRINT ::"DO YOU HAVE MORE PROBLEMS
     {3 SPACES}OF THIS TYPE? (Y/N)"
3120 CALL KEY(0,K,S)
3130 IF K=78 THEN 540
3140 IF K<>89 THEN 3120
```

```
3150 CALL CLEAR
3160 RETURN
3170 END
```

String Functions

Usually, the computer expects all information to be numeric. Certain information, however, is treated as strings, or groups of characters. You signal the computer that certain information is a string by enclosing it in quotation marks: PRINT 4+4 causes the computer to print 8; PRINT "4+4" causes the computer to print 4+4. You signal the computer to treat the value of a variable as a string by ending the variable name with $.

String expressions may contain letters, numbers, and characters, and may be up to 255 characters long. Longer strings are truncated on the right.

Strings are combined or concatenated with the ampersand. To combine string A$, which is "TI-", with string B$, which is "99/4A", use the statement PRINT A$&B$.

The string functions that are built into TI BASIC are very powerful and useful. Any function that ends in a dollar sign gives a string as a result. You cannot combine string and numeric expressions.

ASC

ASC(X$) returns the ASCII character code of the first character in the string X$. If the string expression is a constant, it must be contained in quotation marks:

PRINT ASC("*")
PRINT ASC("B")

This program returns the ASCII code of any character you enter.

```
100 REM ASC
110 CALL CLEAR
120 PRINT "WANT TO KNOW THE ASCII CODE?"::
130 INPUT "WHAT CHARACTER? ":C$
140 PRINT "ASCII CODE = ";ASC(C$)::
150 GOTO 130
160 END
```

CHR$

CHR$($x$) returns the *character* for the ASCII code x. If x is not an integer, it is rounded to obtain an integer. Try these commands:

PRINT CHR$(42)

PRINT CHR$(66)

PRINT CHR$(65+4)

The CALL KEY command returns an ASCII code number for the key pressed. If you wish to print the key pressed, the ASCII code first needs to be translated to the character which corresponds to the number. Here is a program using CHR$.

```
100 REM CHR$
110 CALL CLEAR
120 PRINT ::"PRESS ANY KEY."
130 CALL KEY(0,K,S)
140 IF S<>1 THEN 130
150 PRINT CHR$(K)
160 GOTO 120
170 END
```

In this next program, you can enter a value and get the character which corresponds to the number.

```
100 REM CHR$ 2
110 CALL CLEAR
120 INPUT "ENTER A NUMBER: ":N
130 IF N>=0 THEN 170
140 PRINT :"SORRY, NUMBER MUST BE"
150 PRINT "GREATER THAN ZERO."::
160 GOTO 120
170 IF N<=32767 THEN 210
180 PRINT :"SORRY, NUMBER MUST BE"
190 PRINT "LESS THAN 32767."::
200 GOTO 120
210 PRINT :"CHARACTER = ";CHR$(N)::
220 GOTO 120
230 END
```

STR$

There are times when you need to manipulate numbers as numeric expressions *and* as string expressions. For example, if you want to combine a name and an age, the name is a string and the age is a number. To concatenate the name and age, you will first need to convert the age number to a string, then combine the two strings. STR$(*x*) will convert the number *x* to a string. If *x* is an expression, the expression is evaluated first, then the result is converted to a string. The string will be the number only, with no leading or trailing spaces.

VAL

VAL(X$) will give the numeric value of the string X$. In this case X$ must be the ASCII characters for a number or a numeric expression. If strings contain numbers that you wish to use in calculations, the strings must first be converted to numbers with the VAL statement.

Some valid commands are:

```
PRINT STR$(529)

A$=N$&STR$(N)

M$=STR$(COST)&"/"&STR$(X)

A=VAL(A$)

PRINT VAL("27"&".45")

PRINT STR$(VAL(M$) )
```

LEN

LEN(X$) is a string function which gives the length of, or number of characters in, the string X$. In TI BASIC you may have a null string ""; the length of a null string is zero. Leading and trailing blank spaces are counted in the number of characters for the length.

POS

POS(*string*1,*string*2,*n*) is the position function. *String*1 and *string*2 are string expressions. The numeric expression *n* is evaluated and rounded to an integer. POS finds the first occurrence of *string*2 within *string*1, starting at character number *n*. The value returned is the character position of the first character of *string*2 in *string*1. If *string*2 is not found, a value of zero is returned.

Perhaps the best way to explain this function is with some examples. Run the following program. P=POS(B$,A$,1) finds the first occurrence of A$ in B$ starting with the first character of B$. The number P is the position, or the number of characters in from the first character. P=POS(B$,A$,4) finds the first occurrence of A$ in B$, starting at the fourth character of B$.

```
100 REM POS
110 CALL CLEAR
120 PRINT "A$","B$";TAB(26);"P"
130 A$="X"
140 B$="BOXES"
150 P=POS(B$,A$,1)
160 PRINT ::"P=POS(B$,A$,1)"
170 PRINT :A$,B$;TAB(26);P
180 A$="BOB"
190 B$="BOBBY"
200 P=POS(B$,A$,1)
210 PRINT :A$,B$;TAB(26);P
220 A$="B"
230 P=POS(B$,A$,1)
240 PRINT :A$,B$;TAB(26);P
250 PRINT ::"P=POS(B$,A$,4)"
260 P=POS(B$,A$,4)
270 PRINT :A$,B$;TAB(26);P
280 A$="X"
290 P=POS(B$,A$,4)
300 PRINT :A$,B$;TAB(26);P
310 END
```

SEG$

SEG$ (*string expression,numeric expression1,numeric expression2)* is the TI BASIC string *segment* function, and is comparable to the LEFT$, MID$, and RIGHT$ functions in BASIC on some other microcomputers. The command PRINT SEG$(A$,N1,N2) will print a segment of string A$ starting with the character in the N1 position, continuing until the segment is N2 characters long.

Here are some examples.

```
100 REM SEG
110 CALL CLEAR
120 A$="HERE IS A MESSAGE."
130 PRINT A$::
140 PRINT :"SEG$(A$,1,4)";TAB(22);SEG$(A$,
    1,4)
150 PRINT :"SEG$(A$,3,5)";TAB(22);SEG$(A$,
    3,5)
160 PRINT :"SEG$(A$,12,3)";TAB(22);SEG$(A$
    ,12,3)
170 PRINT :"SEG$(A$,12,12)";TAB(22);SEG$(A
    $,12,12)
180 PRINT :"SEG$(A$,20,3)";TAB(22);SEG$(A$
    ,20,3)
190 PRINT :"SEG$(A$,LEN(A$)-4,5)";TAB(22);
    SEG$(A$,LEN(A$)-4,5)
200 END
```

String Functions in Practice

Following are several programs or partial programs that illustrate the use of these string functions.

You may want to combine graphics and text on a screen. A PRINT statement will print a message, but will scroll. HCHAR or VCHAR statements are slightly slower, but will not scroll the screen. Here is a subroutine (in lines 280-310) that allows you to print a message (M$) on a certain row (ROW), starting in column number COL+1.

```
100 REM HCHAR MESSAGE
110 CALL CLEAR
120 M$="MESSAGE"
130 ROW=10
140 COL=15
150 GOSUB 280
160 M$="EXAMPLE"
170 ROW=15
180 COL=3
```

```
190 GOSUB 280
200 M$="HELLO"
210 COL=18
220 GOSUB 280
230 M$="TRY YOUR OWN!"
240 ROW=6
250 COL=4
260 GOSUB 280
270 STOP
280 FOR I=1 TO LEN(M$)
290 CALL HCHAR(ROW,COL+I,ASC(SEG$(M$,I,1)))
300 NEXT I
310 RETURN
320 END
```

Many word puzzle games award points for using particular letters in a word. Each letter of the alphabet is given a value, such as A = 15, B = 25, C = 30, D = 21, etc. The point value of the word is calculated by adding up the individual values of the letters in the word. For example, the word CAB would be worth 30 + 15 + 25, for a total of 70.

Here is a program to calculate the value of a word after you've entered the values for each letter.

Program 5-3. Letter Puzzles

```
100 REM  LETTER PUZZLES
110 DIM V(26)
120 CALL CLEAR
130 PRINT "ENTER THE VALUE FOR EACH
    {4 SPACES}LETTER."::
140 FOR A=65 TO 90
150 PRINT CHR$(A)&"  ";
160 INPUT V(A-64)
170 NEXT A
180 PRINT :::"NOW ENTER A WORD"
190 INPUT W$
200 T=0
210 FOR I=1 TO LEN(W$)
220 L$=SEG$(W$,I,1)
230 A=ASC(L$)
240 IF A>64 THEN 270
```

```
250 PRINT :"PLEASE USE LETTERS ONLY."::
260 GOTO 190
270 IF A>90 THEN 250
280 T=T+V(A-65)
290 NEXT I
300 PRINT :"TOTAL VALUE OF WORD IS";T
310 GOTO 180
320 END
```

Bingo

There is a variation on Bingo in which each letter of the alphabet has a value. You are given a 5×5 square and may write the word *bingo* diagonally or in any column. You must then fill in the rest of the squares to make five five-letter words that include the letters of *bingo* where you placed them. The object is to find words that use high-value letters; your score is the total of the five word values.

The computer can be used to find high-scoring words. The following program gives high-scoring words for the game. Line 160 is a DATA statement. Change this statement to READ the point values of each letter of the alphabet, in order, for your particular contest.

Lines 360-870 are DATA statements that contain five-letter words. Most contests require you to use a certain dictionary. In preparing the program for play, you should go through the dictionary to find all the five-letter words that qualify, and that contain the letters B, I, N, G, or O; then type these words in the DATA statements. The last word in the list should be ZZZZ.

Warning: The list in this program may not be inclusive. Also, this list only includes words starting with A through H.

When you run the program, you will be asked for a letter. Type in *B* and press ENTER. The computer will find all words which start with *B* and total the values of the letters. It will print the first word it comes to, and its score; from that point on, it will print only words with higher totals than those already found.

Next, the computer will find all words with the letter *B* in the second position, then in the third position, and so forth.

Run the program again and enter *I*. For each run, the computer will search for a different letter that you INPUT.

The values for each letter of the alphabet are read in as data in an array L(I). Lines 190-340 perform a loop for each of the five positions in the word. A word is READ in from DATA. If the letter in the particular position is not equal to the letter you had requested, then the next word is read. If the letter is the one being searched for, the total value of the word is calculated by adding the values for each letter in the word (lines 260-290). *T* is the total. SEG$ finds out the individual letter, then ASC gets the ASCII value of the letter. Since the ASCII value of A is 65, and each letter has a corresponding ASCII code in order, the program subtracts 64 from the ASCII value of the letter in the word. *L* gives the value of the particular letter. *TT* is the high total so far.

Program 5-4. Bingo

```
100 REM  BINGO A-H
110 DIM L(26)
120 FOR I=1 TO 26
130 READ N
140 L(I)=N
150 NEXT I
160 DATA 32,17,31,13,14,15,18,33,29,30,11,1
    6,19,28,20,12,34,23,26,10,21,22,35,24
    ,25,27
170 CALL CLEAR
180 INPUT "LETTER ":A$
190 FOR I=1 TO 5
200 PRINT
210 TT=0
220 RESTORE 360
230 READ W$
240 IF W$="ZZZZZ" THEN 340
250 IF SEG$(W$,I,1)<>A$ THEN 230
260 T=0
270 FOR J=1 TO 5
280 T=T+L(ASC(SEG$(W$,J,1))-64)
290 NEXT J
300 IF T<TT THEN 230
310 PRINT T;"{3 SPACES}";W$
320 TT=T
330 GOTO 230
340 NEXT I
350 STOP
```

```
360 DATA ABACK,ABAFT,ABASE,ABASH,ABATE,ABBE
    Y,ABBOT,ABEAM,ABHOR,ABIDE,ABODE,ABORT
    ,ABOUT,ABOVE,ABSTR,ABUSE,ABYSM
370 DATA ABYSS,ACORN,ACRID,ACTOR,ADAGE,ADDN
    L,ADIEU,ADIOS,ADMAN,ADMIN,ADMIT,ADMIX
    ,ADOBE,ADOPT,ADORE,ADORN,AEGIS
380 DATA AERIE,AFFIX,AFIRE,AFOOT,AGAIN,AGEN
    T,AGILE,AGLOW,AGONY,AGORA,AGREE,AISLE
    ,ALBUM,ALIAS,ALIBI,ALIEN,ALIGN
390 DATA ALIKE,ALIVE,ALLOT,ALLOW,ALLOY,ALOF
    T,ALOHA,ALONE,ALONG,ALOOF,ALOUD,AMAIN
    ,AMBER,AMBLE,AMEBA,AMEND,AMISS
400 DATA AMITY,AMONG,AMOUR,ANENT,ANGEL,ANGE
    R,ANGLE,ANGLO,ANGRY,ANGST,ANGUS,ANION
    ,ANISE,ANKLE,ANNEX,ANNOY,ANNUL
410 DATA ANODE,ANTIC,ANVIL,AORTA,APHID,APHI
    S,APORT,APRIL,APRON,ARBOR,ARDOR,ARENA
    ,ARGON,ARGOT,ARGUE,ARISE,ARITH
420 DATA ARMOR,AROMA,ARROW,ARSON,ASCOT,ASHE
    N,ASIAN,ASIDE,ASPEN,ASPIC,ASSOC,ASTIR
    ,ATILT,ATOLL,ATONE,ATTIC,AUDIO
430 DATA AUDIT,AUGER,AUGUR,AUXIN,AVAIL,AVGA
    S,AVIAN,AVOID,AXIAL,AXIOM,BABEL,BACON
    ,BADGE,BAGEL,BAGGY,BAIRN,BAIZA
440 DATA BAIZE,BALKY,BALMY,BALSA,BANAL,BAND
    Y,BANJO,BANNS,BANTU,BARGE,BARON,BASAL
    ,BASIC,BASIL,BASIN,BASIS,BASSO
450 DATA BASTE,BATCH,BATHE,BATIK,BATON,BATT
    Y,BAWDY,BAYOU,BEACH,BEANO,BEARD,BEAST
    ,BEECH,BEEFY,BEFIT,BEFOG,BEGET
460 DATA BEGUM,BEIGE,BEING,BELAY,BELCH,BELI
    E,BELLE,BELLS,BELLY,BELOW,BENCH,BENNY
    ,BERET,BERRY,BERTH,BERYL,BESET
470 DATA BESOM,BESOT,BETEL,BEVEL,BEZEL,BIBL
    E,BIDDY,BIDET,BIGHT,BIGOT,BILGE,BILLY
    ,BINGE,BIPED,BIRCH,BIRTH,BISON
480 DATA BITCH,BLACK,BLADE,BLAIN,BLAME,BLAN
    D,BLANK,BLARE,BLASE,BLAST,BLAZE,BLEAK
    ,BLEAR,BLEAT,BLEED,BLEND,BLESS
490 DATA BLIMP,BLIND,BLINK,BLISS,BLITZ,BLOA
    T,BLOCK,BLOND,BLOOD,BLOOM,BLOWY,BLUES
    ,BLUET,BLUFF,BLUNT,BLURB,BLURT
```

```
500 DATA BLUSH,BOARD,BOAST,BOBBY,BOGEY,BOGU
    S,BOLUS,BONER,BONGO,BONNY,BONUS,BONZE
    ,BOOBY,BOOST,BOOTH,BOOTY,BOOZE
510 DATA BORAX,BORNE,BORON,BOSKY,BOSOM,BOSU
    N,BOTCH,BOUGH,BOULE,BOUND,BOURN,BOWEL
    ,BOWER,BOXER,BRACE,BRACT,BRAID
520 DATA BRAIN,BRAKE,BRAND,BRASH,BRASS,BRAV
    E,BRAVO,BRAWL,BRAWN,BRAZE,BREAD,BREAK
    ,BREAM,BREED,BRIAR,BRIBE,BRICK
530 DATA BRIDE,BRIEF,BRIER,BRINE,BRING,BRIN
    K,BRISK,BROAD,BROIL,BROKE,BROOD,BROOK
    ,BROOM,BROTH,BROWN,BRUIN,BRUIT
540 DATA BRUNT,BRUSH,BRUTE,BUDDY,BUDGE,BUGG
    Y,BUGLE,BUILD,BULGE,BULKY,BULLY,BUNCH
    ,BUNCO,BUNNY,BURGH,BURLY,BURRO
550 DATA BURST,BUSBY,BUTTE,BUTUT,BUXOM,BYLA
    W,BYWAY,CABAL,CABBY,CABIN,CABLE,CACAO
    ,CADGE,CAGEY,CAIRN,CAMEO,CANAL
560 DATA CANDY,CANNA,CANNY,CANOE,CANON,CANT
    O,CARGO,CAROL,CAROM,CAVIL,CELLO,CHAIN
    ,CHAIR,CHANT,CHAOS,CHIAO,CHICK
570 DATA CHIDE,CHIEF,CHILD,CHILI,CHILL,CHIM
    E,CHIMP,CHINA,CHINE,CHINK,CHINO,CHIRP
    ,CHIVE,CHOCK,CHOIR,CHOKE,CHOMP
580 DATA CHOPS,CHORD,CHORE,CHOSE,CHRON,CHUN
    K,CHURN,CIDER,CIGAR,CINCH,CIRCA,CIVET
    ,CIVIC,CIVIL,CLAIM,CLANG,CLANK
590 DATA CLEAN,CLICK,CLIFF,CLIMB,CLIME,CLIN
    G,CLINK,CLOAK,CLOCK,CLOSE,CLOTH,CLOUD
    ,CLOUT,CLOVE,CLOWN,CLUNG,COACH
600 DATA COAST,COBRA,COCOA,CODEX,COLIC,COLO
    N,COLOR,COMBO,COMDG,COMDR,COMDT,COMER
    ,COMET,COMFY,COMIC,COMMA,COMMO
610 DATA CONCH,CONEY,CONGA,CONIC,CONST,CONT
    D,CONTG,CONTR,COPRA,CORAL,CORNY,CORPS
    ,COUCH,COUGH,COULD,COUNT,COUPE
620 DATA COURT,COVEN,COVER,COVET,COVEY,COWE
    R,COYPU,COZEN,CRANE,CRANK,CRICK,CRIER
    ,CRIME,CRIMP,CROAK,CROCK,CRONE
630 DATA CRONY,CROOK,CROON,CROSS,CROUP,CROW
    D,CROWN,CRUMB,CUBAN,CUBIC,CUBIT,CUPID
    ,CURIA,CURIO,CYNIC,DAILY,DAIRY
```

```
640 DATA DAISY,DANCE,DANDY,DAUNT,DAVIT,DEBI
    T,DEBUT,DECOR,DECOY,DEFOG,DEGAS,DEICE
    ,DEIGN,DEISM,DEITY,DEMON,DENIM
650 DATA DENSE,DEPOT,DERBY,DERIV,DEVIL,DIAR
    Y,DIGIT,DINAR,DINGO,DINGY,DINKY,DIODE
    ,DIRGE,DIRTY,DISCO,DISTN,DISTR
660 DATA DITCH,DITTO,DITTY,DIVAN,DIVOT,DIZZ
    Y,DLITT,DODGE,DODOS,DOGGY,DOGIE,DOGMA
    ,DOILY,DOLLY,DOLOR,DONOR,DONUT
670 DATA DOPEY,DOUBT,DOUGH,DOUSE,DOWDY,DOWE
    L,DOWER,DOWNY,DOWRY,DOWSE,DOYEN,DOZEN
    ,DRAIN,DRANK,DRIER,DRIFT,DRILL
680 DATA DRILY,DRINK,DRIVE,DROLL,DRONE,DROO
    L,DROOP,DROSS,DROVE,DROWN,DRUID,DRUNK
    ,DUNCE,DYING,EAGER,EAGLE,EBONY
690 DATA EDICT,EERIE,EGRET,EIDER,EIGHT,ELAN
    D,ELBOW,ELEGY,ELIDE,ELITE,ELOPE,EMBED
    ,EMBER,EMEND,EMOTE,ENACT,ENDOW
700 DATA ENDUE,ENEMA,ENEMY,ENJOY,ENNUI,ENTE
    R,ENTOM,ENTRY,ENVOI,ENVOY,EPOCH,EPOXY
    ,EQUIP,EQUIV,ERGOT,ERODE,ERROR
710 DATA ETHOS,EVICT,EXIST,EXILE,EXTOL,EXUR
    B,EYRIE,EYRIR,FABLE,FAGOT,FAINT,FAIRY
    ,FAITH,FAKIR,FANCY,FAUNA,FAVOR
720 DATA FEIGN,FEINT,FENCE,FETID,FIBER,FICH
    E,FIELD,FIEND,FIERY,FIFTH,FIGHT,FILAR
    ,FILCH,FILLY,FILTH,FINAL,FINCH
730 DATA FINIS,FINNY,FIORD,FIRST,FIRTH,FISH
    Y,FIXED,FJORD,FLAIL,FLAIR,FLANK,FLICK
    ,FLIED,FLIER,FLING,FLINT,FLIRT
740 DATA FLOAT,FLOCK,FLOOD,FLOOR,FLORA,FLOS
    S,FLOUR,FLOUT,FLOWN,FLUID,FLUNG,FLUNK
    ,FLYBY,FOCUS,FOEHN,FOIST,FOLIO
750 DATA FOLLY,FORAY,FORCE,FORGE,FORGO,FORT
    E,FORTH,FORTY,FORUM,FOUND,FOUNT,FOXED
    ,FOYER,FRAIL,FRANC,FRANK,FRIAR
760 DATA FRILL,FRISK,FRIZZ,FROCK,FROND,FRON
    T,FROST,FROTH,FROWN,FROZE,FRUIT,FSLIC
    ,FUDGE,FUGUE,FUNKY,FUNNY,FUROR
770 DATA GABBY,GABLE,GAFFE,GAILY,GAMIN,GAMU
    T,GAUDY,GAUGE,GAUNT,GAUSS,GAUZE,GAVEL
    ,GAWKY,GEESE,GENIE,GENRE,GENUS
```

```
780 DATA GEODE,GETUP,GHOST,GHOUL,GIANT,GIDD
    Y,GIMPY,GIPSY,GIRTH,GIVEN,GIZMO,GLADE
    ,GLAND,GLANS,GLARE,GLASS,GLAZE
790 DATA GLEAM,GLEAN,GLEBE,GLIDE,GLINT,GLOA
    T,GLOBE,GLOOM,GLORY,GLOSS,GLOVE,GLOZE
    ,GNARL,GNASH,GNOME,GODLY,GONAD
800 DATA GOODY,GOOFY,GOOSE,GORGE,GORSE,GOUD
    A,GOUGE,GOURD,GRACE,GRADE,GRAFT,GRAIL
    ,GRAIN,GRAND,GRANT,GRAPE,GRAPH
810 DATA GRASP,GRASS,GRATE,GRAVE,GRAVY,GRAZ
    E,GREAT,GREBE,GREED,GREEK,GREEN,GREET
    ,GRIEF,GRILL,GRIME,GRIND,GRIPE
820 DATA GRIST,GRITS,GROAN,GROAT,GROIN,GROO
    M,GROPE,GROSS,GROSZ,GROUP,GROUT,GROVE
    ,GROWL,GRUEL,GRUFF,GRUNT,GUANO
830 DATA GUARD,GUAVA,GUESS,GUEST,GUIDE,GUIL
    D,GUILE,GUILT,GUISE,GULCH,GULLY,GUMBO
    ,GUNNY,GUPPY,GUSHY,GUSTO,GUTTY
840 DATA GUYOT,GYPSY,HABIT,HAIKU,HAIRY,HAJJ
    I,HALLO,HANDY,HAOLE,HAUNT,HAVEN,HAVOC
    ,HEDGE,HEIST,HELIX,HELLO,HELOT
850 DATA HENCE,HENNA,HERON,HINGE,HITCH,HIVE
    S,HOARD,HOARY,HOBBY,HOGAN,HOIST,HOKUM
    ,HOLLO,HOLLY,HOMER,HOMEY,HONEY
860 DATA HONOR,HOOEY,HORDE,HORSE,HOTEL,HOUN
    D,HOURI,HOUSE,HOVEL,HOVER,HUMAN,HUMID
    ,HUMOR,HUNCH,HURON,HYENA,HYMEN
870 DATA AWAIT,BEGIN,CARNY,COCKY,COPSE,CRIS
    P,DINER,ENSUE,EVENT,EVOKE,FICHU,FIFTY
    ,HINDI,HYDRO,ZZZZZ
880 END
```

Birthday List

Here is a program that keeps track of birthdays. The list can be printed either by name, in alphabetical order, or by birthday, in calendar order.

The DATA statements at the end of the main program contain the names in alphabetical order by last name (you may prefer to arrange the data in order by family). The number following the name is the birthdate as a four-digit number. The first two digits stand for the number of the month, and the last two digits are the day. If either is unknown, the number should be entered as 00. For example, a birthday of November 14 would be listed as 1114 — 11th month, 14th day. A birthday sometime in May would be 0500 — fifth month, unknown day.

After the names are listed in alphabetical order, there is a delay while the names are sorted by birthday; then the list is printed by date with a double space between months.

If you have more than 30 names, increase the parameters in the DIM statement in line 110. You may also want to change to a faster sort routine. N$ is the name and B$ is the birthday. M$ is an array that holds the month names. SEG$ looks at either the first two digits or the last two digits of the birthday code. VAL gives the numerical value of the string number.

How "Birthday List" Works

Lines	
110	DIMension the name array, the birthday array, and the month array.
120-130	Print title.
140-180	READ month names into M$ array.
190	L is a counter for the number of lines on the screen.
200	READ last name LN$, first name FN$, and birthday B$.
210	Branch if LN$ is the last item on the data list.
220	Print the last name and first name on the screen.
230-260	Print the day and month of the birthday.
270	Combine first name and last name as N$ in array.
280-290	Increment the counter in name array and the counter in number of lines printed on screen.
300-350	If the screen is filled, press any key to continue.
360-380	Print the message; wait for key to be pressed.
390-410	Clear the screen and print the title.
420-550	Birthday sort routine.
560-570	Clear the message and initialize the line count.
580-750	Print the day, month, and name. Double-space if the months are different; keep track of the number of lines printed so the names don't scroll off the screen.

760	End of main program logic.
770-1090	Sample DATA. These names and dates are fictional.

Program 5-5. Birthday List

```
100 REM  BIRTHDAY LIST
110 DIM N$(30),B$(30),M$(12)
120 CALL CLEAR
130 PRINT TAB(6);"BIRTHDAY LIST":::
140 DATA ???,JAN,FEB,MAR,APR,MAY,JUN,JUL
150 DATA AUG,SEP,OCT,NOV,DEC
160 FOR I=0 TO 12
170 READ M$(I)
180 NEXT I
190 L=1
200 READ LN$,FN$,B$(J)
210 IF LN$="ZZZ" THEN 360
220 PRINT LN$;", ";FN$;TAB(20);
230 DAY$=SEG$(B$(J),3,2)
240 IF DAY$<>"00" THEN 260
250 DAY$="??"
260 PRINT DAY$;" ";M$(VAL(SEG$(B$(J),1,2)))
270 N$(J)=FN$&" "&LN$
280 J=J+1
290 L=L+1
300 IF L<18 THEN 200
310 PRINT :"PRESS ANY KEY TO CONTINUE.";
320 CALL KEY(0,K,S)
330 IF S<1 THEN 320
340 CALL CLEAR
350 GOTO 190
360 PRINT ::"PRESS ANY KEY FOR NEXT LIST.";
370 CALL KEY(0,K,S)
380 IF S<1 THEN 370
390 CALL CLEAR
400 PRINT TAB(6);"BIRTHDAY LIST":::
410 PRINT "--SORTING--";
420 LIM=J-2
430 SW=0
440 FOR K=0 TO LIM
```

