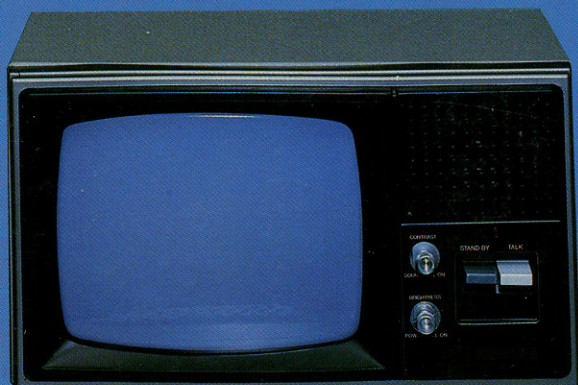


THE LAST WHOLE TI 99/4A BOOK

PROGRAMS AND POSSIBILITIES



PAUL GARRISON

The Last Whole TI-99/4A Book

Programs and Possibilities

The Last Whole TI-99/4A Book

Programs and Possibilities

Paul Garrison

**A Wiley Press Book
Wiley & Sons, Inc.**

New York • Chichester • Brisbane • Toronto • Singapore

Publisher: Judy V. Wilson

Editor: Theron Shreve

Managing Editor: Katherine Schowalter

Electronic Book Publishing Services: The Publisher's Network, Morrisville, PA

Copyright© 1984 by John Wiley & Sons, Inc.

All rights reserved. Published simultaneously in Canada.

Reproduction or translation of any part of this work beyond that permitted by Section 107 or 108 of the 1976 United States Copyright Act without the permission of the copyright owner is unlawful. Requests for permission or further information should be addressed to the Permissions Department, John Wiley & Sons, Inc.

Library of Congress Cataloging in Publication Data

Garrison, Paul.

The Last Whole TI-99/4A Book.

Includes index.

1. TI 99/4A (Computer)--Programming. I. Title.

QA76.8.T133G37 1984 001.64'2 83-26046

ISBN 0-471-87920-7

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

Acknowledgments

*I want to thank Texas Instruments
for providing me with the hardware and software
that made this book possible.*

Paul Garrison
Santa Fe, NM
September 20, 1983

Contents

Chapter 1 Introduction 1

What, Exactly, Is a Computer? 2

The Anatomy of the TI-99/4A Home Computer System 4

Chapter 2 Disks and Disk Drives 8

Initializing Your Disk 9

Recording Programs on Disk 14

Finding out What's on a Disk 15

Making Backup Disks 15

Rearranging Disk Contents 18

Chapter 3 Getting Started 20

Immediate Mode 20

Deferred Mode 30

Chapter 4 Commercial Software 38

Home Management/Personal Finance 39

Education 49

Home Entertainment 57

Chapter 5 Word Processing 60

Line Printers 61

Word Processing with TI-WRITER 66

Word Processing Without a Word Processor 82

Chapter 6 TI BASIC and TI EXTENDED BASIC 86

Commands and Statements 86

Subprograms 107

Chapter 7 A Primer to Personal Program Writing 125

A Simple Data-Base Program 126

A Size/Price Comparison Program 133

Standard Formulas 134

File Programs 137

Arrays 140

Chapter 8 Your Computer Talks Back 148

Chapter 9 Programs for the Home 154

Checkbook Program 155

Day of the Week Program 162

Travel Program 167

Name and Address List 176

Electronic Cookbook Program 184

Schedule C Tax Program 194

Kitchen Timer Program 206

Household Budget Program 209

Chapter 10 Educational Programs 214

Simple Arithmetic Program 214

Words That Make Other Words 221

The Presidents of the United States 227

Speed Program for Mathematics or Grammar 236

Arithmetic with Voice 244

Scrambled Words 249

History Lesson 254

Metric Conversions 258

Authors and Their Works 268

Object Recognition 273

Children's Counting game 273

Chapter 11 Sprites and Other Graphics 280

CALL HCHAR 280
CALL VCHAR 283
CALL CHAR 285
CALL COLOR and CALL SCREEN 288
CALL SPRITE 295
CALL MAGNIFY 297
CALL MOTION 298
CALL DELSPRITE 300
CALL PATTERN 301
CALL LOCATE 302
CALL COLOR and Sprites 302
CALL DISTANCE 303
CALL COINC 304

Chapter 12 Programs Strictly for Fun 307

A Dice Game 307
Model Railroading Program 315
A Numbers Game 325
A Game with Airplanes 328

Chapter 13 Programs for Business or Profession 336

Loan, Saving, and Investment Program 336
Currency Conversion Program 346
Invoice Writer 349
A Ledger Sheet Program 354
Analyzing Advertising Cost Versus Return 360
Analyzing Direct Mail Advertising 367
Introducing a New Product 370
Business Profit/Loss Analysis 376

Chapter 14 Advanced TI BASIC Statements 381

Defined Functions 381
Numerology 390

Chapter 15 Having Fun with TI LOGO II 397

- Turtle Graphics 397
- LOGO Procedures 401
- Turtle Commands 402
- Procedures with Inputs 405
- Procedures Within Procedures 408
- Sprite Graphics 409
- Musical LOGO 415
- Other TI LOGO Capabilities 418

Chapter 16 Telecommunication: Modems and Data Banks 420

- Getting "On Line" 421
- Services of The SOURCE 424

Appendix I ASCII Character Codes 427

Appendix II Speech Synthesizer Word List 430

Glossary of Computer Terms and Abbreviations 436

Computer Magazines 454

Index 457

1

Introduction

Revolution. It may not look as if we were living in the middle of a revolution, but we most certainly are. And it is the kind of revolution that will affect all of us, especially the young. I'm talking, of course, about the computer revolution, which is likely to make the industrial revolution look simple by comparison.

It's all the fault of Texas Instruments, the company that proudly proclaims itself to be the inventor of the integrated circuit, the microprocessor, and, in turn, the microcomputer. Without these incredible advances in the art of electronic microminiaturization, the personal computer—even the lowly pocket calculator—would not have been possible. But today they are not only possible, they are invading our homes, offices, and schools in ever-growing numbers, changing the way we keep records, make important decisions, and educate ourselves and our children.

This book is about home computers: the uses of personal computers in homes