```
450 IF VAL(B$(K))<=VAL(B$(K+1))THEN 540
460 BB$=B$(K)
470 NN$=N$(K)
480 B$(K)=B$(K+1)
490 N$(K)=N$(K+1)
500 B$(K+1)=BB$
510 N$(K+1)=NN$
520 SW=1
530 LIM=K
540 NEXT K
550 IF SW=1 THEN 430
560 CALL HCHAR(24,3,32,28)
570 L=0
580 FOR KK=0 TO J-1
590 DAY$=SEG$(B$(KK),3,2)
600 MON$=M$(VAL(SEG$(B$(KK),1,2)))
610 IF MON$=MON1$ THEN 640
620 PRINT
630 L=L+1
640 IF DAY$<>"00" THEN 660
650 DAY$="??"
660 PRINT DAY$;" ";MON$;"{3 SPACES}";N$(KK)
670 MON1$=MON$
680 L=L+1
690 IF L<18 THEN 750
700 PRINT :"PRESS A KEY TO CONTINUE.";
710 CALL KEY(0,K,S)
720 IF S<1 THEN 710
730 CALL CLEAR
740 L=0
750 NEXT KK
760 STOP
770 REM  SAMPLE DATA
780 DATA ADAMS,LEWIS,0000
790 DATA BAKER,MELISSA,1112
800 DATA CHILD,ED,0830
810 DATA DAINES,BILL,0520
820 DATA EVANS,JOHN,0415
830 DATA EVANS,JIM,1000
840 DATA JONES,DOUG,1115
850 DATA NELSON,ANDY,0500
860 DATA NELSON,LENA,0700
```

```
870 DATA NELSON,SHEILA,1115
880 DATA PETERSON,GRANT,0400
890 DATA PETERSON,ROGER,1005
900 DATA PETERSON,SHERYL,0618
910 DATA S,GRANDMA,0815
920 DATA S,GRANDPA,1017
930 DATA SMITH,BOBBY,0510
940 DATA SMITH,CHARLES,0611
950 DATA SMITH,CHRISTY,1115
960 DATA SMITH,CHERY,0802
970 DATA SMITH,CINDY,0415
980 DATA SMITH,RANDY,0302
990 DATA SMITH,RICHARD,0509
1000 DATA W,GRANDMA,1120
1010 DATA W,GRANDPA,0221
1020 DATA WHITE,ANGELA,0000
1030 DATA WHITE,BRYAN,0700
1040 DATA WHITE,DEAN,0104
1050 DATA WHITE,JENNIE,0200
1060 DATA WHITE,KELLY,1014
1070 DATA WHITE,RELLE,0928
1080 DATA ZZZ,Z,0000
1090 END
```

Chapter 6

Programming Techniques

Chapter 6

Programming Techniques

If you ask ten different programmers to write a basic home inventory program, you'll get ten different programs. In computer programming, many different methods accomplish the same thing. The "correct" method is the one that works — the program that will run without a bug.

The programmer often has to make a choice — one way of solving a problem may be easier for the user to understand but will take more memory than another method, while a third method may execute more efficiently than either of the first two.

Arrays

Memory locations are like a wall full of post office boxes, each with its own name. Each address holds a value for a variable name. For example, suppose we have the beginning of a program:

```
100 A=3
110 B=4
120 X=10
```

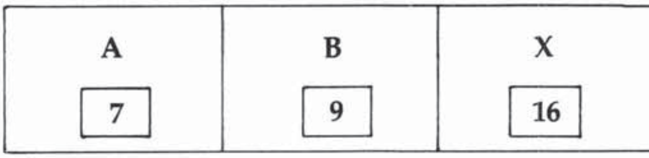
The boxes would look like this:

A	B	X
3	4	10

Later in the program you may change the values:

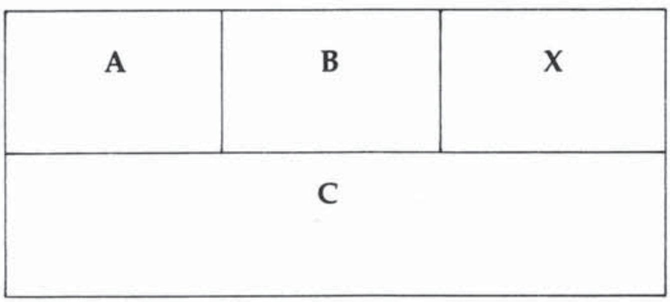
```
200 A=7
210 B=A+2
220 X=A+B
```

The values in the boxes change; they become:

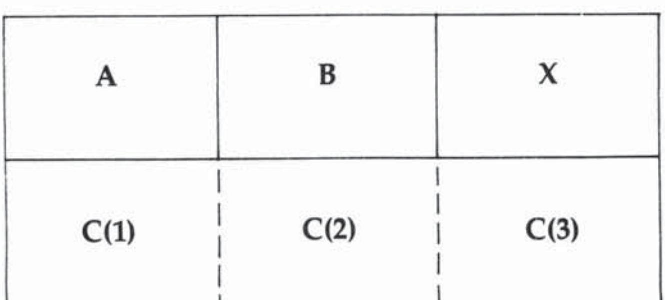


Each of these boxes has a name, and each name has only one box.

Now, just like in the post office, some boxes are bigger than others.



The C box can be divided up into smaller parts, but they are still parts of C. In this case, the C box holds an array, and different values can go into each particular part of C. We specify each part of C with a subscript, a number in parentheses. So the names of the *elements of the array* C are C(1), C(2), and C(3).



Boxes can be even larger — representing one, two, or three dimensions in TI BASIC. Here is a chart of *D,* which has two dimensions, one with two elements (first subscript), the other with four (second subscript).

A	**B**	**X**	**D(1,1)**	**D(1,2)**	**D(1,3)**	**D(1,4)**
C(1)	**C(2)**	**C(3)**	**D(2,1)**	**D(2,2)**	**D(2,3)**	**D(2,4)**

Arrays can make a repetitive computer program more efficient. If you do a process several times, it may be worth using a variable with a subscript. Suppose you are describing three boys. Their names are Richard, Robert, and Randy. We can say:

NAME$(1)="RICHARD"
NAME$(2)="ROBERT"
NAME$(3)="RANDY"

Now we wish to list some things about these people:

AGE(1)=11
AGE(2)=6
AGE(3)=9

COLOR$(1)="BLACK"
COLOR$(2)="BLUE"
COLOR$(3)="RED"

SPORT$(1)="FOOTBALL"
SPORT$(2)="BASEBALL"
SPORT$(3)="BASKETBALL"

You can print a list of the boys by using a single loop and a variable subscript:

```
200 FOR J=1 TO 3
210 PRINT NAME$(J);AGE(J);SPORT$(J)
220 NEXT J
```

If you wish to know about a particular person, print only his information by searching the arrays for a particular subscript.

```
300 N=2
310 PRINT NAME$(N),COLOR$(N)
```

If you have a longer list, you could sort. To find all the boys with an age of 6, and there are a total number (T) of boys:

```
400 FOR J=1 TO T
410 IF AGE(J)<>6 THEN 430
420 PRINT NAMES$(J)
430 NEXT J
```

The computer will only execute line 420, PRINT NAME$(J), when the value of AGE(J) is 6.

This information about the boys could be in a two-dimensional array rather than in the four one-dimensional arrays above. Call the main array PERSON$. The data may be arranged like this:

PERSON$(1,1)="RICHARD"
PERSON$(1,2)="11"
PERSON$(1,3)="BLACK"
PERSON$(1,4)="FOOTBALL"

PERSON$(2,1)="ROBERT"
PERSON$(2,2)="6"
PERSON$(2,3)="BLUE"
PERSON$(2,4)="BASEBALL"

PERSON$(3,1)="RANDY"
PERSON$(3,2)="2"
PERSON$(3,3)="RED"
PERSON$(3,4)="BASKETBALL"

The first subscript tells us *which boy's* data is held in that variable, and the second subscript identifies the category of information. The word or number in quotation marks is the string placed in each address of our post office boxes.

In TI BASIC, both numeric variables and string variables may be arrays. You may not use the same variable name for subscripted and non-subscripted variables. For example, you may not use A and A(3) in the same program.

If you use a variable name with a subscript without first DIMensioning that variable, the computer automatically reserves eleven elements for the array. If you need more than eleven, use a DIM statement to clear enough space:

```
100 DIM D(30)
```

If your program is running nearly full memory and you do not need all eleven elements, you can save memory by DIMensioning the array for fewer elements:

```
100 DIM A(6)
```

If you have a two- or three-dimensional array, you must specify how many locations you want to reserve in each dimension.

```
100 DIM F(4,5,10)
```

The DIMension statement must appear before any reference to the array; it is wise to put all DIMension statements near the beginning of the program.

The computer automatically starts numbering all subscripts with zero. In other words, there can be elements such as D(0) and E(1,0). Since the zero variable counts as one element, a statement like DIM A(10) reserves *eleven* subscripted variables, A(0) through A(10). If you prefer to use only elements numbered 1 and above, you may use the OPTION BASE statement:

```
100 OPTION BASE 1
110 DIM A(10)
```

Now there will only be ten variables reserved, A(1) through A(10).

Edible Arrays

"Cookie File" illustrates the use of arrays. This program uses a data structure to keep a file of cookie recipes. You may select a cookie recipe from the menu screen, then that recipe will be printed on the screen along with a picture of the cookie type. If you choose to convert the recipe (double, triple, or halve it, for example), enter a multiplication factor, and the converted recipe is printed. Another option of this program is to indicate on an inventory list which ingredients you have and which you do not have. The computer will then report which cookies can be made with the ingredients you have.

Line 380 DIMensions ING$(19) for a list of ingredients and INV$(19,1) for an inventory list. Subscripts start at zero. Lines 390-420 READ, from DATA statements (lines 2260-2300), first A$, which is a measurement, and then INV$(I,0), which is an ingredient. ING$(I) is equal to the measurement combined with the ingredient as one string. This process is repeated for 20 items.

Later, in lines 1390-1470, as each ingredient is listed using INV$(K,0), the user presses Y or N. The character pressed will be stored in INV$(K,1) to make up an inventory list.

In one section of the program, the recipe is listed. Lines 1010-1020 set the amount AMT(I) and the ingredient INGR$(I) for each item of the recipe. Lines 1220-1240 convert the recipe by multiplying a factor F by the amount AMT(K) and printing the corresponding ingredient INGR$(K).

The DATA statement for each cookie is entered in the following order: title, graphics code, cups of shortening, cups of sugar, cups of brown sugar, cups of powdered sugar, tablespoons of honey, eggs, teaspoons of vanilla, cups of flour, teaspoons of baking powder, teaspoons of baking soda, teaspoons of salt, teaspoons of cinnamon, tablespoons of cocoa, teaspoons of almond extract, cups of milk, cups of oatmeal, ounces of chocolate chips, dozens of almonds, teaspoons of cake decors, cups of cinnamon sugar, and the cookies' baking temperature. If a recipe doesn't use a particular ingredient, I enter no data at all before the comma:

```
DATA ALMOND COOKIES,1,2,2,,,,2,,4,2,,,,,2,,,,4,,,375
```

This indicates that almond cookies use graphics style 1, and the recipe is 2 cups of shortening, 2 cups of sugar, 2 eggs, 4 cups of flour, 2 tsp. baking powder, 2 tsp. almond extract, and 4 dozen almonds, and the cookies bake at 375 degrees.

You can put your own recipes in this program by changing the DATA statements. Other ingredients may be added or deleted by adjusting the first DATA statements, which create the ingredient list, and the DIMension statement which creates the number of ingredients as a parameter in the arrays. You will also need to change the titles on the menu screen and the corresponding RESTORE numbers.

This program does not include mixing directions because with cookies you usually know the procedure and need only the proportions of ingredients. You could add mixing instructions by adding some codes in the DATA statements to correspond to certain print statements. An example in this program is graphics code 2, which includes the instruction "Roll in powdered sugar."

In case you wish to try some of these recipes, just mix the ingredients in order, then bake. Some of the specifics are:

Almond cookies: Roll into balls, flatten slightly, place blanched almond on top; brush with egg if desired.

Ball cookies: Drop cookies onto sheet; then flatten with ice cube or moist rag; sprinkle colored cake decors on top; bake just until golden brown around the edges.

Brownies: Melt the cocoa with the shortening first; bake in square pan.

Butterscotch bars: Melt shortening (or butter) with brown sugar; cool; then add other ingredients; bake in rectangular glass baking dish.

Chocolate chip bars: Bake in 9×13 pan.

Chocolate chip cookies: Make as drop cookies.

Chocolate drop cookies: Make as drop cookies, good with chocolate frosting.

Honey balls: Roll into balls; bake about 25 minutes; roll in powdered sugar while still warm, then again when cool.

Honey spice cookies: Make as drop cookies.

Mexican wedding cookies: Like honey balls.

Oatmeal chocolate chips: Make as drop cookies.

Oatmeal crisps: Refrigerator cookies; form into long roll; slice, then bake.

Snickerdoodles: Roll dough into balls, then roll in cinnamon and sugar mixture before baking.

Toffee bars: Press into 9×13 pan or on cookie sheet (about ½-inch thick).

Program 6-1. Cookie File

```
100 REM  COOKIE FILE
110 REM  BY REGENA
120 GOSUB 1760
130 GOTO 380
140 CALL HCHAR(22,27,137)
```

```
150 CALL HCHAR(22,28,136,2)
160 CALL HCHAR(22,30,138)
170 CALL HCHAR(21,28,128,2)
180 RETURN
190 CALL HCHAR(21,27,124)
200 CALL HCHAR(21,28,126)
210 CALL HCHAR(22,27,125)
220 CALL HCHAR(22,28,127)
230 RETURN
240 CALL HCHAR(22,26,137)
250 CALL HCHAR(22,27,136,2)
260 CALL HCHAR(22,29,138)
270 CALL HCHAR(21,27,139)
280 CALL HCHAR(21,28,140)
290 RETURN
300 CALL HCHAR(22,26,96,4)
310 CALL HCHAR(21,26,103,4)
320 RETURN
330 CALL HCHAR(22,26,129)
340 CALL HCHAR(22,27,130,2)
350 CALL HCHAR(22,29,131)
360 CALL HCHAR(21,27,103,2)
370 RETURN
380 DIM ING$(19),INV$(19,1)
390 FOR I=0 TO 19
400 READ A$,INV$(I,0)
410 ING$(I)=A$&INV$(I,0)
420 NEXT I
430 CALL CLEAR
440 CALL COLOR(2,2,1)
450 CALL COLOR(9,7,1)
460 PRINT "CHOOSE:":::"1  NEED TO KNOW WHAT
    ":"{3 SPACES}CAN BE MADE"
470 PRINT :::"2  WANT TO SEE A":"
    {3 SPACES}CERTAIN RECIPE"::::
480 PRINT "3  END PROGRAM":::
490 CALL KEY(0,KEY,S)
500 IF KEY=49 THEN 1300
510 IF KEY=51 THEN 2470
520 IF KEY<>50 THEN 490
530 CALL CLEAR
540 PRINT "CHOOSE:"::
```

```
550 PRINT "A  ALMOND COOKIES":"B  BALL COOK
    IES":"C  BROWNIES"
560 PRINT "D  BUTTERSCOTCH BARS":"E  CHOCOL
    ATE CHIP BARS":"F  CHOCOLATE CHIP COO
    KIES"
570 PRINT "G  CHOCOLATE DROP COOKIES":"H  H
    ONEY BALLS":"I  HONEY SPICE COOKIES"
580 PRINT "J  MEXICAN WEDDING COOKIES":"K
    OATMEAL CHOCOLATE CHIPS":"L  OATMEAL
    CRISPS"
590 PRINT "M  SNICKERDOODLES":"N  SUGAR COO
    KIES":"O  TOFFEE BARS"
600 CALL KEY(0,KEY,S)
610 IF (KEY<65)+(KEY>79)THEN 600
620 CALL CLEAR
630 ON KEY-64 GOTO 640,660,680,700,720,740,
    760,780,800,820,840,860,880,900,920
640 RESTORE 2310
650 GOTO 930
660 RESTORE 2320
670 GOTO 930
680 RESTORE 2330
690 GOTO 930
700 RESTORE 2340
710 GOTO 930
720 RESTORE 2350
730 GOTO 930
740 RESTORE 2360
750 GOTO 930
760 RESTORE 2370
770 GOTO 930
780 RESTORE 2380
790 GOTO 930
800 RESTORE 2390
810 GOTO 930
820 RESTORE 2400
830 GOTO 930
840 RESTORE 2410
850 GOTO 930
860 RESTORE 2420
870 GOTO 930
880 RESTORE 2430
```

```
890 GOTO 930
900 RESTORE 2440
910 GOTO 930
920 RESTORE 2450
930 READ A$,G
940 PRINT A$:::
950 ON G GOSUB 140,190,240,300,330
960 I=0
970 FOR J=0 TO 19
980 READ B$
990 IF B$="" THEN 1050
1000 IF B$="0" THEN 1050
1010 AMT(I)=VAL(B$)
1020 INGR$(I)=ING$(J)
1030 PRINT AMT(I);INGR$(I)
1040 I=I+1
1050 NEXT J
1060 READ T
1070 PRINT :"BAKE AT";T;"DEGREES."
1080 IF G<>2 THEN 1100
1090 PRINT "ROLL IN POWDERED SUGAR."
1100 PRINT :"WANT TO CONVERT RECIPE?(Y/N)"
1110 CALL KEY(0,KEY,S)
1120 IF KEY=78 THEN 1270
1130 IF KEY<>89 THEN 1110
1140 PRINT :"MULTIPLY BY WHAT NUMBER"
1150 INPUT "OR DECIMAL FRACTION? ":F
1160 IF F>0 THEN 1190
1170 PRINT :"SORRY, F>0"
1180 GOTO 1140
1190 CALL CLEAR
1200 PRINT F;"TIMES ORIGINAL RECIPE":::
1210 PRINT A$::
1220 FOR K=0 TO I-1
1230 PRINT F*AMT(K);INGR$(K)
1240 NEXT K
1250 PRINT :"CONVERT AGAIN? (Y/N)"
1260 GOTO 1110
1270 PRINT :"PRESS ANY KEY TO CONTINUE."
1280 CALL KEY(0,KEY,S)
1290 IF S=0 THEN 1280 ELSE 430
1300 CALL CLEAR
```

```
1310 PRINT "IN THE FOLLOWING LIST,"
1320 PRINT "PRESS ""Y"" IF YOU HAVE"
1330 PRINT "THE INGREDIENT."
1340 PRINT "PRESS ""N"" IF YOU DO NOT."
1350 PRINT :"PRESS ""S"" TO START OVER.":::
     :::
1360 CALL SOUND(150,1397,2)
1370 YS=0
1380 FOR K=0 TO 19
1390 PRINT "_ ";INV$(K,0)
1400 CALL KEY(0,KEY,S)
1410 IF KEY=83 THEN 1300
1420 IF KEY=78 THEN 1450
1430 IF KEY<>89 THEN 1400
1440 YS=YS+1
1450 CALL HCHAR(23,3,KEY)
1460 INV$(K,1)=CHR$(KEY)
1470 NEXT K
1480 C=0
1490 PRINT ::"YOU CAN MAKE:"::
1500 IF INV$(0,1)="N" THEN 1530
1510 IF INV$(7,1)="N" THEN 1530
1520 IF YS>4 THEN 1550
1530 PRINT "NOTHING TODAY.":"YOU NEED MORE
     SUPPLIES."
1540 GOTO 1270
1550 RESTORE 2310
1560 READ A$,G
1570 FOR J=0 TO 19
1580 READ B$
1590 IF B$="" THEN 1620
1600 IF B$="0" THEN 1620
1610 IF INV$(J,1)="N" THEN 1660
1620 NEXT J
1630 CALL SOUND(150,1397,2)
1640 PRINT A$
1650 C=C+1
1660 READ D$
1670 IF D$="ZZZ" THEN 1720
1680 IF LEN(D$)<6 THEN 1660
1690 A$=D$
1700 READ G
```

```
1710 GOTO 1570
1720 IF C=0 THEN 1530
1730 PRINT :"GO AHEAD AND BAKE!"
1740 GOTO 1270
1750 STOP
1760 CALL CLEAR
1770 CALL CHAR(96,"EFFDB7FEDBFFB7FD")
1780 CALL COLOR(2,13,13)
1790 CALL CHAR(97,"F6BCE8F0A0C08")
1800 CALL COLOR(9,16,1)
1810 PRINT "{4 SPACES}++++++++++":"
     {4 SPACES}++++++++++"
1820 PRINT "{4 SPACES}++COOKIE++":"
     {4 SPACES}++++++++++"
1830 PRINT "{4 SPACES}+++FILE+++":"
     {4 SPACES}++++++++++":"{4 SPACES}+++++
     +++++"::::
1840 CALL CHAR(98,"FEFDFBF50FDBAE7F")
1850 CALL CHAR(99,"FFFFFFF00FFFFFF")
1860 CALL CHAR(100,"0103070F003F7FFF")
1870 CALL VCHAR(12,17,98)
1880 CALL VCHAR(13,17,96,6)
1890 CALL VCHAR(19,17,97)
1900 CALL VCHAR(11,18,98)
1910 CALL VCHAR(12,18,96,6)
1920 CALL VCHAR(18,18,97)
1930 CALL VCHAR(10,19,98)
1940 CALL VCHAR(11,19,96,6)
1950 CALL VCHAR(17,19,97)
1960 CALL CHAR(101,"007E7E7E7EFFFFFF")
1970 CALL HCHAR(12,7,100)
1980 CALL HCHAR(12,8,99,9)
1990 CALL HCHAR(11,8,100)
2000 CALL HCHAR(11,9,99,9)
2010 CALL HCHAR(10,9,100)
2020 CALL HCHAR(10,10,99,9)
2030 CALL HCHAR(12,9,101)
2040 CALL HCHAR(11,11,101)
2050 CALL HCHAR(10,13,101)
2060 CALL CHAR(124,"071F3F7F7FFFFFFF")
2070 CALL CHAR(125,"FFFFFF7F7F3F1F07")
2080 CALL CHAR(126,"E0F8FCFEFEFFFFFF")
```

```
2090 CALL CHAR(127,"FFFFFFFEFEFCF8E")
2100 CALL CHAR(136,"FFFFFFFFFFFFFFF")
2110 CALL CHAR(137,"01071F3F7F7FFFFF")
2120 CALL CHAR(138,"80E0F8FCFEFEFFFF")
2130 CALL CHAR(139,"0000000030F1F7F")
2140 CALL CHAR(140,"00000000C0F0F8FE")
2150 CALL CHAR(103,"00000000000000055")
2160 CALL CHAR(128,"0000000000000003C")
2170 CALL CHAR(129,"0F3F7FFFFF")
2180 CALL CHAR(130,"FFFFFFFFFF")
2190 CALL CHAR(131,"F0FCFEFFFF")
2200 CALL COLOR(12,16,1)
2210 CALL COLOR(13,11,1)
2220 CALL COLOR(14,12,1)
2230 CALL CHAR(64,"3C4299A1A199423C")
2240 PRINT :
2250 RETURN
2260 DATA "C. ",SHORTENING,"C. ",SUGAR,"C.
     ",BROWN SUGAR,"C. ",POWDERED SUGAR,"T
     BSP. ",HONEY,"",EGGS
2270 DATA "TSP. ",VANILLA,"C. ",FLOUR,"TSP.
      ",BAKING POWDER,"TSP. ",BAKING SODA,
     "TSP. ",SALT
2280 DATA "TSP. ",CINNAMON,"TBSP. ",COCOA,"
     TSP. ",ALMOND EXTRACT,"C. ",MILK,"C.
     ",OATMEAL
2290 DATA "OZ. ",CHOCOLATE CHIPS,"DOZ. ",AL
     MONDS
2300 DATA "TSP. ","CAKE DECORS","C. ","CINN
     AMON & SUGAR"
2310 DATA ALMOND COOKIES,1,2,2,,,,2,,4,2,,,
     ,,2,,,,4,,,375
2320 DATA BALL COOKIES,5,.5,.33,,,,1,.5,.75
     ,,,,,,,,,,,2,,375
2330 DATA BROWNIES,4,.5,1,,,,2,1,.75,.5,,.5
     ,,6,,,,,,,,350
2340 DATA BUTTERSCOTCH BARS,4,.5,,2,,,2,1,1
     .75,2,,.25,,,,,,,,,,375
2350 DATA CHOCOLATE CHIP BARS,4,.5,,1,,,1,1
     ,1.75,,.5,.5,,,,,.5,,12,,,,350
2360 DATA CHOCOLATE CHIP COOKIES,3,.5,.25,.
     5,,,1,.5,1,,.5,.5,,,,,,6,,,,375
```

```
2370 DATA CHOCOLATE DROP COOKIES,3,.5,,1,,,
     1,1,1.67,,.5,.5,,6,,.5,,,,,,350
2380 DATA HONEY BALLS,2,.5,,,,2,,1,1,,,.25,
     ,,,,,,,,,300
2390 DATA HONEY SPICE COOKIES,1,.5,.75,,,4,
     ,.5,1,,,,.5,,,,,,,,,375
2400 DATA MEXICAN WEDDING COOKIES,2,.75,,,.
     67,,,1,1.5,,,.25,1,,,,.75,,,,,325
2410 DATA OATMEAL CHOCOLATE CHIPS,3,1,1,.5,
     ,,2,1,2,,1,1,,,,,2,6,,,,350
2420 DATA OATMEAL CRISPS,1,1,1,1,,,2,1,1.5,
     ,1,1,,,,,3,,,,,350
2430 DATA SNICKERDOODLES,1,1,1.5,,,,2,,2.75
     ,3,,.5,,,,,,,,,,.5,400
2440 DATA SUGAR COOKIES,5,.67,.75,,,,1,.5,2
     ,1.5,,.25,,,,.25,,,,,,375
2450 DATA TOFFEE BARS,4,1,,1,,,,1,2,,,,,,,,
     ,6,,,,350
2460 DATA ZZZ
2470 CALL CLEAR
2480 END
```

DATA Statements

DATA statements contain numbers or strings or both, and may be placed anywhere in your program. They are ignored until the computer comes to a READ statement; then the computer finds the first DATA statement and READs the appropriate number of items.

If the computer encounters another READ statement, it goes to the very next data item, whether it's in the same DATA statement or in the next DATA statement, and continues to READ in order. All items are separated by commas.

```
100 REM DATA 1
110 FOR I=1 TO 5
120 READ A,B
130 PRINT ::A;"+";B;"=";A+B
140 NEXT I
150 DATA 1,2,3,4,10,13,11,5,23,45
160 END
```

When you RUN this program, the results are:

```
    1+2=3
    3+4=7
    10+13=23
    11+5=16
    23+45=68
** DONE **
```

The first time through the loop, A will be 1 and B will be 2; the second time, A will be 3 and B will be 4, and so forth. You can see that a DATA statement is more efficient (as far as amount of memory used) to get a lot of numbers into the computer than a number of LET statements. With DATA statements, you do need to be careful that commas are in the right places, that the DATA items match the READ statements, and that there is sufficient data for the number of items in the READ statements. If READ can't find any more DATA, the program crashes.

In TI BASIC, strings in DATA statements do not need to be in quotation marks unless there are leading or trailing spaces or commas within the string. An example of a DATA statement using strings is

```
300 DATA GEORGE,HENRY,932 EVERGREEN,"PROVO, UTAH"
```

Working with RESTORE

One of the most useful commands in working with data is the RESTORE statement — it makes it much easier to keep track of where your data lists start. Ordinarily, the computer goes straight through the DATA statements in order, as needed by the READ statements. RESTORE, used without any parameters, will start the data list all over again with the first DATA statement.

Suppose I want to use the same list of numbers in two operations. Instead of having identical DATA statements, I finish the first operation, use RESTORE, and start over on the data list for the second operation.

```
100 REM DATA 2
110 FOR I=1 TO 5
120 READ A,B
```

```
130 PRINT :A;"+";B;"=";A+B
140 NEXT I
150 DATA 1,2,3,4,10,13,11,5,23,45
160 RESTORE
170 FOR I=1 TO 5
180 READ A,B
190 PRINT :A;"*";B;"=";A*B
200 NEXT I
210 END
```

RESTORE with Parameters

The nicest thing about the RESTORE statement is that you do not have to RESTORE back to the beginning of the very first DATA statement in the program; you may RESTORE a certain line number. If you use a statement such as RESTORE 380, the very next READ statement will start with the data in line 380.

Take another look at the "Cookie File" program a few pages back. Lines 2260 to the end contain DATA statements. Lines 390-420 read A$ and INV$(I,0) 20 times and use the data in lines 2260-2300. If you want to see a certain cookie recipe, you make a choice from a menu screen; in lines 630-920 the program RESTOREs the appropriate DATA statement for the particular recipe you chose. At the next READ statement, in line 930, the computer will READ whatever DATA statement RESTORE specified.

In the ingredient inventory section, line 1550 is RESTORE 2310, so the next READ statement will start at the data in line 2310 and read through all the cookie recipes.

The following program illustrates the use of DATA statements and READ statements in a high-resolution graphics display. Lines 170-340 are DATA statements that contain character definitions. Lines 130-160 READ in the information. C holds the character number, which is used as a counter in the FOR-NEXT loop.

The first iteration of the loop reads C$ as FFFFFFFFFFFFFFFF and defines character number 33 as a filled-in square. The second iteration defines character 34 to be a null character. The third iteration defines character 35 to be 0001070F1F3F7F, and so forth to character 140.

These 22 lines replace 107 CALL CHAR statements. This method uses less memory, but it makes it harder to debug and keep track of which string goes with which character number.

Since many of the defined graphics characters are actually redefined printable characters, PRINT statements can be used to draw the graphics (lines 350-460). Since these symbols and letters have been redefined, you will see, not symbols and letters, but the graphics characters which form a bull's head.

Lines 470-500 draw graphics on the screen in the non-PRINTing method. The DATA in lines 510-530 are sets of row, column, and character numbers for use in the CALL HCHAR statement.

Program 6-2. Angry Bull

```
120 CALL CLEAR
130 FOR C=33 TO 140
140 READ C$
150 CALL CHAR(C,C$)
160 NEXT C
170 DATA FFFFFFFFFFFFFFFF,,0001070F1F3F7F7F
    ,40C080000000808,0000000003C4582,000
    00304081020E,7FC
180 DATA C0303F080402,00008768101008,000080
    6C12473804,0004060703030307,000000008
    0C0E0F,E0FFFFFFFFFFFFFF
190 DATA 0102FFFEFFFFFAFC,05489020C08,00000
    30301110E,008080000CF3,070F3F2F271D06
    02,F0FCFFFFFFFF1F0D
200 DATA 0000FFFFFFFFFFFF,0F1FFFFFFFFFFFFF
    ,FCFCFCFCFCFCFCFC,7F7F7F3F1F1F2F2,FFF
    FFFFFFCF0C,FCF9FA0D
210 DATA 7080384488102021,0300010204040402,
    438C304040818282,0E166EBF7E,FFFFFFFF0
    F0301,F8F0F0E0C08
220 DATA 0000000106040E0F,20204183071F7FFF,
    008000808CFFFFFF,0000806A7FFFFFFF,222
    4455EFFFEFFFE
230 DATA 01FD0379850301 0D,84B42424241C0101,
    0C0830408090 20C,0080707C3E3E1F1F,0000
    101C3E3FFFFF
240 DATA 38300E81406,00000080C020100C,1F1F3
    F3F7F7F797,FFFFFFFFFFFCFAFD,FEFFFCFCF
    858810B,749C2008A8F8FCFC
```

```
250 DATA 8040404020204010 08,1F0F0F0F070707E
    7,070301,FFFFFF7F,F4E9CB830F070707,17
    FFFFF9FDFCFEFE
260 DATA 0F000818FCFCFCFC,F8C8070060906,38D
    890187C94E407,FFFFFFFEF090909,FFFFFF7
    F3F1F272,FFFFFFFFFFFCF8
270 DATA 03030303010101 01,7F7F7DF8E0FFFFFF,
    0707030101010303,FFFFFFCFCFCEFCF1,909
    0A0A06040C09,202020202020202
280 DATA FEFCF8F0E0C0C081,101020204043941 8,
    204040808,7F7F3F3F3F1F1F0F,FCF8F0E0E0
    E6FFFF,00001F205F84C7E
290 DATA 040484848 0C0F3FF,000000000C020101,1
    20202020404 0808,0F0F0707070737C7,FFFF
    FFB38080F0FF,E0E0C08000003FFF
300 DATA 7F7F7F3E1C0080F,808000181C1E3F7E,1
    010202040808038,FFFF3F3F3F3F1F1F,FEFE
    FEFCF0F2F1F,0F0F,FF7F,FFF8
310 DATA F0E,0780402018050381,00E040808,422
    2120A06020101,86463A01,800000000030505
    05,000003FC,408
320 DATA 0808081010 10202,0808040404040404,0
    0000080402010 1,000007080A0A04,0333428
    00810204,0E70800 1020C106
330 DATA 8000402010101 02,800102008080 80C,A0
    100F,00010638C,8080407807,40408080808
    0808,808080808E513E2
340 DATA 000007182020404,00C0201000000001
350 PRINT TAB(6);"#$ %&'()*  +,"
360 PRINT TAB(6);"!-./  0123456"
370 PRINT TAB(6);"789:  ;< =>!?"
380 PRINT TAB(5);"@ABCDE  FGHIJK"
390 PRINT TAB(5);"L!!MNOP{3 SPACES}Q!!5,"
400 PRINT TAB(6);"RSTU!VWX#YZ[S"
410 PRINT TAB(8);"\!]6 ^_`a"
420 PRINT TAB(9);"!!6 \bcd"
430 PRINT TAB(9);"e!fghij"
440 PRINT TAB(9);"k!lmnop"
450 PRINT TAB(10);"q!!!r"
460 PRINT TAB(10);"s tuv"::::
470 FOR I=1 TO 25
480 READ X,Y,C
```

```
490 CALL HCHAR(X,Y,C)
500 NEXT I
510 DATA 18,17,119,18,18,120,19,17,121,20,1
    8,122,19,18,123,20,19,124,20,20,125,1
    9,20,126
520 DATA 18,20,127,17,20,128,17,19,129,18,1
    1,130,18,10,131,19,11,132,20,11,125,2
    0,10,134
530 DATA 19,10,133,20,9,135,20,8,136,19,8,1
    37,18,8,138,17,8,139,17,9,39,17,10,14
    0,1,1,32
540 GOTO 540
550 END
```

Western States

This drill to review the 11 western states and their capital cities also shows the use of DATA and RESTORE. A map of the United States is drawn. One of the western states is outlined, and you must type in the name of the state. If you type the state correctly, you are then asked to type in the capital. Names must be spelled correctly to be accepted. If you get a state and the capital correct, it will not appear again; but if you miss either the state or the capital, the state will appear again later in the drill. The states appear in a random order.

Lines 240-320 define graphics characters using DATA. A RESTORE statement is not necessary because I want to begin with the first DATA statements in the program.

Lines 340-410 READ the 11 states and their capitals. As each state is identified, the S$(R) variable is set to "" so it won't be chosen again. If the user wants to try the quiz again, the DATA must be RESTOREd and read in again to fill up the S$(R) array.

Lines 560-590 randomly choose one of the states. If the state has already been identified, S$(R) will be "" and another state will be chosen. Line 590 branches according to which state is chosen.

Lines 1520-2090 RESTORE the proper data for the state which was chosen randomly. The program then branches back to the main procedure at line 600.

Line 610 READs *N*, how many characters must be defined; lines 620-650 READ the strings to define the graphics characters. Line 660 READs *N* for the number of characters to be drawn, and lines 670-700 outline the state on the map. Line 1270 READs *N* for the number of lines; then lines 1280-1310 READ the data to erase the state.

Each state's DATA statements contain the data separated by commas in the following order: *N*, strings for defining graphics characters, *N*, row, column, and graphics character number to outline state, *N*, row, column, graphics character number, and repetitions to erase the state.

You'll notice that I don't have a DIM statement for S$(R). That's because TI BASIC automatically DIMensions arrays up to subscript 10. Since that includes subscript 0, that gives me enough for the eleven states.

Program 6-3. Western States

```
100 REM  WESTERN STATES
110 CALL CLEAR
120 FOR G=9 TO 12
130 CALL COLOR(G,12,1)
140 NEXT G
150 CALL COLOR(13,1,12)
160 CALL COLOR(14,1,12)
170 CALL COLOR(15,2,11)
180 CALL CHAR(64,"3C4299A1A199423C")
190 PRINT "  ***********************":"  *"
    ;TAB(25);"*"
200 PRINT "  * IDENTIFY THE STATES *":"  *"
    ;TAB(25);"*"
210 PRINT "  ***********************"
220 PRINT :::TAB(7);"WESTERN STATES"
230 PRINT :::::
240 FOR G=96 TO 123
250 READ G$
260 CALL CHAR(G,G$)
270 NEXT G
280 DATA FFFFFFFFFFFFFFFF,3F1F0F0707030301,
    7F3F1F0F,FFFF7F7F3F3F3F3F,FFFFF3C,F0F
    0F0E0E0C0C08,0F0F0F0F0F0F0F0F
290 DATA 0F0F070703030101,010103030707 0F0F,
    0F0F0F0FFFFFFFFF,FFFFFFFF7F1F0701,FF3
    F0F03,FFFFFFFFFFFF0F0F
```

```
300 DATA F8FCFEFE7F3F1F0E,FFFFFFFFFEFCF8F,F
    0F8F8FCFCFEFEFF,008080C0C0E0E0F,F0E0C
    08,FCFCF8F8F0F0F0F
310 DATA 8080C0C0E0E0F0F,0F1F3F7FFFFFFFFF,0
    000000000030F3FFF,0000000000010307,E0E0
    E0F0F8FCFEFF,0000000000080C0E
320 DATA 00E0F0FEFFFFFFFF,0000000000E0F8FE,
    E0E0E1E3FFFFFEFC
330 L$="``````````````````"
340 RESTORE 380
350 FOR G=0 TO 10
360 READ S$(G),C$(G)
370 NEXT G
380 DATA WASHINGTON,OLYMPIA,OREGON,SALEM,CA
    LIFORNIA,SACRAMENTO
390 DATA NEVADA,CARSON CITY,IDAHO,BOISE,MON
    TANA,HELENA
400 DATA WYOMING,CHEYENNE,UTAH,SALT LAKE CI
    TY,ARIZONA,PHOENIX
410 DATA NEW MEXICO,SANTA FE,COLORADO,DENVE
    R
420 CALL CLEAR
430 PRINT "ONE OF THE UNITED STATES"::"WILL
    BE OUTLINED.":::"TYPE THE NAME OF TH
    E STATE"
440 PRINT :"THEN PRESS <ENTER>.":::"IF THE
    STATE IS CORRECT,"
450 PRINT :"TYPE THE CAPITAL CITY"::"THEN P
    RESS <ENTER>."
460 PRINT :::"NAMES MUST BE SPELLED"::"CORR
    ECTLY TO BE ACCEPTED.":::TAB(15);"PRE
    SS <ENTER>";
470 CALL KEY(0,K,S)
480 IF K<>13 THEN 470
490 CALL CLEAR
500 PRINT TAB(27);"ts":" i``````````````yz
    {7 SPACES}u`e":" ";L$;"yx{3 SPACES}t`r"
    :"h";L$;"`w vt``{"
510 PRINT "f";L$;"`` t``nq":"f";L$;"``t```"
    :"f";L$;"``````x":"f";L$;"``````e":"g
    ";L$;"```````"
```

```
52Ø PRINT " c";L$;"````n":" g";L$;"```nq":"
    j";L$;"``e":"{4 SPACES}kj``l`````````
    `````":TAB(1Ø);"a````````ndj``p"
53Ø PRINT TAB(11);"bdc`ndddm{3 SPACES}co":T
    AB(13);"a`{8 SPACES}a`":TAB(14);"b";TAB
    (24);"b"::::::
54Ø FOR C=Ø TO 1Ø
55Ø T=Ø
56Ø RANDOMIZE
57Ø R=INT(11*RND)
58Ø IF S$(R)="" THEN 57Ø
59Ø ON R+1 GOTO 152Ø,156Ø,161Ø,167Ø,173Ø,17
    9Ø,184Ø,189Ø,194Ø,199Ø,2Ø5Ø
6ØØ CALL HCHAR(2Ø,1,96,16Ø)
61Ø READ N
62Ø FOR I=128 TO 127+N
63Ø READ G$
64Ø CALL CHAR(I,G$)
65Ø NEXT I
66Ø READ N
67Ø FOR I=1 TO N
68Ø READ X,Y,G
69Ø CALL HCHAR(X,Y,G)
7ØØ NEXT I
71Ø FOR I=1 TO 7
72Ø CALL HCHAR(21,2+I,ASC(SEG$("STATE ?",I,
    1)))
73Ø NEXT I
74Ø CALL HCHAR(21,11,96,15)
75Ø S1$=""
76Ø CALL SOUND(15Ø,1397,2)
77Ø FOR L=1 TO 15
78Ø CALL KEY(Ø,K,S)
79Ø IF S<1 THEN 78Ø
8ØØ IF K=13 THEN 84Ø
81Ø CALL HCHAR(21,1Ø+L,K)
82Ø S1$=S1$&CHR$(K)
83Ø NEXT L
84Ø CALL SOUND(1ØØ,88Ø,2)
85Ø IF S$(R)=S1$ THEN 97Ø
86Ø CALL SOUND(1ØØ,33Ø,2)
87Ø CALL SOUND(1ØØ,262,2)
```
```

```
880 T=T+1
890 IF T<2 THEN 740
900 CALL HCHAR(21,11,96,15)
910 FOR L=1 TO LEN(S$(R))
920 CALL HCHAR(21,10+L,ASC(SEG$(S$(R),L,1))
    )
930 NEXT L
940 GOSUB 1400
950 C=C-1
960 GOTO 1270
970 GOSUB 1470
980 FOR I=1 TO 9
990 CALL HCHAR(23,2+I,ASC(SEG$("CAPITAL ?",
    I,1)))
1000 NEXT I
1010 T=0
1020 CALL HCHAR(23,13,96,15)
1030 S1$=""
1040 CALL SOUND(150,1397,2)
1050 FOR L=1 TO 15
1060 CALL KEY(0,K,S)
1070 IF S<1 THEN 1060
1080 IF K=13 THEN 1120
1090 CALL HCHAR(23,12+L,K)
1100 S1$=S1$&CHR$(K)
1110 NEXT L
1120 CALL SOUND(100,880,2)
1130 IF C$(R)=S1$ THEN 1250
1140 CALL SOUND(100,330,2)
1150 CALL SOUND(100,262,2)
1160 T=T+1
1170 IF T<2 THEN 1020
1180 CALL HCHAR(23,12,96,15)
1190 FOR L=1 TO LEN(C$(R))
1200 CALL HCHAR(23,12+L,ASC(SEG$(C$(R),L,1)
     ))
1210 NEXT L
1220 GOSUB 1400
1230 C=C-1
1240 GOTO 1270
1250 GOSUB 1470
1260 S$(R)=""
```

```
1270 READ N
1280 FOR I=1 TO N
1290 READ X,Y,G,J
1300 CALL HCHAR(X,Y,G,J)
1310 NEXT I
1320 NEXT C
1330 CALL HCHAR(21,1,96,96)
1340 PRINT "TRY AGAIN? (Y/N)";
1350 CALL KEY(0,K,S)
1360 IF K=89 THEN 340
1370 IF K<>78 THEN 1350
1380 CALL CLEAR
1390 STOP
1400 FOR I=1 TO 11
1410 CALL HCHAR(24,20+I,ASC(SEG$("PRESS ENT
     ER",I,1)))
1420 NEXT I
1430 CALL KEY(0,K,S)
1440 IF K<>13 THEN 1430
1450 CALL HCHAR(24,21,96,11)
1460 RETURN
1470 CALL SOUND(100,262,2)
1480 CALL SOUND(100,330,2)
1490 CALL SOUND(100,392,2)
1500 CALL SOUND(200,523,2)
1510 RETURN
1520 RESTORE 1530
1530 DATA 3,0101010101010101,FF,00000000E01
     00807,5,3,6,128,4,6,128,5,6,129,5,5,1
     29,4,4,130,3,4,4,96,4
1540 DATA 5,5,96,2,3,6,96,1
1550 GOTO 600
1560 RESTORE 1570
1570 DATA 6,00000000E0100807,00000000000000
     FF,0101010101010101,0101010 1FF,000000
     00FF,F0F0F0F0FFF0F0F,9
1580 DATA 4,4,128,4,5,129,4,6,129,5,6,130,6
     ,6,130,7,6,131,7,5,132,7,4,132,7,3,13
     3,5,4,4,96,3
1590 DATA 5,6,96,1,6,6,96,1,7,3,102,1,7,4,9
     6,3
1600 GOTO 600
```

```
1610 RESTORE 1620
1620 DATA 9,00000000FF,00000000F010101,1010
     101010101010l,1008040201,0000000008040
     2,1008040201010202
1630 DATA 0201010202010101,F0F0F0F0FFF0F0F,
     01C1F1FDFEFFFFFF,12,7,3,135,7,4,128,7
     ,5,129,8,5,130,9,5,130
1640 DATA 10,5,131,10,6,132,11,6,131,11,7,1
     32,12,7,133,13,7,134,14,7,136,9,7,3,1
     02,1,7,4,96,2,8,5
1650 DATA 96,1,9,5,96,1,10,5,96,2,11,6,96,2
     ,12,7,96,1,13,7,96,1,14,7,107,1
1660 GOTO 600
1670 RESTORE 1680
1680 DATA 9,000000001F10101,00000000FF,0000
     0000F010101,10101010101010l,101010101
     01010F,1109050301
1690 DATA 00000000000080402,1008040201,000000
     001F10101,15,7,5,128,7,6,129,7,7,129,
     7,8,130
1700 DATA 8,8,131,9,8,131,10,8,131,11,8,132
     ,12,7,133,11,7,134,11,6,135,10,6,134,
     10,5,135
1710 DATA 9,5,131,8,5,131,6,7,5,96,4,8,5,96
     ,4,9,5,96,4,10,5,96,4,11,6,96,3,12,7,
     96,1
1720 GOTO 600
1730 RESTORE 1740
1740 DATA 8,8181818181818181,8080404020201 0
     1,1010080602020101,8345390101010101,0
     10101010101 01FF
1750 DATA 00000000000000FF,80808080808080FF
     ,8080808080808080,10,3,7,128,4,8,129,5
     ,8,130,6,9,131
1760 DATA 7,9,132,7,8,133,7,7,134,6,7,135,5
     ,7,135,4,7,135,5,3,7,96,1,4,7,96,2,5,
     7,96,2,6,7,96,3
1770 DATA 7,7,96,3
1780 GOTO 600
1790 RESTORE 1800
1800 DATA 7,1010101010101010l,101010101010 10F
     ,00000000000000FF,134538,101008060202
     0101,8080404020201 01
```

```
181Ø DATA Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1,1Ø,3,13,128,4,13
     ,128,5,13,129,5,12,13Ø,5,11,13Ø,5,1Ø,
     13Ø,6,9,131
182Ø DATA 5,8,132,4,8,133,3,7,134,4,3,7,96,
     7,4,8,96,6,5,8,96,6,6,9,96,1
183Ø GOTO 6ØØ
184Ø RESTORE 185Ø
185Ø DATA 8,FF8Ø8Ø8Ø8Ø8Ø8Ø8,FF,FØ1Ø1Ø1Ø1Ø1Ø
     1Ø1,1Ø1Ø1Ø1Ø1Ø1Ø1Ø1,1Ø1Ø1Ø1Ø1Ø1Ø1ØF,Ø
     ØØØØØØØØØØØØØFF
186Ø DATA 8Ø8Ø8Ø8Ø8Ø8Ø8ØFF,8Ø8Ø8Ø8Ø8Ø8Ø8Ø8Ø
     ,1Ø,6,1Ø,128,6,11,129,6,12,129,6,13,1
     3Ø,7,13,131
187Ø DATA 8,13,132,8,12,133,8,11,133,8,1Ø,1
     34,7,1Ø,135,3,6,1Ø,96,4,7,1Ø,96,4,8,1Ø,
     96,4
188Ø GOTO 6ØØ
189Ø RESTORE 19ØØ
19ØØ DATA 8,3F2Ø2Ø2Ø2Ø2Ø2Ø2,FF,8Ø8Ø8Ø8Ø8Ø8Ø
     8ØFE,Ø2Ø2Ø2Ø2Ø2Ø2Ø2Ø2,Ø2Ø2Ø2FE,ØØØØ
     ØØØØFF,2Ø2Ø2Ø3F
191Ø DATA 2Ø2Ø2Ø2Ø2Ø2Ø2Ø2,1Ø,8,8,128,8,9,12
     9,8,1Ø,13Ø,9,1Ø,131,1Ø,1Ø,131,11,1Ø,1
     32,11,9,133
192Ø DATA 11,8,134,1Ø,8,135,9,8,135,4,8,8,9
     6,3,9,8,96,3,1Ø,8,96,3,11,8,96,3
193Ø GOTO 6ØØ
194Ø RESTORE 195Ø
195Ø DATA 6,ØØØØØØØ1F1Ø1ØF,ØØØØØØØØFF,ØØØØ
     ØØØØFCØ4Ø4Ø4,Ø4Ø4Ø4Ø4Ø4Ø4Ø4Ø4,8Ø8Ø8Ø8
     ØCØEØF8FE
196Ø DATA Ø1Ø1Ø1Ø1Ø2Ø2Ø2Ø1,9,11,8,128,11,9,
     129,11,1Ø,13Ø,12,1Ø,131,13,1Ø,131,14,
     1Ø,131,14,8,132
197Ø DATA 13,7,133,12,7,133,5,11,8,96,3,12,
     7,96,4,13,7,96,4,14,8,1Ø6,1,14,1Ø,96,1
198Ø GOTO 6ØØ
199Ø RESTORE 2ØØØ
2ØØØ DATA 7,ØØØØØØØØØ1Ø1Ø1Ø1,ØØØØØØØØFF,ØØØ
     ØØØØØFFØ1Ø1Ø1,Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1,Ø1Ø1Ø1
     FF,ØØØØØØ3F2ØEØEØE
```

```
2010 DATA 0000000000F0F0F,12,11,10,128,11,
     11,129,11,12,129,11,13,130,12,13,131,
     13,13,131,14,13,132
2020 DATA 14,12,133,14,11,134,14,10,131,13,
     10,131,12,10,131,5,11,10,96,4,12,10,96,
     4,13,10,96,4
2030 DATA 14,10,96,4,14,11,108,1
2040 GOTO 600
2050 RESTORE 2060
2060 DATA 8,FF,F01010101010101,101010101010
     101,1010101 0F,00000000FF,0202020203,0
2020202020202
2070 DATA 0302020202020202,12,9,11,128,9,12
     ,128,9,13,128,9,14,129,10,14,130,11,14,
     131,11,13,132
2080 DATA 11,12,132,11,11,132,11,10,133,10,
     10,134,9,10,135,3,9,10,96,5,10,10,96,
     5,11,10,96,5
2090 GOTO 600
2100 END
```

Planning Color Sets

The character numbers are divided into groups of eight characters each, and each group has a color set number. The CALL COLOR statement assigns to a certain color set, by number, its foreground and background colors. Every character in the same color set will have the same color. Here is a brief list of characters and sets. (See the Appendix for an extended list.)

Set	ASCII Codes	Set	ASCII Codes
1	32-39	9	96-103
2	40-47	10	104-111
3	48-55	11	112-119
4	56-63	12	120-127
5	64-71	13	128-135
6	72-79	14	136-143
7	80-87	15	144-151
8	88-95	16	152-159

Suppose you want to print the letter *R* in red. The ASCII code for *R* is 82. Looking at the chart above, you can see that character 82 is in set 7. Use the statement CALL COLOR(7,9,1) to assign a medium red foreground and a transparent background to set 7. Not only *R*, but also the other letters in set 7 will be red. Any character in set 7 that is currently on the screen will change to red when the CALL COLOR statement is carried out.

Color Sets in New England

In "Western States," you had to spell the states and capitals correctly. In this easier program, you are shown a map and a menu, and only have to recognize the state's name.

The New England states are drawn on the screen, each in a different color, and labeled. When you know the state names, press ENTER and the labels will be cleared. In a random order the states will be numbered and listed in a menu. Also in a random order, one state at a time will flash. Press the number of the correct state name.

After all six states have been named correctly, a multiple-choice quiz of the capital cities is presented (lines 1360-1790). This program logic could be adapted for other multiple-choice or matching drills. The six states and six capitals are each in arrays. In a random order the states are numbered and listed, and the capitals are listed with the letters *a* through *f*. For the quiz, the student must press the correct letter for each numbered state.

The map of the New England states was first drawn on 24-by-32 graph paper with each state in a different color. (See Figure 6-1.) Notice that I adapted all the more-or-less straight boundaries so they could be drawn with solid squares.

Next, the states were drawn in more detail on paper designed for character definitions. Ideally, each state could be defined with the characters in a single color set. However, Maine had so many characters that needed to be defined that two color sets were necessary. Maine uses characters 97 through 111 and color sets 9 and 10 were assigned a yellow foreground and transparent background.

Connecticut uses characters 144-148, and the color set is light red on transparent.

Vermont and New Hampshire have a common border with defined characters, so they share a color set, set 11, with a

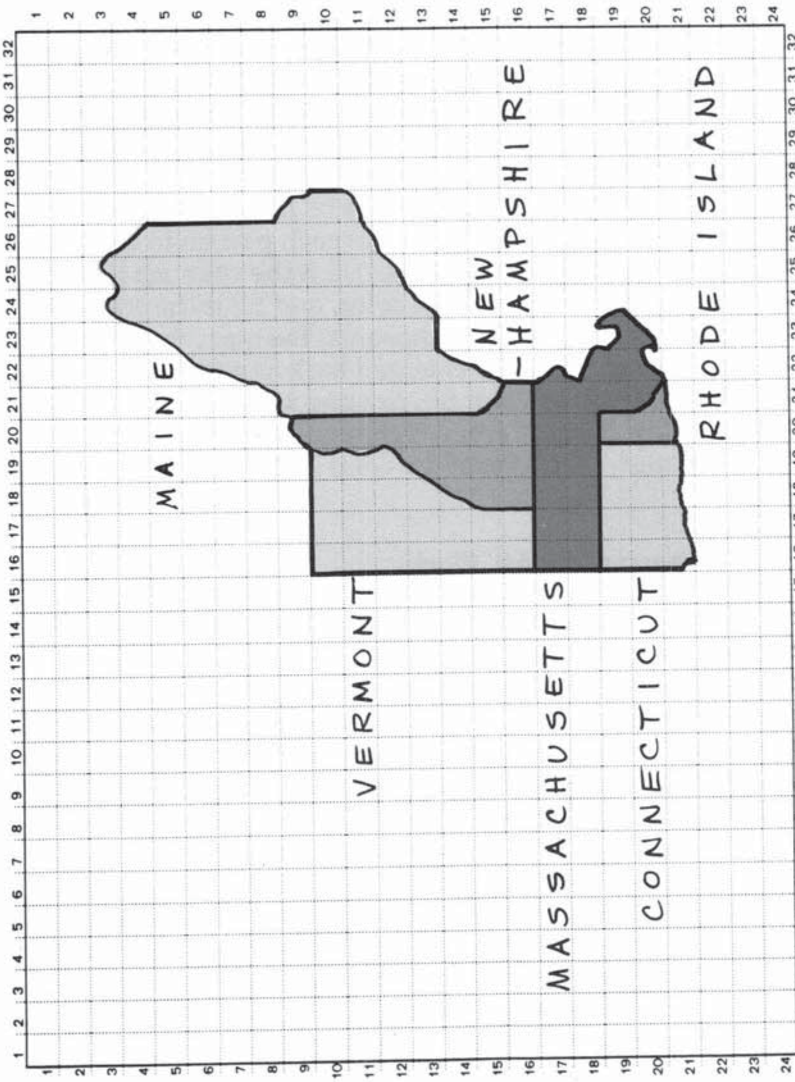


Figure 6-1. Planning the Screen for "New England States"

green foreground and red background. New Hampshire has two more characters that require different colors, so character 152 was defined in set 16 with red on transparent; and character 40 was defined in set 2 with red on yellow.

Massachusetts has special characters 120-125 defined in set 12 with a magenta foreground and transparent background. Rhode Island uses two characters in set 14 defined as blue on transparent. Rhode Island and Massachusetts share one graphics character, 128, that needed to be magenta and blue.

Arrays are set up so the subscripts each pertain to a particular state. S$ is the state name, SS is the color set for the state, SF is the state's foreground color, and SB is the state's background color. As a state is chosen in the quiz, the state's particular color set SS can be changed back and forth from white to the foreground color SF, causing it to blink. The exception is New Hampshire, which requires blinking the background color. Only one of the color sets is blinked for Maine.

How "New England States" Works

Lines

110-170	Print the title screen.
180-380	Define graphics characters and colors.
390-450	Clear the screen; print instructions.
460-700	Clear the screen; draw New England states with labels.
710-800	After the student presses ENTER, clear labels.
810-860	READ arrays for state names, color set numbers, foreground colors, and background colors.
870-1000	Print a list of the states in random order.
1010-1060	Randomly choose a state that has not been previously chosen; if New Hampshire, branch.
1070-1100	Blink the colors while waiting for the student's answer.
1110-1140	If the answer is incorrect, play "uh-oh" and wait for another answer.
1150-1230	If answer is correct, return the state to its original color; play an arpeggio. Set SS$ element to null so the state will not be chosen again, then go to another state.

1240-1270	Print the option to try again and branch appropriately.
1280-1350	Procedure for New Hampshire.
1360-1380	Begin the drill for capitals.
1390-1440	READ the array of states with capitals.
1450-1530	Print a list of the states in random order.
1540-1620	Print a list of the capitals in random order.
1630-1790	For each state, receive the student's choice of capital city.
1800-1880	Present the options for the states quiz, the capitals quiz, or the end of the program; branch appropriately.

Program 6-4. New England States

```
100 REM  NEW ENGLAND STATES
110 CALL CHAR(64,"3C4299A1A1994237")
120 CALL CLEAR
130 PRINT "  ***********************":"  *"
    ;TAB(25);"*"
140 PRINT "  * IDENTIFY THE STATES *":"  *"
    ;TAB(25);"*"
150 PRINT "  ***********************"
160 PRINT :::TAB(5);"NEW ENGLAND STATES"
170 PRINT :::::
180 FOR G=97 TO 125
190 READ G$
200 CALL CHAR(G,G$)
210 NEXT G
220 DATA ØØØØØØØØØØ307878,ØØØØØØØØØE1E3FFF,
    ØØØ1Ø3Ø7Ø7ØFØF,8ØCØEØFØF8FCFEFF,ØF1
    F1F3F3F7F7FFF
230 DATA ØØØ1Ø3Ø7ØFØF1F3F,3F3F3F7F7F7FFFFF,
    Ø1Ø1Ø3Ø3Ø3Ø3Ø3Ø3,CØFØF8F8F8F8F8FC,FFF
    CFØ8,FFFFFFFFFFCF88
240 DATA FEFCF8FØEØCØ8,FFFFFEFCFCFCF8F8,F8F
    8FØEØEØCØCØ8,FFFFFFFFFFFFFFFF,FFFFFF
    FFFFFFFFFF
250 DATA FFFEFCFCFCFEFEFE,FCF8F8FØEØC,FEF8F
    8FØFØEØEØC,CØCØCØ8Ø8Ø8Ø8,Ø,,,FFFFFF
    FFFFFFFFFF,ØØØØØØCØCØCØ8Ø8
```

```
260 DATA 0000000000F8FEFE,0603030307 07FFFE,
    FEFCF8F0E0C08,FFFFFFFFF9F0C
270 DATA 128,80C0E0F0F0F8F8FC,136,FFFFFFFF
    FFFFFFF,137,FFF8,144,FFFFFFFFFFFFFFFF
280 DATA 145,FFFF7F3F3F7FF08,146,FFFFFFFFFF
    C,147,FFFFFFFF,148,FFFFFE,152,0000000
    7DFDFFFFF,40,00000000C0F0FCFE
290 DATA 153,000000FF,11,1,11,1,4,7,14,1,5,
    14,5,1,10,1,7,1
300 FOR I=1 TO 11
310 READ G,G$
320 CALL CHAR(G,G$)
330 NEXT I
340 FOR I=9 TO 16
350 READ F,B
360 CALL COLOR(I,F,B)
370 NEXT I
380 A$=CHR$(144)&CHR$(144)&CHR$(144)&CHR$(1
    44)&CHR$(136)
390 CALL CLEAR
400 PRINT "LEARN THE NAMES OF THE"::"NEW EN
    GLAND STATES."
410 PRINT ::"THE STATES WILL BE SHOWN."::"W
    HEN YOU KNOW THE NAMES,"::"PRESS <ENT
    ER>."
420 PRINT ::"THE NAMES WILL CLEAR."::"AS TH
    E STATE BLINKS, PRESS"
430 PRINT :"THE NUMBER OF THE CORRECT"::"NA
    ME."::::TAB(15);"PRESS <ENTER>.";
440 CALL KEY(0,K,S)
450 IF K<>13 THEN 440
460 CALL CLEAR
470 CALL SCREEN(8)
480 CALL COLOR(2,7,11)
490 PRINT TAB(22);"ab":TAB(21);"cood"
500 PRINT TAB(16);"MAINEeooo"
510 PRINT TAB(20);"foooo":TAB(20);"goooo":T
    AB(19);"hooooo"
520 PRINT TAB(18);CHR$(152)&"ooooooi"
530 PRINT TAB(14);"pppquoooooo"
540 PRINT TAB(7);"VERMONTpppquoooookj"
550 PRINT TAB(14);"pppruooool"
```

```
560 PRINT TAB(14);"ppsuuoool"
570 PRINT TAB(14);"pptuuom"
580 PRINT TAB(14);"ppuuu(nNEW"
590 PRINT TAB(14);"ppuuuu"&CHR$(153)&"HAMPS
    HI"
600 CALL HCHAR(23,30,82)
610 CALL HCHAR(23,31,69)
620 PRINT "MASSACHUSETTSxxxxxxy"
630 PRINT TAB(14);"xxxxxxz"
640 PRINT TAB(14);A$&"xx{"
650 PRINT "  CONNECTICUT";A$&CHR$(128)&"
    }¦"
660 PRINT TAB(14);CHR$(145)&CHR$(146)&CHR$(
    147)&CHR$(148)&CHR$(137)
670 PRINT TAB(18);"RHODE ISLA"::
680 CALL HCHAR(22,30,78)
690 CALL HCHAR(22,31,68)
700 PRINT TAB(15);"PRESS <ENTER>";
710 CALL KEY(Ø,K,S)
720 IF K<>13 THEN 710
730 CALL HCHAR(5,18,32,5)
740 CALL HCHAR(11,9,32,7)
750 CALL HCHAR(15,23,32,3)
760 CALL HCHAR(16,22,32,10)
770 CALL HCHAR(17,3,32,13)
780 CALL HCHAR(20,5,32,11)
790 CALL HCHAR(22,20,32,12)
800 CALL HCHAR(24,17,32,13)
810 RESTORE 820
820 DATA MAINE,10,11,1,VERMONT,11,4,7,NEW H
    AMPSHIRE,11,4,7,MASSACHUSETTS,12,14,1
830 DATA CONNECTICUT,15,10,1,RHODE ISLAND,1
    4,5,1
840 FOR I=1 TO 6
850 READ S$(I),SS(I),SF(I),SB(I)
860 NEXT I
870 FOR C=1 TO 6
880 RANDOMIZE
890 X=INT(RND*6)+1
900 IF S$(X)="" THEN 890
910 SS$(X)=S$(X)
920 FF(X)=SF(X)
```

```
930 BB(X)=SB(X)
940 ANS(X)=C
950 CALL HCHAR(2+C,2,48+C)
960 FOR J=1 TO LEN(S$(X))
970 CALL HCHAR(2+C,J+3,ASC(SEG$(S$(X),J,1))
    )
980 NEXT J
990 S$(X)=""
1000 NEXT C
1010 FOR C=1 TO 6
1020 RANDOMIZE
1030 X=INT(RND*6)+1
1040 IF SS$(X)="" THEN 1030
1050 CALL SOUND(150,1397,4)
1060 IF SS$(X)="NEW HAMPSHIRE" THEN 1280
1070 CALL KEY(0,K,S)
1080 CALL COLOR(SS(X),16,SB(X))
1090 CALL COLOR(SS(X),SF(X),SB(X))
1100 IF S<1 THEN 1070
1110 IF K-48=ANS(X)THEN 1150
1120 CALL SOUND(100,330,2)
1130 CALL SOUND(100,262,2)
1140 GOTO 1070
1150 CALL COLOR(SS(X),SF(X),SB(X))
1160 CALL HCHAR(2+ANS(X),1,62)
1170 CALL SOUND(150,262,1)
1180 CALL SOUND(150,330,1)
1190 CALL SOUND(150,392,1)
1200 CALL SOUND(300,523,1)
1210 SS$(X)=""
1220 CALL HCHAR(2+ANS(X),1,32)
1230 NEXT C
1240 PRINT "TRY AGAIN? Y OR N";
1250 CALL KEY(0,K,S)
1260 IF K=89 THEN 460
1270 IF K=78 THEN 1360 ELSE 1250
1280 CALL KEY(0,K,S)
1290 CALL COLOR(11,4,16)
1300 CALL COLOR(11,4,7)
1310 IF S<1 THEN 1280
1320 IF K-48=ANS(X)THEN 1160
1330 CALL SOUND(100,330,2)
```

```
1340 CALL SOUND(100,262,2)
1350 GOTO 1280
1360 CALL CLEAR
1370 CALL COLOR(2,2,1)
1380 PRINT "NOW MATCH THE CAPITALS.":::
1390 RESTORE 1400
1400 DATA MAINE,AUGUSTA,NEW HAMPSHIRE,CONCO
     RD,VERMONT,MONTPELIER
1410 DATA MASSACHUSETTS,BOSTON,CONNECTICUT,
     HARTFORD,RHODE ISLAND,PROVIDENCE
1420 FOR I=1 TO 6
1430 READ S$(I),C$(I)
1440 NEXT I
1450 FOR I=1 TO 6
1460 RANDOMIZE
1470 X=INT(6*RND)+1
1480 IF S$(X)="" THEN 1470
1490 ANS(I)=X
1500 PRINT I;S$(X)
1510 S$(X)=""
1520 NEXT I
1530 PRINT
1540 FOR I=1 TO 6
1550 RANDOMIZE
1560 X=INT(6*RND)+1
1570 IF ANS(X)=0 THEN 1560
1580 CC(I)=X
1590 PRINT TAB(15);CHR$(64+I);" ";C$(ANS(X)
     )
1600 ANS(X)=0
1610 NEXT I
1620 PRINT
1630 FOR I=1 TO 6
1640 PRINT TAB(6);I
1650 CALL KEY(0,K,S)
1660 CALL HCHAR(23,11,63)
1670 CALL HCHAR(23,11,32)
1680 IF S<1 THEN 1650
1690 IF (K<65)+(K>70)THEN 1650
1700 CALL HCHAR(23,11,K)
1710 IF CC(K-64)=I THEN 1750
1720 CALL SOUND(100,330,2)
```

```
1730 CALL SOUND(100,262,2)
1740 GOTO 1650
1750 CALL SOUND(150,262,2)
1760 CALL SOUND(150,330,2)
1770 CALL SOUND(150,392,2)
1780 CALL SOUND(150,523,2)
1790 NEXT I
1800 PRINT :"PRESS 1 FOR STATES QUIZ"
1810 PRINT "{6 SPACES}2 FOR CAPITALS QUIZ"
1820 PRINT "{6 SPACES}3 TO END PROGRAM";
1830 CALL KEY(0,K,S)
1840 IF K=49 THEN 460
1850 IF K=50 THEN 1360
1860 IF K<>51 THEN 1830
1870 CALL CLEAR
1880 END
```

Touch-typing with Color Sets

"Type-ette" is a series of programs to learn touch-typing using the computer. Unit 2, presented here, shows how color sets can be used to make a certain finger appear in red. The whole hand is drawn in yellow; then, when a particular letter is being taught, the finger that should be used to type the letter is blinked and then shown in red.

The characters used in the little fingers, the thumbs, and the backs of the hands are contained in two color sets. The little fingers are not used in this program, so they do not need to be in a separate color set. The forefingers are defined with characters 104 and 105 in set 10; the middle fingers are defined with characters 96 and 97 in set 9; and the ring fingers are defined with characters 120 and 121 in set 12 (lines 990-1520).

At first, all color sets involving the hands are defined as yellow on transparent, so the hand is drawn all yellow (lines 1810-1850). When a new letter is introduced, only the color set of the particular finger involved is changed, so one finger will blink a few times and then stay red (subroutine, lines 460-500). After the screen is completed, the color set is returned to yellow; the next time the hand appears, it will be all yellow again.

As letters are introduced, they appear as black on yellow. Since the regular black-on-transparent letters are necessary for drills and instructions, the black-on-yellow letters are defined in another color set.

The computer can be an excellent instructional aid for learning how to touch-type. Color graphics and sound help to maintain the student's interest and give immediate positive response in an individualized learning situation. A student who needs extra practice can run the program over and over.

Type-ette, Unit 1 (not in this book), draws the hands on the screen and teaches the home position. Starting with Unit 2, the letters are taught gradually and in a sequence, so you can type more words with each letter learned. *E* and *H* are the first new letters taught, then *T* and *I* so many common words may be typed. After a group of new letters is introduced, there is a drill of phrases. A phrase is chosen randomly from nine possible phrases. The student must type five phrases correctly before the program continues.

Program 6-5. Type-ette, Unit 2

```
100 REM  TYPE-ETTE UNIT 2
110 GOTO 700
120 PRINT "PRESS <ENTER> TO CONTINUE."
130 CALL KEY(0,KEY,S)
140 IF KEY<>13 THEN 130
150 RETURN
160 CALL HCHAR(X+1,3,120)
170 CALL HCHAR(X+2,3,121)
180 CALL VCHAR(X+3,3,122,2)
190 CALL HCHAR(X,4,120)
200 CALL VCHAR(X+1,4,121,2)
210 CALL VCHAR(X+3,4,152,2)
220 CALL HCHAR(X,5,96)
230 CALL VCHAR(X+1,5,97,2)
240 CALL VCHAR(X+3,5,152,2)
250 CALL HCHAR(X,6,104)
260 CALL VCHAR(X+1,6,105,2)
270 CALL HCHAR(X+3,6,123)
280 CALL HCHAR(X+4,6,124)
290 CALL HCHAR(X+4,7,125)
300 RETURN
310 CALL HCHAR(X+1,30,120)
320 CALL HCHAR(X+2,30,121)
```

```
330 CALL VCHAR(X+3,30,123,2)
340 CALL HCHAR(X,29,112)
350 CALL VCHAR(X+1,29,113,2)
360 CALL VCHAR(X+3,29,152,2)
370 CALL HCHAR(X,28,96)
380 CALL VCHAR(X+1,28,97,2)
390 CALL VCHAR(X+3,28,152,2)
400 CALL HCHAR(X,27,104)
410 CALL VCHAR(X+1,27,105,2)
420 CALL HCHAR(X+3,27,122)
430 CALL HCHAR(X+4,27,126)
440 CALL HCHAR(X+4,26,127)
450 RETURN
460 FOR I=1 TO 15
470 CALL COLOR(C,12,1)
480 CALL COLOR(C,7,1)
490 NEXT I
500 RETURN
510 PRINT "{3 SPACES}"&R1$
520 PRINT :"{4 SPACES}A S D F "&CHR$(152)&"
    "&CHR$(152)&" J K L ; "&CHR$(159):::
530 RETURN
540 B$=""
550 FOR Y=5 TO 20
560 CALL KEY(0,K,S)
570 IF S<1 THEN 560
580 CALL HCHAR(24,Y,K)
590 B$=B$&CHR$(K)
600 NEXT Y
610 IF POS(B$,B1$,1)>0 THEN 650
620 CALL HCHAR(24,5,152,21)
630 CALL SOUND(150,1397,4)
640 GOTO 550
650 RETURN
660 PRINT "  "&R1$::"{3 SPACES}"&R1$
670 PRINT :"{4 SPACES}A S D F "&CHR$(152)&"
    "&CHR$(152)&" J K L ; "&CHR$(159)
680 PRINT :"{3 SPACES}"&R$&" "&CHR$(152)&CH
    R$(152)
690 RETURN
700 CALL CLEAR
710 CALL CHAR(92,"3C4299A1A199423C")
```

```
720 CALL CHAR(152,"Ø")
730 CALL CHAR(153,"FFFFFFFFFFFFFFFF")
740 CALL CHAR(154,"ØØFFFFØØØØFFFF")
750 PRINT " ** T Y P E - E T T E **":::TAB(
    8);"T Y P I N G"::TAB(1Ø);"ON  THE"::
    TAB(1Ø);"TI 99/4A":::::::::::::
760 CALL CHAR(155,"ØØØØØØFFFF")
770 CALL CHAR(156,"FF8F8F8FFFF8F8F8")
780 CALL CHAR(157,"FFFFFFFF")
790 PRINT ;
800 CALL CHAR(158,"FØFØFØFØFØFØFØF")
810 CALL CHAR(159,"FFE7C38181C3E7FF")
820 CALL COLOR(16,2,12)
830 FOR X=13 TO 23
840 CALL HCHAR(X,7,152,18)
850 NEXT X
860 CALL HCHAR(15,7,155,18)
870 CALL HCHAR(13,20,154,4)
880 CALL HCHAR(14,20,154,4)
890 FOR Y=2Ø TO 23
900 CALL VCHAR(16,Y,153,7)
910 CALL HCHAR(Y-2,8,153,11)
920 NEXT Y
930 CALL HCHAR(22,8,153,11)
940 CALL HCHAR(19,9,156,9)
950 CALL HCHAR(2Ø,9,156,9)
960 CALL HCHAR(21,1Ø,157,7)
970 T=25Ø
980 CALL SOUND(T,392,1,33Ø,6,131,9)
990 CALL CHAR(96,"3C7E7E7E7E7E7E7E")
1000 CALL SOUND(T,33Ø,1,262,6,131,9)
1010 CALL CHAR(97,"7E7E7E7E7E7E7E7E")
1020 CALL SOUND(T/2,33Ø,Ø,262,6,131,9)
1030 CALL SOUND(T/2,349,1,294,6,131,9)
1040 CALL SOUND(T,392,Ø,33Ø,6,131,9)
1050 CALL CHAR(1Ø4,"ØØØØØØØØ3C7E7E7E")
1060 CALL SOUND(T,349,2,294,7,123,1Ø)
1070 CALL CHAR(1Ø5,"7E7E7E7E7E7E7E7E")
1080 CALL SOUND(T,294,2,247,7,123,1Ø)
1090 CALL CHAR(112,"ØØØØØØØØ3C7E7E7E")
1100 CALL SOUND(T/2,294,2,247,7,123,1Ø)
1110 CALL SOUND(T/2,33Ø,2,262,7,123,1Ø)
```

```
1120 CALL SOUND(T,349,2,294,7,123,10)
1130 CALL CHAR(113,"7E7E7E7E7E7E7E7E")
1140 CALL SOUND(T,330,3,196,8,131,11)
1150 CALL CHAR(120,"000000003C7E7E7E")
1160 CALL SOUND(T,262,3,196,8,131,11)
1170 CALL CHAR(121,"7E7E7E7E7E7E7E7E")
1180 CALL SOUND(T/2,262,3,196,8,131,11)
1190 CALL SOUND(T/2,294,3,196,8,131,11)
1200 CALL SOUND(T/2,330,3,196,8,131,11)
1210 CALL SOUND(T/2,262,3,196,8,131,11)
1220 CALL SOUND(2*T,294,3,247,8,196,10)
1230 CALL CHAR(122,"7F7F7F7F7F7F7F7F")
1240 CALL CHAR(123,"FEFEFEFEFEFEFEFE")
1250 CALL CHAR(124,"FEFEFEFEFFFFFFFF")
1260 CALL SOUND(T,392,5,330,10,131,15)
1270 CALL CHAR(125,"1F1F3F3F7EFEFCFC")
1280 CALL SOUND(T,330,5,262,10,131,15)
1290 CALL CHAR(126,"7F7F7F7FFFFFFFFF")
1300 CALL SOUND(T/2,330,5,262,10,131,15)
1310 CALL SOUND(T/2,349,5,294,10,131,15)
1320 CALL SOUND(T,392,5,330,10,13131,15)
1330 CALL CHAR(127,"F8F8FCFC7E7F3F3F")
1340 CALL SOUND(T,349,3,294,8,123,12)
1350 CALL CHAR(144,"0")
1360 CALL SOUND(T,294,3,247,8,123,12)
1370 CALL CHAR(128,"0078444444447800")
1380 CALL SOUND(T/2,294,3,247,8,123,12)
1390 CALL SOUND(T/2,330,3,262,8,123,12)
1400 CALL SOUND(T,349,3,294,8,123,12)
1410 CALL CHAR(129,"007C407840407C")
1420 CALL SOUND(T,330,1,196,7,131,9)
1430 CALL CHAR(130,"0004040404444438")
1440 CALL SOUND(T,262,1,196,7,131,9)
1450 CALL CHAR(131,"0044447C444444")
1460 CALL SOUND(T/2,262,1,196,7,131,9)
1470 CALL SOUND(T/2,294,1,196,7,131,9)
1480 CALL SOUND(T/2,330,1,196,7,131,9)
1490 CALL SOUND(T/2,262,1,196,7,131,9)
1500 CALL SOUND(2*T,294,1,196,7,123,9)
1510 CALL CHAR(132,"007C40784040 4")
1520 CALL CHAR(133,"007C10101010 1")
1530 R$=CHR$(152)
```

```
1540 FOR I=1 TO 9
1550 R$=R$&" "&CHR$(152)
1560 NEXT I
1570 CALL SOUND(1.5*T,294,3)
1580 CALL CLEAR
1590 CALL SCREEN(8)
1600 CALL SOUND(T/2,330,3)
1610 CALL SOUND(1.5*T,294,2)
1620 R1$=R$&" "&CHR$(152)
1630 CALL SOUND(T/2,330,2)
1640 CALL SOUND(1.5*T,349,1,196,7,123,10)
1650 PRINT "  "&R1$
1660 CALL SOUND(T/2,392,1)
1670 CALL SOUND(1.5*T,349,0,196,7,123,9)
1680 PRINT :"{3 SPACES}"&R1$
1690 CALL SOUND(T/2,392,0)
1700 CALL SOUND(1.5*T,330,1,196,7,131,9)
1710 PRINT :"{4 SPACES}"&R$&" "&CHR$(159)
1720 CALL SOUND(T/2,349,1)
1730 CALL SOUND(1.5*T,330,2,196,7,131,10)
1740 PRINT :"{3 SPACES}"&R$&" "&CHR$(152)&C
     HR$(152)
1750 CALL SOUND(T/2,349,2)
1760 CALL SOUND(T,392,2,262,7,165,9)
1770 CALL COLOR(15,16,7)
1780 CALL SOUND(T,392,2,262,7,131,9)
1790 PRINT :::
1800 CALL SOUND(T,392,2,262,7,165,9)
1810 CALL COLOR(9,12,1)
1820 CALL COLOR(10,12,1)
1830 CALL SOUND(T,440,2,175,7)
1840 CALL COLOR(11,12,1)
1850 CALL COLOR(12,12,1)
1860 CALL SOUND(T,392,1,262,6,165,8)
1870 PRINT "{3 SPACES}LEARNING THE KEYBOARD
     "
1880 CALL SOUND(T,392,1,262,6,131,8)
1890 PRINT ::TAB(11);"UNIT  2"
1900 CALL SOUND(T,392,1,262,6,165,8)
1910 PRINT :::::
1920 CALL COLOR(13,2,12)
1930 CALL SOUND(T,330,1,196,7)
```

```
1940 CALL CHAR(134,"ØØ381Ø1Ø1Ø38ØØ")
1950 CALL SOUND(T,392,1,165,7)
1960 CALL CHAR(135,"ØØ485Ø6Ø5Ø4844")
1970 CALL SOUND(T,392,1,131,7)
1980 CALL CHAR(158,"ØØØØØØØØØ3Ø3")
1990 CALL SOUND(T,392,1,165,7)
2000 CALL SOUND(T/2,349,1,175,7)
2010 CALL SOUND(T/2,330,1)
2020 CALL SOUND(T,294,1,196,7)
2030 CALL SOUND(T/2,330,1,131,7)
2040 CALL SOUND(T/2,349,1)
2050 CALL SOUND(1.5*T,294,1,196,7)
2060 CALL SOUND(T/2,262,1)
2070 CALL SOUND(2*T,262,1,196,6,165,8)
2080 CALL CLEAR
2090 CALL SCREEN(4)
2100 PRINT "IN UNIT 1 YOU LEARNED":"THE ""H
     OME KEYS"".":::
2110 GOSUB 520
2120 PRINT ::"AFTER EACH LETTER YOU TYPE, Y
     OUR FINGERS RETURN ""HOME""."
2130 PRINT :"THIS UNIT WILL ADD MORE":"LETT
     ERS FOR YOU TO LEARN."::::
2140 GOSUB 120
2150 FOR GS=1 TO 7
2160 CALL CLEAR
2170 ON GS GOSUB 2200,2430,2640,2730,2970,3
     080,3290
2180 NEXT GS
2190 GOTO 3380
2200 PRINT "THE MOST USED LETTER OF":"THE A
     LPHABET IS ""E"".":::TAB(12);"USE YO
     UR LEFT"
2210 PRINT TAB(12);"MIDDLE FINGER."::TAB(12
     );"GO UP FROM "&CHR$(128):TAB(12);"TO
     STRIKE THE "&CHR$(129)::::::
2220 X=13
2230 GOSUB 160
2240 GOSUB 510
2250 CALL HCHAR(21,11,128)
2260 CALL HCHAR(19,10,129)
2270 C=9
```

```
2280 GOSUB 460
2290 GOSUB 120
2300 CALL HCHAR(23,1,32,32)
2310 PRINT "PRACTICE TYPING THIS LINE:":::
2320 CALL HCHAR(22,1,152,96)
2330 FOR Y=5 TO 17 STEP 4
2340 CALL HCHAR(23,Y,68)
2350 CALL HCHAR(23,Y+1,69)
2360 CALL HCHAR(23,Y+2,68)
2370 CALL HCHAR(23,Y+3,32)
2380 NEXT Y
2390 B1$="DED"
2400 GOSUB 540
2410 CALL COLOR(9,12,1)
2420 RETURN
2430 PRINT "LET'S LEARN ""H"".":::"USE YOU
     R RIGHT":"POINTER FINGER."::::::
2440 X=17
2450 GOSUB 310
2460 GOSUB 520
2470 CALL HCHAR(21,19,130)
2480 CALL HCHAR(21,17,131)
2490 C=10
2500 GOSUB 460
2510 PRINT :::"REMEMBER TO RETURN TO THE":"
     HOME POSITION AFTER":"STRIKING ANOTHE
     R KEY!"
2520 PRINT ::"TRY TYPING THIS LINE:":::
2530 CALL HCHAR(22,1,152,96)
2540 FOR Y=5 TO 17 STEP 4
2550 CALL HCHAR(23,Y,74)
2560 CALL HCHAR(23,Y+1,72)
2570 CALL HCHAR(23,Y+2,74)
2580 CALL HCHAR(23,Y+3,32)
2590 NEXT Y
2600 B1$="JHJ"
2610 GOSUB 540
2620 CALL COLOR(C,12,1)
2630 RETURN
2640 GOSUB 660
2650 GOSUB 4070
2660 PRINT ::::"NOW YOU CAN TRY PHRASES."
```

```
2670 PRINT :"TYPE THE GIVEN PHRASE":"THEN P
     RESS ENTER.":::::::
2680 DATA "A LAD ASKED DAD;","HE HAD A FALS
     E LEAD","SHE HAS A LEAD","HE HAD ALFA
     LFA;","HAL HAD A SALE;"
2690 DATA "SHE HAS LED SALES;","HE HAS A DE
     SK;","SHE HAS A LEASE","ED HAS ADS;",
     "SEAL A DEAL"
2700 RESTORE 2680
2710 GOSUB 3540
2720 RETURN
2730 PRINT "TIME TO LEARN MORE!":::
2740 PRINT CHR$(133);" IS THE SECOND MOST U
     SED":"  LETTER IN TYPING."
2750 PRINT :TAB(14);"USE YOUR LEFT":TAB(14)
     ;"POINTER FINGER."
2760 PRINT :TAB(14);"REMEMBER TO":TAB(14);"
     REACH UP THEN":TAB(14);"RETURN TO "&C
     HR$(132)&".":::
2770 X=17
2780 GOSUB 160
2790 GOSUB 510
2800 C=10
2810 GOSUB 4070
2820 CALL HCHAR(21,13,132)
2830 CALL HCHAR(19,14,133)
2840 GOSUB 460
2850 PRINT "TYPE THIS EXERCISE:":::
2860 CALL HCHAR(22,1,152,96)
2870 FOR Y=5 TO 17 STEP 4
2880 CALL HCHAR(23,Y,70)
2890 CALL HCHAR(23,Y+1,84)
2900 CALL HCHAR(23,Y+2,70)
2910 CALL HCHAR(23,Y+3,32)
2920 NEXT Y
2930 B1$="FTF"
2940 GOSUB 540
2950 CALL COLOR(C,12,1)
2960 RETURN
2970 PRINT "THE PERIOD USED AT THE"::"END O
     F A SENTENCE IS"
```

```
2980 PRINT :"PRESSED WITH THE"::"RIGHT RING
     FINGER."
2990 PRINT ::::"PRACTICE THIS LINE:"::::::
3000 CALL HCHAR(22,1,152,96)
3010 FOR Y=5 TO 19 STEP 2
3020 CALL HCHAR(23,Y,76)
3030 CALL HCHAR(23,Y+1,46)
3040 NEXT Y
3050 B1$="L.L."
3060 GOSUB 540
3070 RETURN
3080 PRINT CHR$(134);" IS ANOTHER VOWEL TO
     LEARN."::"USE YOUR RIGHT":"MIDDLE FIN
     GER."
3090 PRINT :"STRIKE "&CHR$(134)&" THEN":"RE
     TURN TO "&CHR$(135)::::::
3100 X=13
3110 GOSUB 310
3120 C=9
3130 GOSUB 460
3140 GOSUB 510
3150 GOSUB 4060
3160 CALL HCHAR(19,20,134)
3170 PRINT ::"TYPE THIS EXERCISE:":::
3180 CALL HCHAR(22,1,152,96)
3190 FOR Y=5 TO 17 STEP 4
3200 CALL HCHAR(23,Y,75)
3210 CALL HCHAR(23,Y+1,73)
3220 CALL HCHAR(23,Y+2,75)
3230 CALL HCHAR(23,Y+3,32)
3240 NEXT Y
3250 B1$="KIK"
3260 GOSUB 540
3270 CALL COLOR(C,12,1)
3280 RETURN
3290 GOSUB 660
3300 GOSUB 4040
3310 PRINT ::::"PRACTICE THESE NEW LETTERS"
     :"BY TYPING THESE SENTENCES."::::::::
3320 DATA "JED IS AT THE FIELD.","HE IS AT
     THE LAKE.","SAL DID TAKE THE TEST."
```

```
3330 DATA "HIS AIDES HAD THE LIST.","SHE FI
     LED THE LIST.","HE FLIES A JET."
3340 DATA "IT IS THE LAST TEST.","I HIT IT
     FAST.","HIS IS THE LAST SET."
3350 RESTORE 3320
3360 GOSUB 3540
3370 RETURN
3380 CALL CLEAR
3390 PRINT "REMEMBER,"::"TO LEARN TOUCH TYP
     ING -":::"DO NOT LOOK AT YOUR FINGERS
     ."
3400 PRINT :"MEMORIZE THE KEY POSITIONS.":: 
     "RETURN TO THE HOME POSITION AFTER ST
     RIKING EACH KEY."
3410 PRINT :"PRACTICE!!"::::
3420 GOSUB 120
3430 CALL CLEAR
3440 PRINT "THIS COMPLETES UNIT 2."::::"CHO
     OSE:"::"1  REVIEW ""E""":"2  REVIEW
     ""H"""
3450 PRINT :"3  REVIEW PHRASES FOR E,H"::"4
       REVIEW ""T"""::"5  REVIEW "".""""
3460 PRINT :"6  REVIEW ""I"""::"7  REVIEW S
     ENTENCES"::"8  END PROGRAM"
3470 CALL KEY(Ø,KEY,S)
3480 IF (KEY<49)+(KEY>56)=-1 THEN 3470
3490 CALL CLEAR
3500 ON KEY-48 GOSUB 2200,2430,2640,2740,29
     70,3080,3290,3520
3510 GOTO 3430
3520 GOSUB 3890
3530 STOP
3540 FOR I=1 TO 9
3550 READ A$(I)
3560 NEXT I
3570 RANDOMIZE
3580 FOR I=1 TO 5
3590 J=INT(9*RND)+1
3600 IF A$(J)="" THEN 3590
3610 B$=""
3620 CALL HCHAR(20,1,152,128)
```

```
3630 FOR K=1 TO LEN(A$(J))
3640 CALL HCHAR(21,K+2,ASC(SEG$(A$(J),K,1))
     )
3650 NEXT K
3660 CALL SOUND(150,1397,4)
3670 FOR L=1 TO 28
3680 CALL KEY(0,KEY,S)
3690 IF S<1 THEN 3680
3700 IF KEY=13 THEN 3740
3710 CALL HCHAR(22,L+2,KEY)
3720 B$=B$&CHR$(KEY)
3730 NEXT L
3740 IF B$=A$(J)THEN 3800
3750 I=I-1
3760 CALL SOUND(800,-8,0,110,4)
3770 FOR DELAY=1 TO 500
3780 NEXT DELAY
3790 GOTO 3860
3800 CALL SOUND(100,392,2)
3810 CALL SOUND(100,494,2)
3820 CALL SOUND(100,587,2)
3830 CALL SOUND(100,494,2)
3840 CALL SOUND(100,392,2)
3850 A$(J)=""
3860 NEXT I
3870 GOSUB 3890
3880 RETURN
3890 CALL SOUND(T/2,330,3,262,8,165,10)
3900 CALL SOUND(T/2,349,3,294,8,147,10)
3910 CALL SOUND(T,392,2,330,7,131,10)
3920 CALL SOUND(T/2,349,2,294,7,175,10)
3930 CALL SOUND(T/2,330,1,262,6)
3940 CALL SOUND(1.5*T,294,0,247,6,196,8)
3950 CALL SOUND(T/2,262,1)
3960 CALL SOUND(2*T,262,0,131,10)
3970 CALL SOUND(2*T,294,0,196,8)
3980 CALL SOUND(2*T,330,0,131,8)
3990 CALL SOUND(T,349,0,196,8)
4000 CALL SOUND(T,247,1)
4010 CALL SOUND(4*T,262,1,165,6,131,8)
4020 CALL SOUND(1,9999,30)
4030 RETURN
```

```
4040 CALL HCHAR(23,24,46)
4050 CALL HCHAR(19,20,73)
4060 CALL HCHAR(19,14,84)
4070 CALL HCHAR(21,17,72)
4080 CALL HCHAR(19,10,69)
4090 RETURN
4100 END
```

Timing

Although the TI-99/4A does not have a realtime clock built in and accessible by BASIC, there are ways you can simulate time delays and timing devices. One method of delaying is to use an empty FOR-NEXT loop — one with no statements between FOR and NEXT:

```
100 FOR DELAY=1 TO 100
110 NEXT DELAY
```

The above delay takes about one second. FOR DELAY = 1 TO 1000 takes about 3.8 seconds. The time will vary, depending on how full memory is when your program runs. You can use a stopwatch to determine what the limit on your delay loop should be.

The CALL SOUND Clock

A more accurate way to denote a certain length of time is to use multiple CALL SOUND statements, in which you can specify an exact number of milliseconds for a sound. As you know, a CALL SOUND statement does not usually delay a program. The computer goes on and executes more statements, and the sound has no effect on program speed — with one exception. Until one CALL SOUND has finished, the next one cannot begin. So if your program has one CALL SOUND, a second CALL SOUND, or a repetition of the first one in a loop, will cause the program to wait for exactly as long as you specify.

If you prefer not to hear anything during the delay, use a frequency out of hearing range and a loudness factor of 30.

```
100 CALL SOUND(1000,44000,30)
110 PRINT I
```

```
120 I=I+1
130 GOTO 100
```

Since 1000 milliseconds equals one second, this program segment increments every second.

The CALL KEY Clock

Another way to simulate a time clock while someone is interacting with the computer is to put a counter in the CALL KEY loop. In the game "Find Home" in Chapter 3, the score is incremented in the CALL KEY loop and PRINTed at the end of the game. The faster you play and get to the home base, the fewer times the CALL KEY loop will occur, and the lower the score will be.

Here is a routine you can use if you want the time printed as the game is going. In this program segment, the time prints until you press a key:

```
100 CALL CLEAR
110 T=0
120 CALL KEY(0,K,S)
130 FOR I=1 TO LEN(STR$(T))
140 CALL HCHAR(24,27+I,ASC(SEG$(STR$(T),I,
    1)))
150 NEXT I
160 T=T+1
170 IF S<1 THEN 120
180 END
```

The disadvantage of this method is that the more statements you have within the CALL KEY loop, the less responsive user interaction will be, as the user presses a key and nothing happens for a while.

Timing the Touch Typist

This program was designed as a practice unit for beginning typing students who have learned all the positions of the letters. Each drill consists of ten sentences, and each sentence requires 25 keystrokes. After each sentence the student types, his or her approximate rate in words per minute is calculated and displayed. The rate is calculated from the number of strokes the student typed divided by five strokes per word.

After ten sentences, the overall words-per-minute rate for all ten sentences is calculated and displayed.

The sentences for the drill are chosen randomly from a list of over 40 sentences, and a student can perform the drill four times before repeating a sentence.

The sentences are read in as the A$ array. For each sentence chosen in the drill, a random number *H* is selected and A$(H) is printed. After the sentence has been used, A$(H) is set equal to the null string, "", so it cannot be used again. If the student wishes to continue after the fourth drill, the sentences are RESTOREd and READ into the A$ array before the next drill.

After a sentence is displayed, the student types in the sentence. Each letter the student types is accepted using a CALL KEY loop. INPUT would be faster for the student and easier for the programmer, but I did not use it because the screen would scroll, the student could enter too many strokes and cause a program-ending error or change the graphics sequence, and there would be no way to time the process.

Lines 1640-1710 receive the student's typing. The student is allowed to type 27 strokes maximum. *II* is the timer that increments within the loop. If the student presses ENTER, the program branches out of the loop; otherwise, the character pressed is printed and placed in the *B* array.

The timer value, II, is a function of both the amount of actual time a student uses and the number of strokes. It varies directly with the length of time, but does not increment as quickly if a key is pressed.

Testing for the Realtime Value

To discover the relation between the *II* value and the actual word-per-minute rate, a plot of time and number of strokes was made; then an equation could be derived. First, a constant time of four seconds was selected; then a specific number of strokes was entered and the timer value printed. For graphing purposes, 0, 5, 10, 15, 20, and 25 strokes were entered during the constant four seconds. These numbers of strokes correspond to 0, 15, 30, 45, 60, and 75 words per minute. The timer values were consistent for numerous trials, and the points plotted on the graph were in a straight line. Constant times of three seconds and five seconds were also tried, and resulted in parallel lines on the graph.

Since the general equation for a line is $y = mx + b$,

$$II = -\frac{28}{25}x + 78 = -\frac{28}{25}x + 78 * \frac{1}{4 \text{ seconds}} * \frac{60 \text{ seconds}}{1 \text{ minute}} * m$$

where m is minutes. Solving for m,

$$m = \frac{1}{1170}\left(II + \frac{28}{25}x\right)$$

The typing definition of words per minute is

$$\text{WPM} = \frac{\text{number of strokes/5 strokes per word}}{\text{number of minutes}}$$

So, substituting m and simplifying,

$$\text{WPM} = \frac{234\,x}{II + 1.12\,x} \text{ words per minute}$$

The equation will change if the size of the program is altered because, in general, the program runs more slowly with fuller memory.

In this program, the process of using a timer in the CALL KEY loop, printing the character of the key pressed, and keeping track of the key pressed increases the response time. With faster typists, the spaces and double letters are not as easily detected. However, if you type evenly the response of the keys is better, and this program can handle speeds of about 88 WPM accurate typing. Since this program was designed as a drill for students, it should be adequate for beginning typists' speeds.

How "Type-ette Timer" Works

Lines

110	DIMension the A$ array for 46 sentences and the *B* array to receive the 27 character numbers the student types.
120	Branch past the subroutine.
130-200	Subroutine to print message and wait for student to press ENTER.
210-250	Clear the screen; print the title.

260-310	Define the graphics characters.
320-1000	While music plays, define the graphics characters and colors and draw the title screen.
1010-1150	While music plays, READ the sentences into the A$ array.
1160-1240	READ more sentences into the array.
1250	If the student has previously done the drill, skip the instructions.
1260-1320	Print the instruction screen.
1330	Initialize the number of times the drill has been performed.
1340-1470	Draw the graphics for the drill.
1480-1520	Initialize the variables. *R* is the number of correctly typed sentences; *W* is the number of incorrectly typed sentences; *CO* is a coordinate marker for the birds that appear as a sentence is typed correctly; *TOT* is the total timer value; *TLL* is the total number of strokes typed.
1530	Randomize the selection of sentences.
1540	Perform the drill for ten sentences.
1550-1560	Initialize B$ as the student's typing and *II* as the timer.
1570-1580	Randomly choose one of the sentences.
1590	Clear the background for sentences.
1600-1630	Print the sentence and beep the beginning tone.
1640-1710	Print what the student types while incrementing the timer. (1650-1670 comprise the "waiting loop" while the student has *not* pressed a key.)
1720	Beep the ending tone.
1730-1750	Combine the characters the student typed to form sentence B$.
1760-1860	If the sentence was incorrect, sound "uh-oh" and print the number of wrong sentences so far.
1870-1890	If the sentence was correct, play a tune.
1900-1920	Draw a bird.
1930-1990	Print the number of correct sentences.
2000-2110	Calculate and print words per minute.
2120	Set A$ to null so the sentence won't be used again.
2130-2150	Delay slightly before the next problem.
2160	Clear the last sentence typed.

2170-2230	Print the overall average words per minute and play a tune.
2240-2320	Print the option to try again; branch appropriately depending on the answer and the number of times the drill has been performed; end.

Program 6-6. Type-ette Timer

```
100 REM  TYPE-ETTE TIMER
110 DIM A$(46),B(27)
120 GOTO 210
130 M$="PRESS <ENTER> TO CONTINUE."
140 FOR J=1 TO 26
150 CALL HCHAR(24,2+J,ASC(SEG$(M$,J,1)))
160 NEXT J
170 CALL KEY(0,K,S)
180 IF K<>13 THEN 170
190 CALL HCHAR(24,3,32,27)
200 RETURN
210 CALL CLEAR
220 CALL SCREEN(8)
230 CALL COLOR(10,15,1)
240 CALL COLOR(11,15,6)
250 PRINT "{3 SPACES}** TYPE-ETTE TIMER **"
    ::
260 FOR J=1 TO 9
270 READ C,C$
280 CALL CHAR(C,C$)
290 NEXT J
300 DATA 92,3C4299A1A199423C,105,0103070F1F
    3F7FFF,106,0080C0E0F0F8FCFE,120,FFFFF
    FFFFFFFFFFF,121
310 DATA 7F7F7F7F3F3F3F1F,122,1F0F0F0707030
    1,123,7F3F1F0701,124,FFFFFFFFFF7F0F,1
    36,FFFFFFFFFFFFFFFF
320 CALL COLOR(12,12,1)
330 T=400
340 CALL SOUND(T,523,2,131,8)
350 CALL CHAR(96,"FFFFFFFFFFFFFFFF")
360 CALL CHAR(97,"FF7F3F1F0F070301")
```

```
370 CALL SOUND(T/2,659,2)
380 CALL COLOR(9,6,1)
390 CALL SOUND(T/2,523,2)
400 CALL CHAR(98,"FFFEFCF8F0E0C08")
410 CALL SOUND(T,392,1)
420 PRINT TAB(8);"``````````````":TAB(8);"``
    ````````````"
430 CALL SOUND(T,392,0)
440 CALL CHAR(104,"FFFFFFFFFFFFFFFF")
450 CALL CHAR(112,"00000000000C3EFF")
460 CALL SOUND(T,523,2,131,8)
470 PRINT TAB(8);"`pppppppppp`"
480 CALL SOUND(T/2,659,2)
490 CALL CHAR(115,"FF7F3F1F0F070301")
500 CALL SOUND(T/2,523,2)
510 CALL CHAR(116,"FFFEFCF8F0E0C08")
520 CALL SOUND(T,392,2)
530 PRINT TAB(8);"ashhhhhhhhtb":TAB(9);"as
    hhhhhhtb"
540 CALL SOUND(T,784,1)
550 PRINT TAB(10);"ashhhhtb":TAB(11);"ashh
    htb"
560 CALL CHAR(99,"0103070F1F3F7FFF")
570 CALL SOUND(T/2,698,1,131,7)
580 CALL CHAR(100,"80C0E0F0F8FCFEFF")
590 CALL SOUND(T/2,659,1,131,7)
600 PRINT TAB(12);"ashtb"
610 CALL SOUND(T/2,587,1)
620 PRINT TAB(13);"`h`"
630 CALL SOUND(T/2,523,1)
640 PRINT TAB(13);"`h`"
650 CALL SOUND(T/2,494,2,131,7)
660 PRINT TAB(12);"c`h`d"
670 CALL SOUND(T/2,523,2,131,7)
680 PRINT TAB(11);"c``h``d"
690 CALL SOUND(T/2,494,2)
700 PRINT TAB(10);"c```h```d"
710 CALL SOUND(T/2,523,2)
720 PRINT TAB(9);"c````h````d"
730 CALL SOUND(T/2,587,2)
740 CALL CHAR(113,"0103070F1F3F7FFF")
750 CALL SOUND(T/2,523,2)
```
```

```
760 PRINT TAB(8);"c`````h`````d"
770 CALL SOUND(T/2,494,2)
780 CALL CHAR(114,"80C0E0F0F8FCFEFF")
790 CALL SOUND(T/2,440,2)
800 PRINT TAB(8);"``````h``````"
810 CALL SOUND(T/2,392,2,196,3)
820 PRINT TAB(8);"`````qhr`````"
830 CALL SOUND(T/2,175,3)
840 PRINT TAB(8);"````qhhhr````"
850 CALL SOUND(T/2,165,3)
860 CALL SOUND(T/2,147,3)
870 CALL SOUND(T,523,2,131,3)
880 PRINT :;
890 CALL SOUND(T/2,659,2)
900 CALL SOUND(T/2,523,2)
910 CALL SOUND(T,392,0)
920 CALL CHAR(128,"0F0703014060F7F43")
930 CALL SOUND(T,392,2)
940 CALL CHAR(129,"80C0C0E0E6FFFEE")
950 CALL SOUND(T,659,2,262,6)
960 CALL COLOR(13,5,1)
970 CALL SOUND(T/2,784,2)
980 CALL COLOR(14,10,1)
990 CALL SOUND(T/2,659,2)
1000 CALL SOUND(T,523,2)
1010 DATA 659,185,I MADE CAGES FOR MY PETS.
     ,523,185,GREG BOUGHT A LARGE GONG.,58
     7,196,SHE KEPT A SAFE DISTANCE.
1020 DATA 494,196,TOM HAS SENT THE PACKAGE.
     ,523,131,THEY MAY STOP THEIR WORK.,44
     0,131,CHERY HELPED WITH DISHES.
1030 DATA 494,147,ANDY GAVE MY BAND A HAND.
     ,392,147,LET RANDY SORT THE CARDS.,44
     0,147,PUT A PURE GOLD ONE HERE.
1040 DATA 370,147,TRY TO TYPE A THIRD CARD.
     ,392,196,I WANT TO WIN A SURPRISE.,44
     0,196,SHE HAS MORE TO DO THERE.
1050 DATA 494,196,HE SAID WE RENT A CAMPER.
     ,523,196,THEY ARE AT THE TENT NOW.,58
     7,196,I HOPE THE TAX IS FOR US.
1060 DATA 659,196,WE KNOW WE HAVE TO DO IT.
```

```
     ,740,370,DO NOT PHOTO THOSE OBOES.,78
     4,392,THIS IS FUN TO TYPE THEM.
1070 DATA 880,185,YOU COULD COUNT YOUR OWN.
     ,784,185,SHE GAVE HIM A FINE WAGE.,74
     0,147,SOME OF US HAVE TO DO IT.
1080 DATA 659,147,HAVE THE BOY DO THE WORK.
     ,587,220,HE CAN DO A JOB THE BEST.,52
     3,220,EIGHT OF US ARE HERE NOW.
1090 DATA 494,147,YOUR COWS CAN HELP DO IT.
     ,440,147,I BOUGHT THE BOX OF CANS.,39
     2,196,THEY SHOULD READ MY LIST.
1100 DATA 44000,44000,LET ME GET SEVEN OF T
     HEM.
1110 RESTORE 1010
1120 FOR I=0 TO 27
1130 READ F1,F2,A$(I)
1140 CALL SOUND(200,F1,2,F2,6)
1150 NEXT I
1160 FOR I=28 TO 45
1170 READ A$(I)
1180 NEXT I
1190 DATA YOUR BABY IS AWAKE AGAIN.,CHECK T
     HE PAPER FOR DIRT.,THE QUICK QUIZ WAS
     TODAY.
1200 DATA FOUR OF THE MEN ARE HERE.,BRING C
     ARDS TO THE TABLE.,TRY NOT TO BALK AT
     THESE.
1210 DATA SHE MUST TRY TO WORK NOW.,I THINK
     IT IS WISE TO DO.,EACH OF US HAD FIV
     E JARS.
1220 DATA TRY TO GET SEVEN OF THEM.,THEIR C
     AR IS AT THE FARM.,WE HAVE TO WORK TW
     O DAYS.
1230 DATA SHE SAID THEY STRUCK OIL.,I WANT
     TO GO LATER TODAY.,TRY TO REACH THESE
     GOALS.
1240 DATA JANE HAS TO SPEAK AT ONE.,THEY PR
     OVED THE THEORIES.,SHE SAID SHE HAS A
     LEASE.
1250 IF TEST>4 THEN 1330
1260 CALL CLEAR
```

```
1270 CALL SCREEN(2)
1280 PRINT "YOU WILL SEE A SENTENCE":"ON TH
     E SCREEN."::"TYPE AND ENTER IT."::"YO
     U WILL BE TOLD YOUR"
1290 PRINT "WORDS PER MINUTE (WPM)":"FOR TH
     AT SENTENCE":" (25 STROKES = 5 WORDS)
     ."::"AFTER TEN SENTENCES YOUR"
1300 PRINT "FINAL SCORE AND TOTAL WPM
     {3 SPACES}ARE SHOWN."::::
1310 CALL SCREEN(12)
1320 GOSUB 130
1330 TEST=1
1340 CALL CLEAR
1350 CALL SCREEN(8)
1360 PRINT TAB(7);"ij":TAB(6);"ihhj":TAB(5)
     ;"ihhhhj":TAB(5);"```````":TAB(5);"```
     ```"
1370 PRINT TAB(5);"``xx``";TAB(18);"RIGHT:"
1380 PRINT TAB(5);"``xx``";TAB(18);"WRONG:"
1390 PRINT TAB(5);"``xx``";TAB(18);"WPM:"::
     ::::::::
1400 CALL HCHAR(1,29,120,4)
1410 CALL HCHAR(2,29,121)
1420 CALL HCHAR(2,30,120,3)
1430 CALL HCHAR(3,29,122)
1440 CALL HCHAR(3,30,120,3)
1450 CALL HCHAR(4,30,123)
1460 CALL HCHAR(4,31,124)
1470 CALL HCHAR(4,32,120)
1480 R=0
1490 W=0
1500 CO=-1
1510 TOT=0
1520 TLL=0
1530 RANDOMIZE
1540 FOR I=1 TO 10
1550 B$=""
1560 II=0
1570 H=INT(46*RND)
1580 IF A$(H)="" THEN 1570
1590 CALL HCHAR(18,1,136,128)
1600 FOR J=1 TO 25
```
```

```
1610 CALL HCHAR(19,3+J,ASC(SEG$(A$(H),J,1))
     )
1620 NEXT J
1630 CALL SOUND(100,1397,2)
1640 FOR KK=1 TO 27
1650 CALL KEY(0,K,S)
1660 II=II+1
1670 IF S<1 THEN 1650
1680 IF K=13 THEN 1720
1690 CALL HCHAR(20,KK+3,K)
1700 B(KK)=K
1710 NEXT KK
1720 CALL SOUND(100,262,2)
1730 FOR K1=1 TO KK-1
1740 B$=B$&CHR$(B(K1))
1750 NEXT K1
1760 IF B$=A$(H)THEN 1870
1770 CALL SOUND(100,392,2)
1780 CALL SOUND(100,330,2)
1790 W=W+1
1800 IF W<10 THEN 1850
1810 CALL HCHAR(13,27,49)
1820 CALL HCHAR(13,28,48)
1830 CALL HCHAR(12,28,48)
1840 GOTO 2000
1850 CALL HCHAR(13,28,W+48)
1860 GOTO 2000
1870 FOR F1=523 TO 723 STEP 20
1880 CALL SOUND(50,F1,2)
1890 NEXT F1
1900 CO=CO+3
1910 CALL HCHAR(5,CO,128)
1920 CALL HCHAR(5,CO+1,129)
1930 R=R+1
1940 IF R<10 THEN 1990
1950 CALL HCHAR(12,27,49)
1960 CALL HCHAR(12,28,48)
1970 CALL HCHAR(13,28,48)
1980 GOTO 2000
1990 CALL HCHAR(12,28,R+48)
2000 LL=LEN(B$)
2010 TLL=TLL+LL
```

```
2020 TOT=TOT+II
2030 WPM=INT(((234*LL)/(II+1.12*LL))+.5)
2040 WPM$=STR$(WPM)
2050 CALL HCHAR(14,27,32,3)
2060 WC=26
2070 IF WPM>=10 THEN 2090
2080 WC=27
2090 FOR KK=1 TO LEN(WPM$)
2100 CALL HCHAR(14,KK+WC,ASC(SEG$(WPM$,KK,1
     )))
2110 NEXT KK
2120 A$(H)=""
2130 FOR D=1 TO 200
2140 NEXT D
2150 NEXT I
2160 CALL HCHAR(19,1,136,64)
2170 PRINT "TOTAL AVERAGE WPM =";INT(((234*
     TLL)/(TOT+1.12*TLL))+.5)
2180 RESTORE 2190
2190 DATA 587,784,988,1175,1047,988,880,784
     ,740,659,784,740,880,784,740,659,659,
     587,523,494,587,523,494,440,392
2200 FOR I=1 TO 25
2210 READ F1
2220 CALL SOUND(200,F1,2)
2230 NEXT I
2240 PRINT :"WANT TO TRY AGAIN? (Y/N)"
2250 CALL KEY(0,K,S)
2260 IF K=78 THEN 2310
2270 IF K<>89 THEN 2250
2280 TEST=TEST+1
2290 CALL CLEAR
2300 IF TEST>4 THEN 1110 ELSE 1340
2310 CALL CLEAR
2320 END
```

Sorting

One of the functions of a computer is to organize data. There are many sort routines to take your raw data and arrange it in ascending or descending order. The Birthday List program illustrates sorting by date, and the Name and Address File program illustrates sorting names alphabetically. Here are four BASIC algorithms for sorting.

In the first sort program, lines 110-170 find and print 50 random numbers to sort, and lines 500 to the end print out the sorted numbers. To use names or strings, put a dollar sign after each variable name that contains an item to be sorted. To make the program sort in descending rather than ascending order, change the less than (<) and greater than (>) signs.

Sort 1 is the bubble, or simple interchange, sort. It's better for lists that are not much out of order or that haven't very many items. The program compares each number to the next number and exchanges numbers where necessary. If even one switch has been made during a pass through all the numbers, the loop of comparisons starts over. The number of passes through the loop depends on how many items were out of order.

Program 6-7. Sort 1: Bubble Sort

```
1ØØ REM  SORT 1
11Ø DIM A(5Ø)
12Ø FOR I=1 TO 5Ø
13Ø RANDOMIZE
14Ø A(I)=INT(RND*1ØØ+1)
15Ø PRINT A(I);
16Ø NEXT I
17Ø PRINT ::
2ØØ LIM=49
21Ø SW=Ø
22Ø FOR I=1 TO LIM
23Ø IF A(I)<=A(I+1)THEN 29Ø
24Ø AA=A(I)
25Ø A(I)=A(I+1)
26Ø A(I+1)=AA
27Ø SW=1
28Ø LIM=I
```

```
290 NEXT I
300 IF SW=1 THEN 210
500 FOR I=1 TO 50
510 PRINT A(I);
520 NEXT I
530 END
```

The shell sort is considerably faster than the bubble sort, because the number of comparisons that need to be made is reduced. In general, for a random order of 50 numbers, the shell sort is about two or three times as fast as the bubble sort.

In an array of *N* numbers, first determine *B* so that $2^B < N < 2^{B+1}$. Then initialize *B* to 2^{B-1}. The loop varies counter *I* from 1 to *N*–*B*. First, check if $A(I) = < A(I+B)$. If so, increment *I* and continue comparisons. If not, exchange $A(I)$ and $A(I+B)$ and change the subscript.

When *I* reaches the value of *N*, reduce *B* by a factor of 2 and start the loop again. When $B=0$, the sort is complete.

Program 6-8. Sort 2: Shell Sort

```
100 REM  SORT 2
110 DIM A(50)
120 FOR I=1 TO 50
130 RANDOMIZE
140 A(I)=INT(RND*100+1)
150 PRINT A(I);
160 NEXT I
170 PRINT ::
200 B=1
210 B=2*B
220 IF B<=50 THEN 210
230 B=INT(B/2)
240 IF B=0 THEN 500
250 FOR I=1 TO 50-B
260 C=I
270 D=C+B
280 IF A(C)<=A(D)THEN 340
290 AA=A(C)
300 A(C)=A(D)
310 A(D)=AA
320 C=C-B
```

```
330 IF C>0 THEN 270
340 NEXT I
350 GOTO 230
500 FOR I=1 TO 50
510 PRINT A(I);
520 NEXT I
530 END
```

The third is also faster than the first sort if the numbers are quite out of order. The program goes through all the numbers and places the lowest value in the first spot of the array. The loop keeps finding the smallest of the numbers remaining and places it in order.

Program 6-9. Sort 3: Minimum Search

```
100 REM  SORT 3
110 DIM A(50)
120 N=50
130 FOR I=1 TO N
140 RANDOMIZE
150 A(I)=INT(RND*100+1)
160 PRINT A(I);
170 NEXT I
180 PRINT ::
200 M=A(1)
210 IM=1
220 FOR I=2 TO N
230 IF A(I)<M THEN 260
240 M=A(I)
250 IM=I
260 NEXT I
270 AA=A(N)
280 A(N)=A(IM)
290 A(IM)=AA
300 N=N-1
310 IF N>1 THEN 200
500 FOR I=1 TO 50
510 PRINT A(I);
520 NEXT I
530 END
```

In this fourth sort, each pass through the numbers finds both the minimum and maximum numbers and places them at the end points.

Program 6-10. Sort 4: Minimum and Maximum

```
100 REM  SORT 4
110 DIM A(50)
120 N=50
130 FOR I=1 TO 50
140 RANDOMIZE
150 A(I)=INT(RND*100+1)
160 PRINT A(I);
170 NEXT I
180 PRINT ::
200 S=1
210 MN=A(S)
220 IMIN=S
230 MX=MN
240 IMAX=S
250 FOR I=S TO N
260 IF A(I)<=MX THEN 290
270 MX=A(I)
280 IMAX=I
290 IF A(I)>=MN THEN 320
300 MN=A(I)
310 IMIN=I
320 NEXT I
330 IF IMIN<>N THEN 350
340 IMIN=IMAX
350 AA=A(N)
360 A(N)=A(IMAX)
370 A(IMAX)=AA
380 N=N-1
390 AA=A(S)
400 A(S)=A(IMIN)
410 A(IMIN)=AA
420 S=S+1
430 IF N>S THEN 210
500 FOR I=1 TO 50
510 PRINT A(I);
520 NEXT I
530 END
```

Conserving Memory

If you are used to working with programs on large mainframe computers, one of your biggest challenges with a microcomputer may be to stay within the available memory. However, as you work with your computer you'll soon be able to judge about how much programming the 16K RAM can handle without memory problems. (Note: The TI-99/4A console has 256 fewer bytes available than the TI-99/4 console.)

Keep in mind that there are trade-offs in programming. You may have to sacrifice clear documentation, easy-to-read lines, ideal graphics, or even speed and efficiency in order to gain enough memory to RUN. Here are a few hints to help you reduce memory requirements in your programs.

Specify One OPEN File

If the disk system is plugged into the computer, memory is reduced by 534 bytes, plus 518 bytes for each OPEN file. In TI BASIC the number of files open is preset to 3, so 2088 bytes of RAM are used. By specifying only one OPEN file, you use the least amount of memory you can and still have the disk system connected.

When you first sit down at your computer, use this procedure:

1. From the title screen, press any key to begin.
2. Press 1 for TI BASIC.
3. Enter CALL FILES(1)
4. Enter NEW
5. Proceed as usual.

All of the programs in this book will work with the disk system connected; however, several of the programs require the CALL FILES(1) procedure.

Remove or Shorten REM Statements

Deleting REMs is the easiest and perhaps the first step to reduce program size. While you are developing your program, REMs help you keep track of different sections or procedures. However, each REM uses one byte per character, plus the line number.

While you are developing programs, avoid having GOSUB, GOTO, THEN, and ELSE commands branch to the line number of a REM; go to the next statement number instead. Later, if you delete the REM statement, you won't have to worry about changing all references to that line number.

Combine PRINT Statements

One PRINT statement for each line of print on the screen is easy to read and understand in the program listing, but each line number uses more memory. One statement can be 112 characters long, so you can use colons and spaces to combine several lines into one longer one to save memory.

In this example, line 500 does exactly the same thing as lines 100-230.

```
100 PRINT "HELLO"
110 PRINT
120 PRINT
130 PRINT
140 PRINT "CHOOSE:"
150 PRINT
160 PRINT "1 COLORFUL DEMONSTRATION #1"
180 PRINT "2 GAME TWO"
190 PRINT "3 OPTION THREE"
200 PRINT
210 PRINT "4 END PROGRAM"
220 PRINT
230 PRINT
500 PRINT "HELLO":::"CHOOSE:"::"1 COLORFUL
     DEMONSTRATION #1":"2 GAME TWO":
    "3 OPTION THREE"::"4 END PROGRAM":::
```

Plan Your DIMensions

When you use a subscripted variable you haven't already DIMed, TI BASIC automatically reserves space for eleven elements in each dimension used — up to E(10,10,10), when subscripts start with zero. When you RUN the program, even if you don't fill those extra elements, eight bytes per subscript are reserved for numeric expressions.

If you really are using ten or eleven subscripts, a DIM statement wouldn't save you any memory, since the statement itself takes up several bytes. However, if you are close to full memory and you need only six elements, a DIM statement like

120 DIM E(5) will save 40 bytes — eight per numeric element.

For numbers higher than 10, DIM only the number of elements you need. Don't arbitrarily choose a nice round number such as 50 when you really need only 43.

String variable elements don't require as much reserved memory — the subscripted string element is null until it is actually filled. When you use a DIM statement for a string, the process uses eight bytes plus twice the value of each subscript.

Also remember that subscripts start with zero unless you use OPTION BASE 1, which starts subscripts with one.

Trim Your Variables

Limit the number of different variable names used, and use short variable names. While you are developing a program, you may use meaningful variable names, such as NAME$, ADDRESS$, SCORE, DELAY, and COUNTER. However, if you need to conserve memory, you may have to sacrifice clarity to be able to RUN your program. Longer names take up more memory each time they are used than shorter names.

One programming trick is to use the *same* short variable name for each independent loop counter, rather than use several different names, such as FOR MONTH=1 TO 12, FOR DELAY=1 TO 500, or FOR CHARACTER=96 TO 120. For each loop you could use FOR C=1 TO 12, FOR C=1 TO 500, and FOR C=96 TO 120, etc.

Use Subroutines for Repetitious Code

Take a look at your listing and note any repetitious code or sequences of similar statements. Sometimes a FOR-NEXT loop will be more efficient than a sequence. A GOSUB can be used for program segments that are used more than once. You may have a procedure that is used in several different places — write the coding once, then every time you need the procedure use a GOSUB. The same subroutine can be used for different purposes when you "prime" the program before using GOSUB by assigning new values to the variables the subroutine uses.

Use DATA and READ

Using DATA statements may increase typing errors because of all the numbers and commas involved, and the program logic may be harder to follow, but a DATA routine can save a lot of memory by READing values for variables instead of using endless LET statements. Usually, if you have more than eight statements in a row that are doing the same process, using a DATA routine instead would save memory.

Combine Data Where You Can

Rather than working with 20 different last names and their corresponding first names, combine the names into one variable: N$(I)=LAST$&'', ''&FIRST$. Consider whether the data might best be kept a number or a string if you have a choice. In a report program for attendance at nine monthly meetings, the numbers could be READ as DATA: 0,0,3,1,0,0,2,0,1,0,0,1. Stored that way, each digit takes up three bytes of memory. I also decided I wanted to use a symbol other than zero for a person who joined the group late.

The numbers can be READ as individual strings, and in the DATA statement the zeros can be deleted:

500 DATA SMITH,JIM,-,,3,1,,,2,,1,,,1

Another way to arrange this data is to put the zeros back in, READ all the numbers and symbols as one string, and then use a few lines of logic to separate each month's digit:

500 DATA SMITH,JIM,-03100201001

To work with each digit, use the SEG$ statement:

```
200 READ L$,F$,A$
210 FOR M=1 TO 12
220 B$=SEG$(A$,M,1)
        .
        .
        .
290 NEXT M
```

Draw Efficiently

Check your graphics statements to make sure you are drawing your picture in the best time sequence and in an efficient manner. Make good use of the repetition factor in CALL HCHAR and CALL VCHAR statements. Sometimes you can use fewer statements by using the repetition factor and then erasing part of the design.

For example, to print REGENA vertically, I can use five statements for the six letters:

```
100 CALL HCHAR(15,25,82)
110 CALL VCHAR(16,25,69,3)
120 CALL VCHAR(17,25,71)
130 CALL VCHAR(19,25,78)
140 CALL VCHAR(20,25,65)
```

Keep in mind that the repetitions will continue to the next line if you want them to. Here are three ways to clear a rectangle of the screen; the third method uses the least memory.

```
500 CALL HCHAR(15,1,32,32)
510 CALL HCHAR(16,1,32,32)
520 CALL HCHAR(17,1,32,32)
530 CALL HCHAR(18,1,32,32)
540 CALL HCHAR(19,1,32,32)
550 CALL HCHAR(20,1,32,32)
```

or

```
500 FOR R=15 TO 20
510 CALL HCHAR(R,1,32,32)
520 NEXT R
```

or

```
500 CALL HCHAR(15,1,32,192)
```

Plan Your Logic Carefully

Be careful of overusing GOTO statements. Structured programming experts never use a GOTO. If you plan the sequence of your program, you should be able to rearrange your program lines so that the program executes in sequence rather than jumping all over and back again. Use the RES command if you need more line numbers between statements. Your program will also be much more understandable if you do not GOTO often.

Check your IF-THEN-ELSE statements. Perhaps an IF-THEN-ELSE can be used instead of an IF-THEN and a GOTO. Remember the power of the ON-GOTO and ON-GOSUB statements. You may be able to reduce many lines of IF-THEN logic if you can get a numeric expression to relate to consecutive integer conditions for the ON-GOTO or ON-GOSUB.

One problem with ON-GOTO statements is that if ON tries to evaluate an expression with a value less than 1 or greater than the number of line numbers after GOTO, the program will crash. Unfortunately, true expressions return a value of -1 and false ones a value of 0, both out of the allowable range for ON.

In order to replace IF-THENs with ON-GOTOs, you can use the *negative* of an expression: -(N=N) has a value of 1. This short program contains a safe ON-GOTO statement:

```
100 INPUT H
110 ON 1-(H<0)-2*(H=O) GOTO 120,140,160
120 PRINT "POSITIVE NUMBER."
130 GOTO 100
140 PRINT "NEGATIVE NUMBER."
150 GOTO 100
160 PRINT "ZERO."
170 GOTO 100
```

The 1 at the beginning of the statement acts like an ELSE statement. No matter what value the other two expressions return, the ON will have somewhere to branch — in this case, line 120. There is no value of H that will crash the program. Notice that because true statements return a negative value, the subtractions in line 110 actually *add* if the expression that follows is true.

Scrutinize the Program Listing

Even if you have a knack for keeping track of things in your program, it is a good idea to take a final look at the listing. Sometimes in the process of editing you miss statements that should have been deleted. Perhaps there are statements that the program never branches to.

Check all GOTO statements and other branching statements to make sure they don't just branch to another GOTO statement, such as IF S=10, THEN 500 ELSE 600, when line 600 says GOTO 1000.

As you look at the listing, you may notice patterns or repetitious code that you overlooked previously. Review the built-in functions of TI BASIC, because a single function may be able to replace several lines of IF-THEN logic.

If It Still Won't Fit

If all else fails, write another program. Change your approach entirely, or write a series of programs. The 16K RAM is a good size for one learning unit in educational programs. Rather than worry about fitting an entire educational program into 16K, it works better to plan the programs in logical teaching units, without sacrificing graphics or sound or color because of memory limitations.

For example, the touch typing programs started out as one idea, but developed into seven units — seven programs. Another example is the geography programs to learn the states. The ideal program would have all 50 states and be able to blink one state at a time. With high resolution graphics that wasn't possible, so the programs were divided up into the regions of the United States. "Western States" (Program 6-3) is one unit.

File Processing

You can use either a cassette recorder or a disk system for file processing with your computer, and store programs or data on cassettes or diskettes. For cassette files you may use either one or two cassettes and the dual cassette cable. The remote switch capabilities are necessary for file processing.

In general, if you have a program that processes stored information, your program will require an OPEN statement to alert the computer which device and what type of data file you are using. Then you may INPUT to read data or PRINT to save data. The program should also include a CLOSE statement.

Keeping Data on Cassette

Here is a program that illustrates the use of a cassette to store information in a name and address file. The first menu screen gives the following options:

1 Load data
2 Add data
3 Edit data
4 Print list
5 Save list
6 End program

The first time you use the program, you would press 2 for "Add data." You may then enter names, addresses, and phone numbers. There is also a field for a code. You can set up the code however you wish. For example, you may want three characters in the code — the first character for the number of children, the second character for which club the person belongs to, and the third character for *Y* if you received a Christmas card last year and *N* if you didn't. Bob's code may then be 4JY for "four children," "jogging club," and "yes." Jim's code may be 3CN for "three children," "computer club," and "no."

This code section is quite versatile. If you are keeping a separate cassette list for computer owners, the codes could be TI, TRS, APPLE, VIC, and ATARI — or you might want to use numbers such as 1 for TI, 2 for TRS, 3 for Apple, etc. You may wish to use codes to determine which region of the country the person lives in. Or you may wish to use codes to tell you which people in your advertising list have purchased items from you and which have not.

If you need to change the information, choose the "Edit" option. The program will prompt you so that you can change any part of the information or delete a name from the list.

Before too long, you will want to save the list. The program will first alphabetize the list by name. The computer will prompt you for the procedure to save the list on cassette.

Option 4 is to print the list; you may print the list either on the screen or with a printer. If you use a printer, you will need to enter your printer configuration, such as RS232.TW.BA=110 or whatever your usual configuration is.

With the print option you may also choose whether to print the whole list or just the people who fit a certain code. If you want to select by code, you enter the code. Using the above example of codes, suppose I'm planning a party for all my close friends who like to jog with their four children. I would enter the code 4JY, and the computer would print a list of all the people who have a code of 4JY.

If you are printing on the screen, the names will scroll. To stop the scrolling at any point, just press any key. To continue the list, press any key.

After you have saved your data once, the next time you run the program you can select the first option, "Load data," to read in previously stored information.

How "Name and Address File" Works

Lines	
100-140	Print the title.
150	DIM variables for 25 names, addresses, phone numbers, and codes.
160-230	Print the main options and branch appropriately.
240-710	Subroutine to alphabetize the list by name. If the list has been alphabetized, FLAG=1.
720-800	Procedure for reading in data. Line 730 OPENs device #1, the cassette CS1, for input with internal and fixed format for a length of 128. First the number of names (*N*) is read in, then all the information.
810-1150	Procedure for adding data. As soon as you add a name, FLAG=2, because the names may be out of alphabetical order. The last name and first name are combined for N$(*n*). The street address, city, state, and zip code are combined for A$(*n*).
1160-2110	Procedure to edit information. POS and SEG$ are used to work with parts of the name and the address.
2120	Before the list is printed, it is alphabetized.
2130-2450	Procedure to print the list. If all the names are to be printed, the code is ZZZZ and the code is not checked. Otherwise, the codes are compared before printing.
2460-2600	Procedure to print the list using the printer.
2610	Before the list is saved, it is alphabetized.
2620-2690	Procedure to save the information on cassette.
2700-2760	Procedure if the user selects the "End program" option. The user is first reminded to save the information. End.

Program 6-11. Name and Address File (Cassette)

```
100 REM  NAME & ADDRESS FILE
110 CALL CLEAR
120 PRINT "NAME AND ADDRESS FILE"
130 CALL CHAR(64,"3C4299A1A1994237")
```

```
140 PRINT ::::::
150 DIM N$(25),A$(25),P$(25),C$(25)
160 GOTO 180
170 CALL CLEAR
180 PRINT "PRESS:"::"  1 LOAD DATA"::"  2 A
    DD DATA"::"  3 EDIT DATA"
190 PRINT :"  4 PRINT LIST"::"  5 SAVE LIST
    "::"  6 END PROGRAM"
200 CALL KEY(0,K,S)
210 IF (K<49)+(K>54)THEN 200
220 CALL CLEAR
230 ON K-48 GOTO 720,810,1170,2120,2610,270
    0
240 PRINT ::"--ALPHABETIZING NAMES--"
250 IF FLAG=1 THEN 700
260 NN=N
270 S=1
280 MN$=N$(S)
290 IMIN=S
300 MX$=MN$
310 IMAX=S
320 FOR I=S TO NN
330 IF N$(I)<=MX$ THEN 360
340 MX$=N$(I)
350 IMAX=I
360 IF N$(I)>=MN$ THEN 390
370 MN$=N$(I)
380 IMIN=I
390 NEXT I
400 IF IMIN<>NN THEN 420
410 IMIN=IMAX
420 AA$=N$(NN)
430 N$(NN)=N$(IMAX)
440 N$(IMAX)=AA$
450 AA$=A$(NN)
460 A$(NN)=A$(IMAX)
470 A$(IMAX)=AA$
480 AA$=P$(NN)
490 P$(NN)=P$(IMAX)
500 P$(IMAX)=AA$
510 AA$=C$(NN)
520 C$(NN)=C$(IMAX)
```

```
530 C$(IMAX)=AA$
540 NN=NN-1
550 AA$=N$(S)
560 N$(S)=N$(IMIN)
570 N$(IMIN)=AA$
580 AA$=A$(S)
590 A$(S)=A$(IMIN)
600 A$(IMIN)=AA$
610 AA$=P$(S)
620 P$(S)=P$(IMIN)
630 P$(IMIN)=AA$
640 AA$=C$(S)
650 C$(S)=C$(IMIN)
660 C$(IMIN)=AA$
670 S=S+1
680 IF NN>S THEN 280
690 FLAG=1
700 CALL CLEAR
710 RETURN
720 PRINT "READING IN DATA"::
730 OPEN #1:"CS1",INPUT ,INTERNAL,FIXED 128
740 INPUT #1:N
750 FOR I=1 TO N
760 INPUT #1:N$(I),A$(I),P$(I),C$(I)
770 NEXT I
780 CLOSE #1
790 FLAG=1
800 GOTO 170
810 PRINT "ADDING DATA"
820 IF N<25 THEN 860
830 PRINT "SORRY, THIS PROGRAM IS FOR":"UP
    TO 25 NAMES, AND YOU HAVEENTERED YOUR
    QUOTA.":::"PRESS ANY KEY."
840 CALL KEY(0,K,S)
850 IF S=1 THEN 170 ELSE 840
860 FLAG=2
870 PRINT :"ENTER 'E' TO EXIT"::
880 INPUT "LAST NAME: ":LN$
890 IF LN$="E" THEN 170
900 INPUT "FIRST NAME: ":FN$
910 IF FN$="E" THEN 170
920 PRINT :"STREET ADDRESS:"
```

```
930 INPUT AA$
940 IF AA$="E" THEN 170
950 INPUT "CITY: ":CC$
960 IF CC$="E" THEN 170
970 INPUT "STATE: ":S$
980 IF S$="E" THEN 170
990 INPUT "ZIP CODE: ":Z$
1000 IF Z$="E" THEN 170
1010 INPUT "PHONE: ":PP$
1020 IF PP$="E" THEN 170
1030 INPUT "CODE: ":C1$
1040 IF C1$="E" THEN 170
1050 PRINT :"IS THE ABOVE INFORMATION":"COR
     RECT? (Y/N)"
1060 CALL KEY(0,K,S)
1070 IF K=89 THEN 1090
1080 IF K=78 THEN 810 ELSE 1060
1090 N=N+1
1100 N$(N)=LN$&", "&FN$
1110 A$(N)=AA$&"/"&CC$&", "&S$&"  "&Z$
1120 P$(N)=PP$
1130 C$(N)=C1$
1140 CALL CLEAR
1150 GOTO 810
1160 CALL CLEAR
1170 PRINT "EDIT DATA"
1180 PRINT ::"PRESS:"::"  1 DELETE A NAME":
     "  2 CHANGE NAME"
1190 PRINT "  3 CHANGE ADDRESS":"  4 CHANGE
     PHONE":"  5 CHANGE CODE":"  6 RETURN
     TO MAIN MENU"
1200 CALL KEY(0,K,S)
1210 IF (K<49)+(K>54)THEN 1200
1220 IF K=54 THEN 170
1230 ED=K-48
1240 PRINT
1250 INPUT "LAST NAME? ":LN$
1260 INPUT "FIRST NAME? ":FN$
1270 PRINT
1280 EDN$=LN$&", "&FN$
1290 FOR I=1 TO N
1300 IF N$(I)=EDN$ THEN 1370
```

```
1310 NEXT I
1320 PRINT :"SORRY, THAT NAME NOT FOUND."
1330 PRINT :"PRESS:  1 EDIT":TAB(9);"2 PRIN
     T LIST":TAB(9);"3 GO TO MAIN MENU"
1340 CALL KEY(0,K,S)
1350 IF (K<49)+(K>51)THEN 1340
1360 ON K-48 GOTO 1160,2120,170
1370 PRINT N$(I)
1380 P=POS(A$(I),"/",1)
1390 AA$=SEG$(A$(I),1,P-1)
1400 PRINT AA$
1410 A2$=SEG$(A$(I),P+1,LEN(A$(I)))
1420 PRINT A2$
1430 PRINT P$(I)
1440 PRINT C$(I)::
1450 ON ED GOTO 1460,1580,1690,1910,1960
1460 PRINT "PRESS 'D' TO DELETE NAME":"
     {6 SPACES}'1' TO RETURN TO MENU"
1470 CALL KEY(0,K,S)
1480 IF K=49 THEN 170
1490 IF K<>68 THEN 1470
1500 FOR J=I TO N-1
1510 N$(J)=N$(J+1)
1520 A$(J)=A$(J+1)
1530 P$(J)=P$(J+1)
1540 C$(J)=C$(J+1)
1550 NEXT J
1560 N=N-1
1570 GOTO 1160
1580 P=POS(N$(I),",",1)
1590 L$=SEG$(N$(I),1,P-1)
1600 F$=SEG$(N$(I),P+1,LEN(N$(I)))
1610 INPUT "LAST NAME: ":LN$
1620 IF LN$="" THEN 1640
1630 L$=LN$
1640 INPUT "FIRST NAME: ":FN$
1650 IF FN$="" THEN 1670
1660 F$=FN$
1670 N$(I)=L$&", "&F$
1680 FLAG=2
1690 PRINT :"PRESS <ENTER> IF DATA IS OK"::
1700 PRINT AA$:"STREET ADDRESS: "
```

```
1710 INPUT AA1$
1720 IF AA1$="" THEN 1740
1730 AA$=AA1$
1740 PRINT :A2$
1750 INPUT "CITY: ":CC$
1760 P=POS(A2$,",",1)
1770 MN$=SEG$(A2$,1,P-1)
1780 MX$=SEG$(A2$,P+2,LEN(A2$))
1790 IF CC$="" THEN 1810
1800 MN$=CC$
1810 INPUT "STATE: ":S$
1820 P=POS(MX$,"  ",1)
1830 PS$=SEG$(MX$,1,P-1)
1840 PZ$=SEG$(MX$,P+2,LEN(MX$))
1850 IF S$="" THEN 1870
1860 PS$=S$
1870 INPUT "ZIP CODE: ":Z$
1880 IF Z$="" THEN 1900
1890 PZ$=Z$
1900 A$(I)=AA$&"/"&MN$&", "&PS$&"  "&PZ$
1910 PRINT :"PRESS <ENTER> IF DATA IS OK"
1920 PRINT P$(I)
1930 INPUT "PHONE NUMBER: ":PP$
1940 IF PP$="" THEN 1960
1950 P$(I)=PP$
1960 PRINT :"PRESS <ENTER> IF DATA IS OK"
1970 PRINT C$(I)
1980 INPUT "CODE: ":C1$
1990 IF C1$="" THEN 2010
2000 C$(I)=C1$
2010 CALL CLEAR
2020 PRINT N$(I)
2030 P=POS(A$(I),"/",1)
2040 PRINT SEG$(A$(I),1,P-1)
2050 PRINT SEG$(A$(I),P+1,LEN(A$(I)))
2060 PRINT :P$(I)
2070 PRINT :C$(I)
2080 PRINT :"INFORMATION CORRECT? (Y/N)"
2090 CALL KEY(0,K,S)
2100 IF K=89 THEN 170
2110 IF K=78 THEN 1580 ELSE 2090
2120 GOSUB 240
```

```
2130 PRINT "PRESS: "::"  1 PRINT COMPLETE L
     IST"::"  2 SELECT BY CODE"
2140 CALL KEY(0,K,S)
2150 IF (K<49)+(K>50)THEN 2140
2160 IF K=50 THEN 2190
2170 CODE$="ZZZZ"
2180 GOTO 2210
2190 PRINT ::"ENTER DESIRED CODE"
2200 INPUT CODE$
2210 CALL CLEAR
2220 PRINT "PRESS: "::"  1 PRINT LIST ON SC
     REEN"::"  2 PRINT LIST ON PRINTER"
2230 CALL KEY(0,K,S)
2240 IF K=50 THEN 2460
2250 IF K<>49 THEN 2230
2260 CALL CLEAR
2270 PRINT "PRESS ANY KEY TO PAUSE;":"PRESS
      ANY KEY TO RESUME."::
2280 FOR I=1 TO N
2290 IF CODE$="ZZZZ" THEN 2310
2300 IF CODE$<>C$(I)THEN 2410
2310 PRINT N$(I)
2320 P=POS(A$(I),"/",1)
2330 PRINT SEG$(A$(I),1,P-1)
2340 PRINT SEG$(A$(I),P+1,LEN(A$(I)))
2350 PRINT :P$(I)
2360 PRINT :C$(I):::
2370 CALL KEY(0,K,S)
2380 IF S=0 THEN 2410
2390 CALL KEY(0,K,S)
2400 IF S<>1 THEN 2390
2410 NEXT I
2420 PRINT "END OF LIST.":"PRESS ANY KEY."
2430 CALL KEY(0,K,S)
2440 IF S<>1 THEN 2430
2450 GOTO 170
2460 PRINT ::"PLEASE LIST PRINTER":"CONFIGU
     RATION."
2470 INPUT CON$
2480 OPEN #3:CON$
2490 FOR I=1 TO N
2500 IF CODE$="ZZZZ" THEN 2520
```

```
2510 IF CODE$<>C$(I)THEN 2580
2520 PRINT #3:N$(I)
2530 P=POS(A$(I),"/",1)
2540 PRINT #3:SEG$(A$(I),1,P-1)
2550 PRINT #3:SEG$(A$(I),P+1,LEN(A$(I)))
2560 PRINT #3::P$(I)
2570 PRINT #3::C$(I):::
2580 NEXT I
2590 CLOSE #3
2600 GOTO 170
2610 GOSUB 240
2620 PRINT "SAVING DATA"::
2630 OPEN #2:"CS1",OUTPUT,INTERNAL,FIXED 12
     8
2640 PRINT #2:N
2650 FOR I=1 TO N
2660 PRINT #2:N$(I),A$(I),P$(I),C$(I)
2670 NEXT I
2680 CLOSE #2
2690 GOTO 170
2700 PRINT "IF YOU END PROGRAM YOU LOSE"::"
     ALL DATA."
2710 PRINT ::"PRESS:"::"  1 SAVE DATA"::"
     2 END PROGRAM"
2720 CALL KEY(0,K,S)
2730 IF (K<49)+(K>50)THEN 2720
2740 CALL CLEAR
2750 IF K=49 THEN 2610
2760 END
```

Using a Printer

One of the first peripherals you may want to add is a printer. Texas Instruments sells a 30-column thermal printer that attaches to the side of the computer. To use any other kind of printer, such as a dot-matrix or letter quality printer, you will need the RS-232 Interface. This is a serial interface that makes printers compatible with the computer. If you use the Peripheral Expansion Box method for accessories, you will need the RS-232 card to use a printer.

Perhaps one of the main uses of a printer is to get a listing of your program. The manual that comes with the RS-232 discusses your *printer configuration,* which you need to specify in order to use your printer. To list a whole program on the printer, here are some sample configurations:

LIST "RS232.TW.BA=110"	(teletype)
LIST "RS232.BA=600"	(TI 825 or TI 840 printer)
LIST "RS232.BA=9600.DA=8"	(Epson MX 80)

To list only certain program lines on your printer, use a colon and the range of line numbers:

LIST "RS232.TW.BA=110":250-350

Using OPEN with Your Printer

You use the same printer configuration when you use your printer during the run of a program. As with disk drives and cassette recorders, you will need an OPEN statement to open a certain device number; then you can PRINT to that device. For example:

```
100 OPEN #1:"RS232.BA=600"
110 PRINT #1:TAB(14);"TITLE OF REPORT"
```

A plain PRINT statement will print to the screen, and PRINT with a number will print to the device that it has been assigned to. You may number your devices as you wish, and you may have several devices open at once. This program will print HELLO on the screen, speak the word, and then print HELLO on the printer.

```
100 OPEN #1:"RS232.BA=600"
110 OPEN #2:"SPEECH",OUTPUT
120 PRINT "HELLO"
130 PRINT #1:"HELLO"
140 PRINT #2:"HELLO"
150 CLOSE #1
160 CLOSE #2
170 END
```

You should always CLOSE the device when you have finished using it, or at the end of the program.

Getting a Hard Copy

Suppose you were in the market for a house and had to borrow money. For various amounts of money borrowed, and different percentage rates, this program calculates what the monthly payment would be over various time spans. Before you try this program, be sure to put the appropriate printer configuration in line 110.

Program 6-12. Monthly Payments

```
100 REM  MONTHLY PAYMENTS
110 OPEN #1:"RS232.BA=600"
120 PRINT #1:TAB(25);"MONTHLY PAYMENTS"
130 FOR AMT=40000 TO 80000 STEP 5000
140 PRINT #1::"AMOUNT BORROWED: $";AMT
150 PRINT #1::"YEARS{8 SPACES}10%
    {7 SPACES}11%{7 SPACES}12%{7 SPACES}13%
    {7 SPACES}14%{7 SPACES}15%"
160 PRINT #1:"-----{8 SPACES}---{7 SPACES}-
    --{7 SPACES}---{7 SPACES}---{7 SPACES}-
    --{7 SPACES}---"::
170 FOR YRS=10 TO 30 STEP 5
180 PRINT #1:YRS;
190 T=2
200 FOR I=10 TO 15
210 II=I/1200
220 N=YRS*12
230 F=(1+II)^N
240 M=AMT*(II*F/(F-1))
250 M=(INT(100*(M+.005)))/100
260 M$=STR$(M)
270 P=POS(M$,".",1)
280 IF P<>0 THEN 310
290 M$=M$&".00"
300 GOTO 330
310 IF LEN(M$)-P=2 THEN 330
320 M$=M$&"0"
330 IF LEN(M$)=7 THEN 350
340 T=T+1
350 T=T+9
360 PRINT #1:TAB(T);M$;
```

```
370 IF LEN(M$)=6 THEN 390
380 T=T+1
390 NEXT I
400 PRINT #1
410 NEXT YRS
420 PRINT #1:::
430 NEXT AMT
440 CLOSE #1
450 END
```

Chapter 7

A Dozen More Programs

A Dozen More Programs

Division

Usually you can use a calculator to check students' homework that involves calculations. However, if the problem is division, the calculator will return an answer with the decimal equivalent of the remainder. This program asks the student to enter the dividend and the divisor and will give the answer as a quotient with a remainder. Notice how the INT function is used. All the calculating is done in lines 240 and 250.

Program 7-1. Division with Remainder

```
110 REM DIVISION WITH REMAINDER
120 CALL CHAR(37,"8040202020204 08")
130 CALL CHAR(38,"00000000000000FF")
140 CALL CLEAR
150 PRINT "DIVISION WITH REMAINDER"::::
160 PRINT TAB(10);"QUOTIENT"
170 PRINT TAB(9);"&&&&&&&&&&"
180 PRINT " DIVISOR%DIVIDEND"::::
190 INPUT "DIVIDEND: ":D
200 INPUT "DIVISOR:  ":I
210 IF I<>0 THEN 240
220 PRINT :"SORRY, DIVISOR CANNOT = 0"::
230 GOTO 200
240 Q=INT(D/I)
250 R=D-Q*I
260 PRINT :"QUOTIENT =";Q;" R";R
270 PRINT :::"PRESS 1 FOR ANOTHER PROBLEM"
280 PRINT TAB(7);"2 TO END PROGRAM"
290 CALL KEY(0,K,S)
300 IF K=49 THEN 140
```

```
310 IF K<>50 THEN 290
320 CALL CLEAR
330 END
```

Equivalent Fractions

This program can quickly find the unknown in problems such as 1/2=?/8. The fractions are of the form:

$$\frac{A}{B} = \frac{C}{D}$$

The student first presses the letter for the unknown, then enters values for the other three numbers. The equivalent fractions will be printed.

Program 7-2. Equivalent Fractions

```
100 REM  EQUIVALENT FRACTIONS
110 REM  ANSWERS ROUNDED TO TWO DECIMAL PLA
    CES
120 CALL CLEAR
130 PRINT TAB(10);"A{5 SPACES}C"
140 PRINT TAB(10);"-  =  -"
150 PRINT TAB(10);"B{5 SPACES}D":::
160 PRINT "WHICH IS THE UNKNOWN?"
170 PRINT "CHOOSE A, B, C, OR D.":::
180 CALL KEY(0,K,S)
190 IF (K<65)+(K>68)THEN 180
200 ON K-64 GOTO 210,260,310,360
210 INPUT "ENTER B = ":B
220 INPUT "ENTER C = ":C
230 INPUT "ENTER D = ":D
240 A=INT(100*(B*C/D+.005))/100
250 GOTO 400
260 INPUT "ENTER A = ":A
270 INPUT "ENTER C = ":C
280 INPUT "ENTER D = ":D
290 B=INT(100*(A*D/C+.005))/100
300 GOTO 400
310 INPUT "ENTER A = ":A
320 INPUT "ENTER B = ":B
330 INPUT "ENTER D = ":D
```

```
340 C=INT(100*(A*D/B+.005))/100
350 GOTO 400
360 INPUT "ENTER A = ":A
370 INPUT "ENTER B = ":B
380 INPUT "ENTER C = ":C
390 D=INT(100*(B*C/A+.005))/100
400 CALL CLEAR
410 PRINT TAB(7);A,C
420 PRINT TAB(7);"----- = -----"
430 PRINT TAB(7);B,D
440 PRINT :::"PRESS 1 FOR ANOTHER PROBLEM"
450 PRINT TAB(7);"2 TO END PROGRAM"
460 CALL KEY(0,K,S)
470 IF K=49 THEN 120
480 IF K<>50 THEN 460
490 CALL CLEAR
500 END
```

Simplifying Fractions

Enter a numerator, then a denominator. The computer simplifies or reduces the fraction to its lowest terms, or tells if it cannot be simplified. This algorithm first checks which is larger, the numerator or the denominator; the first factor to be checked is the smaller number. If either the numerator or the denominator is an odd number, then even factors will be eliminated by choosing a step size of –2 in the checking loop.

Although students usually reduce fractions starting with the smallest factors, the computer starts with the largest possible factor and decreases for each check.

Program 7-3. Simplifying Fractions

```
100 REM  SIMPLIFYING FRACTIONS
110 CALL CLEAR
120 PRINT "** SIMPLIFYING FRACTIONS **"::::
130 INPUT "NUMERATOR ={3 SPACES}":N
140 INPUT "DENOMINATOR = ":D
150 IF D>N THEN 180
160 LIM=D
170 GOTO 190
```

```
180 LIM=N
190 S=-2
200 IF D/2<>INT(D/2)THEN 220
210 IF N/2=INT(N/2)THEN 230
220 S=-1
230 FOR C=LIM TO 2 STEP S
240 A=N/C
250 IF A<>INT(A)THEN 280
260 B=D/C
270 IF B=INT(B)THEN 310
280 NEXT C
290 PRINT ::N;"/";D;" CANNOT BE SIMPLIFIED"
300 GOTO 320
310 PRINT :N;"/";D;" = ";A;"/";B
320 PRINT :::"PRESS 1 FOR ANOTHER PROBLEM"
330 PRINT TAB(7);"2 TO STOP PROGRAM"
340 CALL KEY(0,K,S)
350 IF K=49 THEN 110
360 IF K<>50 THEN 340
370 CALL CLEAR
380 END
```

Multiplying Fractions

This program multiplies from two to nine fractions. First press the total number of fractions, then enter each numerator and denominator. The program multiplies the fractions and simplifies the final answer.

If you have more than nine fractions, either change this program to allow more fractions or run the program in steps.

Program 7-4. Multiplying Fractions

```
100 REM  MULTIPLYING FRACTIONS
110 CALL CLEAR
120 PRINT "** MULTIPLYING FRACTIONS **"::::
130 PRINT "HOW MANY FRACTIONS?":::
140 CALL KEY(0,K,S)
150 IF (K<50)+(K>57)THEN 140
160 CALL HCHAR(21,23,K)
170 C=K-48
```

```
180 NT=1
190 DT=1
200 FOR I=1 TO C
210 PRINT "FRACTION";I
220 INPUT "{4 SPACES}NUMERATOR ={3 SPACES}"
    :N(I)
230 NT=NT*N(I)
240 INPUT "{4 SPACES}DENOMINATOR = ":D(I)
250 IF D(I)<>Ø THEN 28Ø
260 PRINT :"DENOMINATOR CANNOT BE ZERO."::
270 GOTO 24Ø
280 DT=DT*D(I)
290 NEXT I
300 PRINT :::"** MULTIPLY **"::
310 FOR I=1 TO C
320 PRINT STR$(N(I));"/";STR$(D(I))
330 NEXT I
340 PRINT "---------------"
350 FOR I=1 TO C
360 A=NT/D(I)
370 IF A<>INT(A)THEN 42Ø
380 B=DT/D(I)
390 IF B<>INT(B)THEN 42Ø
400 NT=A
410 DT=B
420 NEXT I
430 SW=Ø
440 FOR I=1 TO C-1
450 IF D(I)<=D(I+1)THEN 5ØØ
460 DD=D(I)
470 D(I)=D(I+1)
480 D(I+1)=DD
490 SW=1
500 NEXT I
510 IF SW=1 THEN 43Ø
520 L=D(C)
530 FOR I=L TO 2 STEP -1
540 A=NT/I
550 IF A<>INT(A)THEN 58Ø
560 B=DT/I
570 IF B=INT(B)THEN 61Ø
580 NEXT I
```

```
590 A=NT
600 B=DT
610 IF A>=B THEN 640
620 PRINT ::STR$(A);"/";STR$(B)
630 GOTO 700
640 W=INT(A/B)
650 R=A-W*B
660 IF R<>0 THEN 690
670 PRINT W
680 GOTO 700
690 PRINT W;"{3 SPACES}";STR$(R);"/";STR$(B
    )
700 PRINT :::"PRESS 1 FOR ANOTHER PROBLEM"
710 PRINT TAB(7);"2 TO END PROGRAM";
720 CALL KEY(0,K,S)
730 IF K=49 THEN 110
740 IF K<>50 THEN 720
750 CALL CLEAR
760 END
```

Dividing Fractions

This program divides one fraction by another fraction. The numerators and denominators of each fraction are entered, and the final answer is printed in simplified form.

Program 7-5. Dividing Fractions

```
100 REM  DIVIDING FRACTIONS
110 CALL CLEAR
120 PRINT "THE FIRST FRACTION IS"
130 PRINT "DIVIDED BY THE"
140 PRINT "SECOND FRACTION."
150 PRINT ::TAB(10);"N1/D1"
160 PRINT TAB(9);"-------"
170 PRINT TAB(10);"N2/D2":::
180 INPUT "ENTER N1 = ":N1
190 INPUT "ENTER D1 = ":D1
200 IF D1<>0 THEN 230
210 PRINT :"DENOMINATOR CANNOT BE ZERO.")::
220 GOTO 190
230 PRINT
```

```
240 INPUT "ENTER N2 = ":N2
250 INPUT "ENTER D2 = ":D2
260 IF D2<>Ø THEN 29Ø
270 PRINT :"DENOMINATOR CANNOT BE ZERO."::
280 GOTO 25Ø
290 NT=N1*D2
300 DT=D1*N2
310 PRINT :::STR$(N1);"/";STR$(D1)
320 PRINT "---------------"
330 PRINT STR$(N2);"/";STR$(D2)
340 PRINT :::"EQUALS"::
350 IF NT<DT THEN 38Ø
360 L=DT
370 GOTO 39Ø
380 L=NT
390 FOR I=L TO 2 STEP -1
400 A=NT/I
410 IF A<>INT(A)THEN 44Ø
420 B=DT/I
430 IF B=INT(B)THEN 47Ø
440 NEXT I
450 A=NT
460 B=DT
470 IF A>=B THEN 5ØØ
480 PRINT ::STR$(A);"/";STR$(B)
490 GOTO 59Ø
500 IF B<>1 THEN 53Ø
510 PRINT ::A
520 GOTO 59Ø
530 C=INT(A/B)
540 R=A-C*B
550 IF R=Ø THEN 58Ø
560 PRINT C;"  ";STR$(R);"/";STR$(B)
570 GOTO 59Ø
580 PRINT C
590 PRINT :::"PRESS 1 FOR ANOTHER PROBLEM"
600 PRINT TAB(7);"2 TO END PROGRAM";
610 CALL KEY(Ø,K,S)
620 IF K=49 THEN 11Ø
630 IF K<>5Ø THEN 61Ø
640 CALL CLEAR
650 END
```

Adding Fractions

This program has two main options, adding fractions with like denominators, such as $\frac{1}{12}+\frac{5}{12}+\frac{7}{12}$, or adding fractions with unlike denominators, such as $\frac{1}{2}+\frac{1}{3}+\frac{1}{4}$. The program will add up to nine fractions with like denominators or up to five fractions with unlike denominators, which is usually sufficient for fifth- and six-grade mathematics students.

If the option of like denominators is chosen, first press the total number of fractions to be added. Then enter the denominator, followed by the numerators.

If the option of unlike denominators is chosen, press from two to five for the number of fractions. The numerator and then the denominator are entered for each fraction.

The fractions are added, the problem is rewritten, and then the answer is printed in simplified terms.

Program 7-6. Adding Fractions

```
100 REM  ADDING FRACTIONS
110 CALL CLEAR
120 PRINT "** ADDING FRACTIONS **"
130 PRINT :::"CHOOSE:"
140 PRINT :"1  LIKE DENOMINATORS"
150 PRINT :"2  UNLIKE DENOMINATORS":
    ::
160 CALL KEY(0,K,S)
170 IF K=50 THEN 380
180 IF K<>49 THEN 160
190 CALL CLEAR
200 CH=1
210 PRINT "ADDING FRACTIONS WITH"
220 PRINT "LIKE DENOMINATORS"
230 PRINT ::"HOW MANY FRACTIONS?"
240 CALL KEY(0,K,S)
250 IF (K<50)+(K>57)THEN 240
260 CALL HCHAR(23,23,K)
270 C=K-48
280 PRINT ::"WHAT IS THE DENOMINATOR
    ?"
290 INPUT DT
300 PRINT ::"ENTER THE NUMERATORS"::
```

```
310 NT=Ø
320 FOR I=1 TO C
330 INPUT N(I)
340 NT=NT+N(I)
350 D(I)=DT
360 NEXT I
370 GOTO 69Ø
380 CALL CLEAR
390 PRINT "ADDING UP TO FIVE"
400 PRINT "FRACTIONS WHICH MAY HAVE"
410 PRINT "UNLIKE DENOMINATORS"
420 PRINT ::"HOW MANY FRACTIONS?"::
430 CALL KEY(Ø,K,S)
440 IF (K<5Ø)+(K>53)THEN 43Ø
450 CALL HCHAR(22,23,K)
460 C=K-48
470 NT=Ø
480 DT=1
490 FOR I=1 TO C
500 PRINT "FRACTION";I
510 INPUT "{3 SPACES}NUMERATOR =
    {3 SPACES}":N(I)
520 INPUT "{3 SPACES}DENOMINATOR = "
    :D(I)
530 IF D(I)<>Ø THEN 56Ø
540 PRINT :"DENOMINATOR CANNOT BE ZER
    O"::
550 GOTO 52Ø
560 IF I=1 THEN 6ØØ
570 FOR J=1 TO I-1
580 IF D(I)=D(J)THEN 62Ø
590 NEXT J
600 F=D(I)
610 GOTO 63Ø
620 F=1
630 DT=DT*F
640 NEXT I
650 FOR I=1 TO C
660 F=DT/D(I)
670 NT=NT+N(I)*F
680 NEXT I
690 CALL CLEAR
```

```
700 PRINT "** ADDING FRACTIONS **"::
    :
710 FOR I=1 TO C
720 PRINT STR$(N(I));"/";STR$(D(I))
730 NEXT I
740 PRINT "---------------"::
750 IF DT>NT THEN 780
760 L=DT
770 GOTO 790
780 L=NT
790 ST=-2
800 IF DT/2<>INT(DT/2)THEN 820
810 IF NT/2=INT(NT/2)THEN 830
820 ST=-1
830 FOR I=L TO 2 STEP ST
840 A=NT/I
850 IF A<>INT(A)THEN 880
860 B=DT/I
870 IF B=INT(B)THEN 910
880 NEXT I
890 A=NT
900 B=DT
910 PRINT STR$(A);"/";STR$(B)
920 IF A<B THEN 990
930 W=INT(A/B)
940 R=A-W*B
950 IF R<>0 THEN 980
960 PRINT "OR";W
970 GOTO 990
980 PRINT :"OR ";W;"  ";STR$(R);"/";
    STR$(B)
990 PRINT ::"PRESS 1 FOR ANOTHER PRO
    BLEM"
1000 PRINT TAB(7);"2 TO END PROGRAM"
1010 CALL KEY(0,K,S)
1020 IF K=49 THEN 110
1030 IF K<>50 THEN 1010
1040 CALL CLEAR
1050 END
```

Solving Simultaneous Equations

This program presents a basic algorithm for solving up to nine simultaneous equations using the matrix inversion technique. A 9-by-9 system of equations, which may take hours to calculate by hand, can be solved in less than a minute with this program.

Table 7-1 shows a system of three equations with three unknowns.

Table 7-1. Three Simultaneous Equations

$$x_1 + x_2 + x_3 = 12$$
$$2x_1 + x_2 + 3x_3 = 25$$
$$x_1 + 3x_2 + 2x_3 = 25$$

In Matrix Form

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 3 \\ 1 & 3 & 2 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 12 \\ 25 \\ 25 \end{bmatrix}$$

coefficients unknowns constant vector

$$[A] \bullet [X] = [B]$$

Solution Vector

$$x_1 = 3$$
$$x_2 = 4$$
$$x_3 = 5$$

First, enter the degree of the matrix, or the number of equations and unknowns. Next, enter the coefficients row by row with the corresponding *B* elements. In Table 7-1, the value of *N* would be entered as 3 for three equations with three unknowns. In order, the following numbers are entered:

A(1,1) = 1
A(1,2) = 1
A(1,3) = 1
B(1) = 12

A(2,1) = 2
A(2,2) = 1
A(2,3) = 3
B(2) = 25

A(3,1) = 1
A(3,2) = 3
A(3,3) = 2
B(3) = 25

The solution vector is then printed.

Program 7-7. Solving Simultaneous Equations

```
100 REM  SIMULTANEOUS EQUATIONS
110 CALL CLEAR
120 PRINT "SOLVING SIMULTANEOUS"
130 PRINT "EQUATIONS BY THE"
140 PRINT "MATRIX INVERSION TECHNIQUE"
150 PRINT ::"SOLVE [A][X] = [B]"
160 PRINT :"ENTER DEGREE OF THE MATRIX"
170 PRINT "OR NUMBER OF EQUATIONS"::
180 INPUT "N = ":N
190 IF N<10 THEN 220
200 PRINT :"N MUST BE < 10"::
210 GOTO 180
220 IF N>1 THEN 250
230 PRINT :"1<N<10{3 SPACES}TRY AGAIN"::
240 GOTO 180
250 PRINT ::"THE COEFFICIENTS OF X"
260 PRINT "ARE THE 'A' MATRIX."
270 PRINT :"INPUT THE VALUES ROW BY ROW:"
280 PRINT :"A(1,1),A(1,2),A(1,3),..."
290 PRINT "A(2,1),A(2,2),A(2,3),..."
300 PRINT ".":".":"."
310 PRINT "A(N,1),A(N,2),...,A(N,N)":::
320 FOR I=1 TO N
330 FOR J=1 TO N
```

```
340 INPUT "A("&STR$(I)&","&STR$(J)&") = ":A
    (I,J)
350 W(I,J)=A(I,J)
360 NEXT J
370 PRINT
380 INPUT "B("&STR$(I)&") = ":B(I)
390 PRINT ::
400 NEXT I
410 PRINT ::"--SOLVING--"::
420 REM  INVERT MATRIX A
430 FOR C=1 TO N
440 IF W(C,C)<>0 THEN 460
450 GOSUB 710
460 W(C,C)=1/W(C,C)
470 FOR D=1 TO N
480 IF (D-C)=0 THEN 540
490 W(D,C)=W(D,C)*W(C,C)
500 FOR E=1 TO N
510 IF (E-C)=0 THEN 530
520 W(D,E)=W(D,E)-W(D,C)*W(C,E)
530 NEXT E
540 NEXT D
550 FOR E=1 TO N
560 IF (E-C)=0 THEN 580
570 W(C,E)=-W(C,C)*W(C,E)
580 NEXT E
590 NEXT C
600 PRINT ::"SOLUTION VECTOR X:"::
610 FOR I=1 TO N
620 X(I)=0
630 FOR J=1 TO N
640 X(I)=X(I)+W(I,J)*B(J)
650 NEXT J
660 PRINT :"X("&STR$(I)&") = ";X(I)
670 NEXT I
680 PRINT ::
690 GOTO 870
700 REM  SUB TO SWITCH ROWS
710 FOR F=C+1 TO N
720 IF W(F,C)=0 THEN 820
730 FOR E=1 TO N
740 DW=W(C,E)
```

```
750 W(C,E)=W(F,E)
760 W(F,E)=DW
770 NEXT E
780 DB=B(C)
790 B(C)=B(F)
800 B(F)=DB
810 GOTO 860
820 NEXT F
830 PRINT "SORRY, DETERMINANT=0."
840 PRINT "NO UNIQUE SOLUTION."
850 GOTO 870
860 RETURN
870 PRINT :"PRESS 1 FOR ANOTHER PROBLEM"
880 PRINT TAB(7);"2 TO END PROGRAM"
890 CALL KEY(0,K,S)
900 IF K=49 THEN 110
910 IF K<>50 THEN 890
920 CALL CLEAR
930 END
```

Earning Money

The idea for this program came from the type of problem found in high school mathematics competency tests. The program creates story problems. Twelve different names and six different jobs are READ in as DATA. For each problem the program picks a name at random, chooses the appropriate pronoun in the following statement, and picks a job at random for some wordings. All the numbers chosen are random within certain limits.

These problems are multiplication problems — an hourly wage times the number of hours, or an amount earned per week times a number of weeks.

Program 7-8. Math Competency: Earning Money

```
100 CALL CLEAR
110 PRINT TAB(6);"MATH COMPETENCY"
120 PRINT :::TAB(7);"EARNING MONEY"
130 PRINT :::::
140 DIM N$(5),J$(5),T$(5)
```

```
150 FOR I=Ø TO 5
160 READ N$(I),J$(I),T$(I)
170 NEXT I
180 DATA SAM,DOING ODD JOBS,JOHN,JOE,MOWING
    LAWNS,ANDY,BOB,TENDING CHILDREN,MARK
    ,ANN
190 DATA RUNNING ERRANDS,LENA,SUE,DOING HOU
    SEWORK,AURA,KAY,DELIVERING ADS,DAWN
200 GOTO 370
210 PRINT :TAB(15);"PRESS <ENTER>";
220 CALL KEY(Ø,K,S)
230 IF K<>13 THEN 220
240 RETURN
250 CALL SOUND(100,330,2)
260 CALL SOUND(150,262,2)
270 RETURN
280 CALL SOUND(100,262,2)
290 CALL SOUND(100,330,2)
300 CALL SOUND(100,392,2)
310 CALL SOUND(200,523,2)
320 RETURN
330 P=100+25*INT(RND*10)
340 P$=STR$(P)
350 P$="$"&SEG$(P$,1,LEN(P$)-2)&"."&SEG$(P$
    ,LEN(P$)-1,2)
360 RETURN
370 CALL CLEAR
380 RANDOMIZE
390 N=INT(RND*6)
400 H=8+INT(RND*11)
410 GOSUB 330
420 PRINT N$(N);" WORKS";H;"HOURS PER WEEK.
    "
430 IF N<3 THEN 460
440 PRINT :"SHE EARNS ";
450 GOTO 470
460 PRINT :"HE EARNS ";
470 PRINT P$;" PER HOUR."
480 IF N<3 THEN 510
490 PRINT :"HOW MUCH DOES SHE EARN"
500 GOTO 520
510 PRINT :"HOW MUCH DOES HE EARN"
```

```
520 PRINT :"IN A WEEK?"::
530 INPUT "$":D
540 D1=P*H/100
550 IF ABS(D-D1)>.001 THEN 610
560 GOSUB 280
570 PRINT ::"TRY AGAIN? (Y/N)"
580 CALL KEY(0,K,S)
590 IF K=89 THEN 370
600 IF K=78 THEN 680 ELSE 580
610 GOSUB 250
620 PRINT :"MULTIPLY";H;"HOURS BY ";P$::"PE
    R HOUR."
630 P=H*P
640 GOSUB 340
650 PRINT :"THE ANSWER IS ";P$
660 GOSUB 210
670 GOTO 370
680 CALL CLEAR
690 RANDOMIZE
700 N=INT(RND*6)
710 H=INT(RND*11)+8
720 GOSUB 330
730 PRINT N$(N);" EARNS ";P$;" PER HOUR."
740 IF N<3 THEN 770
750 PRINT :"SHE WORKS";
760 GOTO 780
770 PRINT :"HE WORKS";
780 PRINT H;"HOURS PER WEEK."
790 IF N<3 THEN 820
800 PRINT :"HOW MUCH WILL SHE EARN IN"
810 GOTO 830
820 PRINT :"HOW MUCH WILL HE EARN IN"
830 W=INT(RND*19)+2
840 PRINT :W;"WEEKS?"::
850 INPUT "$":D
860 D1=P*H*W/100
870 IF ABS(D-D1)>.001 THEN 930
880 GOSUB 280
890 PRINT ::"TRY AGAIN? (Y/N)"
900 CALL KEY(0,K,S)
910 IF K=89 THEN 680
920 IF K=78 THEN 1030 ELSE 900
```

```
930 GOSUB 250
940 PRINT :"MULTIPLY";H;"HOURS BY"
950 PRINT :P$;" PER HOUR."
960 PRINT :"THEN MULTIPLY BY";W;"WEEKS."
970 PRINT :"THE ANSWER IS ";
980 P=H*P*W
990 GOSUB 340
1000 PRINT P$:::
1010 GOSUB 210
1020 GOTO 680
1030 CALL CLEAR
1040 J=INT(RND*6)
1050 T=INT(RND*6)
1060 GOSUB 330
1070 W=INT(RND*8)+2
1080 PRINT T$(T);" EARNED ";P$;" LAST WEEK"
1090 PRINT :J$(J);"."
1100 IF T<3 THEN 1130
1110 PRINT :"IF SHE EARNED THIS AMOUNT"
1120 GOTO 1140
1130 PRINT :"IF HE EARNED THIS AMOUNT"
1140 PRINT :"EVERY WEEK, WHAT WOULD THE"
1150 PRINT :"TOTAL INCOME BE FOR"
1160 PRINT :W;"WEEKS?"::
1170 INPUT "$":D
1180 D1=P*W/100
1190 IF ABS(D-D1)>.001 THEN 1250
1200 GOSUB 280
1210 PRINT ::"TRY AGAIN? (Y/N)";
1220 CALL KEY(0,K,S)
1230 IF K=89 THEN 1030
1240 IF K=78 THEN 1330 ELSE 1220
1250 GOSUB 250
1260 PRINT :"MULTIPLY ";P$;" PER WEEK"
1270 PRINT :"BY";W;"WEEKS."
1280 P=P*W
1290 GOSUB 340
1300 PRINT :"THE ANSWER IS ";P$::
1310 GOSUB 210
1320 GOTO 1030
1330 CALL CLEAR
1340 END
```

Buying Items

In this math competency program, a list of items is printed with their costs, which are random numbers within certain limits. One question is how much it would cost to buy everything on the list. The second question, in multiple-choice form, is which items could be purchased by a person who has a certain amount of money.

The DATA statements consist of names and items with a minimum and maximum cost. The subroutine in lines 460-540 converts the number to a string so that items may be printed properly in dollars and cents. Lines 1160-1300 randomly choose the multiple choice items and place the correct answer as one of the choices.

Program 7-9. Math Competency: Buying Items

```
100 CALL CLEAR
110 PRINT TAB(6);"MATH COMPETENCY"
120 CALL CHAR(136,"080402FF020408")
130 PRINT :::TAB(7);"BUYING ITEMS"
140 CALL COLOR(14,9,16)
150 PRINT :::::
160 DIM I$(3,5),I(3,5,2),N$(6),J(5),H$(3),S
    $(4)
170 FOR C=1 TO 6
180 READ N$(C)
190 NEXT C
200 FOR A=1 TO 3
210 FOR C=1 TO 5
220 READ I$(A,C),I(A,C,1),I(A,C,2)
230 NEXT C
240 NEXT A
250 DATA ANGIE,CINDY,CHERY,RICKY,BOBBY,RAND
    Y,PENCIL,8,15
260 DATA ERASER,2,10,NOTEBOOK,35,99,RULER,2
    9,49
270 DATA PAPER,59,90,DOLL,249,599,BALL,49,8
    9,TRUCK,100,150
280 DATA GAME,270,500,MODEL,300,700,CANDY,2
    0,50
290 DATA MEAT,123,425,FRUIT,24,50,CHIPS,100
    ,257,BREAD,100,179
```

```
300 H$(1)="PENCIL AND ERASER"
310 H$(2)="BALL AND TRUCK"
320 H$(3)="CANDY AND FRUIT"
330 GOTO 550
340 PRINT TAB(15);"PRESS <ENTER>";
350 CALL KEY(0,K,S)
360 IF K<>13 THEN 350
370 RETURN
380 CALL SOUND(100,330,2)
390 CALL SOUND(150,262,2)
400 RETURN
410 CALL SOUND(100,262,2)
420 CALL SOUND(100,330,2)
430 CALL SOUND(100,392,2)
440 CALL SOUND(200,523,2)
450 RETURN
460 P$=STR$(P)
470 IF LEN(P$)>1 THEN 490
480 P$="0"&P$
490 IF LEN(P$)>2 THEN 510
500 P$=" "&P$
510 PR$=SEG$(P$,LEN(P$)-1,2)
520 PL$=SEG$(P$,1,LEN(P$)-2)
530 P$="$"&PL$&"."&PR$
540 RETURN
550 RANDOMIZE
560 A=INT(RND*3+1)
570 TP=0
580 CALL CLEAR
590 PRINT "GIVEN THIS PRICE LIST:"::
600 FOR C=1 TO 5
610 D=I(A,C,2)-I(A,C,1)
620 P=I(A,C,1)+INT(RND*D+1)
630 GOSUB 460
640 TP=TP+P
650 PRINT TAB(6);I$(A,C);TAB(15);P$
660 NEXT C
670 R=INT(RND*13+4)
680 CALL COLOR(13,R,R)
690 CALL HCHAR(18,6,128,18)
700 CALL VCHAR(19,6,128,5)
710 CALL VCHAR(19,23,128,5)
```

```
720 CALL HCHAR(24,6,128,18)
730 F=INT(RND*2+1)
740 IF F=2 THEN 790
750 PRINT ::"HOW MUCH WILL IT COST"
760 PRINT "TO BUY ALL THE ITEMS"
770 PRINT "ON THE LIST?"
780 GOTO 830
790 N=INT(RND*6+1)
800 PRINT ::N$(N);" WANTS TO BUY"
810 PRINT "EVERYTHING ON THE LIST."
820 PRINT "WHAT WILL THE TOTAL COST BE?"
830 INPUT "$":X
840 IF ABS(X-TP/100)<.001 THEN 920
850 GOSUB 380
860 PRINT :"ADD ALL FIVE NUMBERS."
870 P=TP
880 GOSUB 460
890 PRINT "THE TOTAL IS ";P$:::
900 GOSUB 340
910 GOTO 550
920 GOSUB 410
930 CALL HCHAR(20,1,32,128)
940 IF F=2 THEN 970
950 PRINT "IF YOU COULD ONLY SPEND"
960 GOTO 980
970 PRINT "IF ";N$(N);" COULD ONLY SPEND"
980 IF A<>1 THEN 1010
990 M=INT(RND*5+25)
1000 GOTO 1050
1010 IF A<>2 THEN 1040
1020 M=INT(RND*36+239)
1030 GOTO 1050
1040 M=INT(RND*18+100)
1050 P=M
1060 GOSUB 460
1070 PRINT P$;", WHICH OF THESE PAIRS"
1080 PRINT "OF ITEMS COULD ";
1090 IF F<>1 THEN 1120
1100 PRINT "YOU BUY?"::
1110 GOTO 1160
1120 IF N>3 THEN 1150
1130 PRINT "SHE BUY?"::
```

```
1140 GOTO 1160
1150 PRINT "HE BUY?"::
1160 R=INT(RND*4+1)
1170 FOR V=1 TO 4
1180 IF V=R THEN 1280
1190 X=INT(RND*2+4)
1200 S$(V)=I$(A,X)
1210 X=INT(RND*3+1)
1220 S$(V)=S$(V)&" AND "&I$(A,X)
1230 IF V=1 THEN 1290
1240 FOR V1=1 TO V-1
1250 IF S$(V1)=S$(V)THEN 1190
1260 NEXT V1
1270 GOTO 1290
1280 S$(V)=H$(A)
1290 PRINT TAB(3);CHR$(64+V);" "&S$(V)
1300 NEXT V
1310 CALL SOUND(150,1397,2)
1320 CALL KEY(0,K,S)
1330 IF (K<65)+(K>68)THEN 1320
1340 CALL HCHAR(K-45,4,42)
1350 IF K<>64+R THEN 1410
1360 GOSUB 410
1370 PRINT :"TRY AGAIN? (Y/N)";
1380 CALL KEY(0,K,S)
1390 IF K=89 THEN 550
1400 IF K=78 THEN 1450 ELSE 1380
1410 GOSUB 380
1420 CALL HCHAR(19+R,3,136)
1430 PRINT :"THE TOTAL OF THE TWO ITEMS  MU
     ST BE LESS THAN ";P$
1440 GOTO 1370
1450 CALL CLEAR
1460 END
```

Musical Bugle

This is a typing drill for someone who has already learned the correct fingering for all the letters on the keyboard. This drill makes it fun to practice typing. As random letters appear in the bugle, type the letters. The faster the letters are pressed, the faster the music goes — and it should be a familiar tune if the letters are typed correctly and fast enough. The letters to be typed are chosen randomly in lines 640-650, and the notes are played by READing the frequencies from DATA statements in lines 680-690.

Program 7-10. Typing Drill: Musical Bugle

```
110 REM MUSICAL BUGLE
120 CALL CLEAR
130 FOR C=2 TO 8
140 CALL COLOR(C,2,12)
150 NEXT C
160 CALL CHAR(95,"0")
170 RESTORE 180
180 DATA FEF8F0E0C0808,7F1F0F0703010l
190 DATA 0001010307 0F1F7F,008080C0E0F0F8FE
200 DATA FFFEFEFCF8F0E08,FF7F7F3F1F0F0701
210 DATA 0103070F1F3F7FFF,FF7F3F1F0F070301
220 DATA 0707070707070707,000000FC78787878
230 CALL COLOR(12,12,1)
240 CALL COLOR(13,12,1)
250 FOR C=123 TO 132
260 READ C$
270 CALL CHAR(C,C$)
280 NEXT C
290 CALL HCHAR(23,3,95,28)
300 PRINT "_TYPE_THE_LETTERS_AS_THEY___"
310 PRINT "_APPEAR_IN_THE_HORN.________"
320 PRINT "_IF_YOU_TYPE_EVENLY_YOU_____"
330 PRINT "_SHOULD_RECOGNIZE_THE_TUNE._"
340 CALL HCHAR(24,3,95,28)
350 PRINT ::::::::::::::
360 CALL HCHAR(17,7,131)
370 CALL HCHAR(17,8,95,19)
380 CALL HCHAR(16,26,125)
```

```
390 CALL HCHAR(18,26,124)
400 CALL HCHAR(15,27,129)
410 CALL VCHAR(16,27,95,3)
420 CALL HCHAR(19,27,130)
430 CALL HCHAR(14,28,129)
440 CALL VCHAR(15,28,95,5)
450 CALL HCHAR(20,28,130)
460 CALL HCHAR(18,11,123)
470 CALL VCHAR(18,10,95,3)
480 CALL HCHAR(21,10,128)
490 CALL HCHAR(20,11,126)
500 CALL HCHAR(21,11,95,11)
510 CALL HCHAR(21,22,127)
520 CALL HCHAR(20,21,125)
530 CALL VCHAR(18,22,95,3)
540 CALL HCHAR(18,21,124)
550 CALL HCHAR(16,15,132,3)
560 RESTORE 570
570 DATA 466,370,415,370,311,370,277
580 DATA 370,466,370,415,370,311,370
590 DATA 277,370,466,370,415,370,311
600 DATA 370,277,370,233,370,208,370
610 CALL SOUND(150,1397,4)
620 FOR C=1 TO 28
630 RANDOMIZE
640 L=INT(RND*26+65)
650 CALL HCHAR(19,16,L)
660 CALL KEY(0,K,S)
670 IF K<>L THEN 660
680 READ N
690 CALL SOUND(-4250,N,1)
700 CALL HCHAR(19,16,95)
710 NEXT C
720 FOR C=1 TO 3
730 L=INT(RND*26+65)
740 CALL HCHAR(19,16,L)
750 CALL KEY(0,K,S)
760 IF K<>L THEN 750
770 CALL SOUND(-300,185,1)
780 CALL HCHAR(19,16,95)
790 NEXT C
800 CALL HCHAR(19,16,32)
```

```
810 FOR C=1 TO 16
820 CALL HCHAR(24,2+C,ASC(SEG$("TRY_AGAIN?_
    (Y/N)",C,1)))
830 NEXT C
840 CALL KEY(0,K,S)
850 IF K=78 THEN 890
860 IF K<>89 THEN 840
870 CALL HCHAR(24,3,32,16)
880 GOTO 560
890 CALL CLEAR
900 END
```

Type Invaders

Here is another typing drill for a student who has learned where all the letters are, but just needs to practice typing faster. This program is a game, like many invader-type games. A letter appears in the sky and starts descending. The sooner you type it, the higher your score will be. If the letter blinks and comes down ten steps without being typed correctly, the score is decreased by five.

At random times a spaceship with a three-letter word appears. If the word is typed correctly, there is a bonus of ten points.

The running score is shown at the bottom of the screen. After ten spaceships the game is over, and the final score and high score are shown.

Program 7-11. Typing Drill: Type Invaders

```
110 REM TYPE INVADERS
120 CALL CLEAR
130 DIM SS$(14)
140 PRINT "ALIEN LETTERS WILL APPEAR"
150 PRINT :"FROM SPACE.  YOU NEED TO"
160 PRINT :"PREVENT THEM FROM ENTERING"
170 PRINT :"EARTH'S ATMOSPHERE BY"
180 PRINT :"TYPING THE LETTER AS SOON"
190 PRINT :"AS POSSIBLE."
200 PRINT ::"ONCE IN A WHILE A SPACESHIP"
210 PRINT :"WILL APPEAR.  TYPE THE WORD"
```

```
220 PRINT :"CORRECTLY FOR 10 POINTS."
230 CALL CHAR(123,"101054565EDEFEFF")
240 CALL COLOR(12,3,1)
250 CALL CHAR(128,"1122448811224488")
260 CALL COLOR(13,12,5)
270 CALL CHAR(136,"081C7F1C3C6642")
280 DATA 030F3F3F3F3F0F03,0000000030F3FFF,
    000000FFFFFFFFFF,00000000C0F0FCFF
290 DATA C0F0FCFCFCFCF0C,FFFCF0C,FFFFFFFFFF
    ,FF3F0F03,FFFFFFFFFFFFFFFF
300 FOR C=144 TO 152
310 READ C$
320 CALL CHAR(C,C$)
330 NEXT C
340 FOR C=0 TO 14
350 READ SS$(C)
360 NEXT C
370 DATA THE,HIS,ITS,SHE,HER,AND,OUR,FEW,RU
    N,TIE,RED,TWO,YOU,ONE,TEN
380 CALL COLOR(15,16,1)
390 CALL COLOR(16,16,1)
400 PRINT :::"PRESS ENTER TO START.";
410 CALL KEY(0,K,S)
420 IF K<>13 THEN 410
430 CALL CLEAR
440 CALL SCREEN(14)
450 CALL HCHAR(22,1,123,32)
460 FOR C=2 TO 8
470 CALL COLOR(C,2,16)
480 NEXT C
490 CALL HCHAR(19,15,128,3)
500 CALL HCHAR(20,14,128,5)
510 CALL HCHAR(21,13,128,7)
520 PRINT "SCORE: ";SC;
530 FOR A=1 TO 10
540 FOR A1=1 TO INT(8*RND+1)
550 RANDOMIZE
560 L=INT(RND*26)+65
570 DX=1-INT(3*RND)
580 ROW=2
590 T=10
600 COL=INT(9*RND)+11
```

```
610 CALL HCHAR(ROW,COL,L)
620 CALL SOUND(150,1397,4)
630 CALL KEY(0,K,S)
640 IF K=L THEN 730
650 T=T-1
660 CALL HCHAR(ROW,COL,32)
670 ROW=ROW+1
680 COL=COL+DX
690 IF T<>0 THEN 610
700 CALL SOUND(1000,-4,2)
710 SC=SC-5
720 GOTO 850
730 CALL SOUND(1000,-7,2)
740 CALL SCREEN(16)
750 CALL HCHAR(20,16,K)
760 CALL SCREEN(10)
770 CALL HCHAR(ROW,COL,136)
780 CALL SCREEN(12)
790 CALL SCREEN(14)
800 FOR C=1 TO 10
810 CALL COLOR(14,16,7)
820 CALL COLOR(14,7,16)
830 NEXT C
840 SC=SC+T
850 CALL HCHAR(ROW,COL,32)
860 GOSUB 1370
870 CALL HCHAR(20,16,128)
880 NEXT A1
890 R=INT(6*RND)+2
900 CL=INT(22*RND)+3
910 CALL HCHAR(R,CL,152,3)
920 CALL HCHAR(R,CL-1,144)
930 CALL HCHAR(R-1,CL,145)
940 CALL HCHAR(R-1,CL+1,146)
950 CALL HCHAR(R-1,CL+2,147)
960 CALL HCHAR(R,CL+3,148)
970 CALL HCHAR(R+1,CL+2,149)
980 CALL HCHAR(R+1,CL+1,150)
990 CALL HCHAR(R+1,CL,151)
1000 RANDOMIZE
1010 W=INT(15*RND)
1020 W$=SS$(W)
```

```
1030 FOR C=1 TO 3
1040 CALL HCHAR(R,CL-1+C,ASC(SEG$(W$,C,1)))
1050 NEXT C
1060 CALL SOUND(200,-1,2)
1070 FOR C=1 TO 3
1080 CALL KEY(0,K,S)
1090 IF S<1 THEN 1080
1100 CALL HCHAR(R+3,CL-1+C,K)
1110 B(C)=K
1120 NEXT C
1130 CALL SOUND(100,880,2)
1140 CALL SCREEN(12)
1150 FOR C=1 TO 3
1160 B$=B$&CHR$(B(C))
1170 NEXT C
1180 CALL SCREEN(14)
1190 IF B$=W$ THEN 1290
1200 CALL SOUND(100,392,2)
1210 CALL SOUND(100,262,2)
1220 B$=""
1230 GOSUB 1370
1240 FOR C=R-1 TO R+3
1250 CALL HCHAR(C,CL-1,32,5)
1260 NEXT C
1270 NEXT A
1280 GOTO 1420
1290 CALL SOUND(1000,-7,2)
1300 CALL HCHAR(R,CL,136,3)
1310 FOR C=1 TO 10
1320 CALL COLOR(14,16,7)
1330 CALL COLOR(14,7,16)
1340 NEXT C
1350 SC=SC+10
1360 GOTO 1220
1370 SC$=STR$(SC)&"  "
1380 FOR C=1 TO LEN(SC$)
1390 CALL HCHAR(24,10+C,ASC(SEG$(SC$,C,1)))
1400 NEXT C
1410 RETURN
1420 CALL CLEAR
1430 CALL SCREEN(8)
1440 FOR C=2 TO 8
```

```
1450 CALL COLOR(C,2,1)
1460 NEXT C
1470 PRINT "YOUR SCORE: ";SC
1480 IF HS>SC THEN 1500
1490 HS=SC
1500 PRINT :::"HIGH SCORE: ";HS
1510 PRINT ::::"TRY AGAIN? (Y/N)"::::
1520 SC=0
1530 CALL KEY(0,K,S)
1540 IF K=89 THEN 430
1550 IF K<>78 THEN 1530
1560 END
```

Car Cost Comparison

This program is an example of how any financial decision might be made easier with the computer. The program makes a cost comparison between two cars. First, enter the EPA comparative mileage for each car, such as 17 mpg and 26 mpg. Next, enter the cost of gasoline, between .50 and 2.00 per gallon. Finally, enter the approximate number of miles driven per year, such as 15000. The screen clears and the information is itemized for the two cars, with the total annual cost difference. You may try again with a different cost for gas, perhaps, or change the miles driven — and practically instantly you can analyze the results.

Program 7-12. Car Cost Comparison

```
100 REM  CAR COST COMPARISON
110 CALL CHAR(101,"070E1E3FFFFF0606")
120 CALL CHAR(102,"FF0808FFFFFF")
130 CALL CHAR(103,"C06060FFFFFF1818")
140 CALL CHAR(104,"070E1E3FFFFF0606")
150 CALL CHAR(105,"FF0808FFFFFF")
160 CALL CHAR(106,"C06060FFFFFF1818")
170 CALL COLOR(9,7,1)
180 CALL COLOR(10,6,1)
190 CALL CLEAR
200 PRINT "COMPARISON OF TWO CARS"::::
210 INPUT "CAR A--GAS MILEAGE, MPG: ":A
```

```
220 IF (A>=1)+(A<=50)=-2 THEN 250
230 PRINT :"SORRY, 1<MPG<50"::
240 GOTO 210
250 PRINT
260 INPUT "CAR B--GAS MILEAGE, MPG: ":B
270 IF (B>=1)+(B<=50)=-2 THEN 300
280 PRINT :"SORRY, 1<MPG<50"::
290 GOTO 260
300 PRINT ::"ENTER COST OF GAS IN DOLLARS"
310 PRINT :"(SUCH AS 1.18)"::
320 INPUT "GAS PRICE = $":C
330 IF (C>=.5)+(C<=2)=-2 THEN 370
340 PRINT :"GAS PRICE SHOULD BE BETWEEN"
350 PRINT :".50 AND 2.00"::
360 GOTO 320
370 PRINT ::"HOW MANY MILES DO YOU DRIVE"::
380 INPUT "PER YEAR? ":M
390 IF (M>0)+(M<100000)=-2 THEN 420
400 PRINT :"ASSUME 0<MILES<100000"::
410 GOTO 370
420 CALL CLEAR
430 PRINT "GAS PRICE: $";C
440 PRINT :"ANALYSIS IS FOR"
450 PRINT M;"MILES PER YEAR."
460 PRINT ::TAB(5);"hij";TAB(19);"efg"
470 PRINT :"{3 SPACES}";A;"MPG","
    {3 SPACES}";B;"MPG"
480 AI=(INT(100*(M*C/A+.005)))/100
490 BI=(INT(100*(M*C/B+.005)))/100
500 PRINT ::"COST FOR GAS:"
510 PRINT :"{3 SPACES}$";AI,"{3 SPACES}$";B
    I
520 PRINT :::"COST DIFFERENCE = $";ABS(AI-B
    I)
530 PRINT :::"TRY AGAIN? (Y/N)";
540 CALL KEY(0,K,S)
550 IF K=89 THEN 190
560 IF K<>78 THEN 540
570 CALL CLEAR
580 END
```

Characters: Code Numbers and Sets

Code #	*Character*
Set #1	
32	(space)
33	!
34	''
35	#
36	$
37	%
38	&
39	'
Set #2	
40	(
41	)
42	*
43	+
44	,
45	–
46	.
47	/
Set #3	
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
Set #4	
56	8
57	9
58	:
59	;
60	<
61	=
62	>
63	?

Code #	*Character*
Set #5	
64	@
65	A
66	B
67	C
68	D
69	E
70	F
71	G
Set #6	
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
Set #7	
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
Set #8	
88	X
89	Y
90	Z
91	[
92	\
93	]
94	^
95	—

Code #	*Character*	*Code #*
Set #9		**Set #13***
96	`	128
97	A	129
98	B	130
99	C	131
100	D	132
101	E	133
102	F	134
103	G	135
Set #10		**Set #14**
104	H	136
105	I	137
106	J	138
107	K	139
108	L	140
109	M	141
110	N	142
111	O	143
Set #11		**Set #15**
112	P	144
113	Q	145
114	R	146
115	S	147
116	T	148
117	U	149
118	V	150
119	W	151
Set #12		**Set #16**
120	X	152
121	Y	153
122	Z	154
123	{	155
124	|	156
125	}	157
126	~	158
127	DEL	159

*There are no standard characters for sets 13 through 16. This has no effect on your ability to define them and use them in CALL HCHAR and CALL VCHAR statements, but it is very difficult to use them in PRINT statements.

Index

A

B

C

E

F

G

O

P

Q

R

COMPUTE! Books

P.O. Box 5406 Greensboro, NC 27403

Ask your retailer for these **COMPUTE! Books**. If he or she has sold out, order directly from **COMPUTE!**

For Fastest Service
Call Our **TOLL FREE US Order Line**

800-334-0868
In NC call 919-275-9809

Quantity	Title	Price	Total
______	The Beginner's Guide to Buying A Personal Computer	$ 3.95**	______
______	COMPUTE!'s First Book of Atari	$12.95*	______
______	Inside Atari DOS	$19.95*	______
______	COMPUTE!'s First Book of PET/CBM	$12.95*	______
______	Programming the PET/CBM	$24.95***	______
______	Every Kid's First Book of Robots and Computers	$ 4.95**	______
______	COMPUTE!'s Second Book of Atari	$12.95*	______
______	COMPUTE!'s First Book of VIC	$12.95*	______
______	COMPUTE!'s First Book of Atari Graphics	$12.95*	______
______	Mapping the Atari	$14.95*	______
______	Home Energy Applications On Your Personal Computer	$14.95*	______
______	Machine Language for Beginners	$12.95*	______

* Add $2 shipping and handling. Outside US add $4 air mail; $2 surface mail.

** Add $1 shipping and handling. Outside US add $4 air mail; $2 surface mail.

*** Add $3 shipping and handling. Outside US add $9 air mail; $3 surface mail.

Please add shipping and handling for each book ordered. ______

Total enclosed or to be charged. ______

All orders must be prepaid (money order, check, or charge). All payments must be in US funds. NC residents add 4% sales tax.

☐ Payment enclosed Please charge my: ☐ VISA ☐ MasterCard
☐ American Express Acc't. No. ______ Expires ___/___

Name ______

Address ______

City ______ State ______ Zip ______

Country ______

Allow 4-5 weeks for delivery.

12-4

A Teaching Book
A Reference Guide

C. Regena, *COMPUTE!* Magazine's columnist on the TI home computer, knows how it feels to be a beginner, trying to figure out how to make a new computer do all the things it can do. Yet she also knows how to write programs that make the most of every feature of the TI-99/4A.

The result is simple, clear explanations, along with dozens and dozens of examples that you can type in and try. The programs range from a few lines, just to show you how to use a particular statement or function, to full-fledged programs that can teach you to touch-type, correct your homework, solve complex engineering problems, or maintain a birthday list.

As you read this book, trying out the example programs as you go, you'll discover many programming ideas and skills. And while you're working on those new ideas, you can refer back to these pages again and again to see how C. Regena solved the problem you're facing.

ISBN 0-942386-12-4

$14.95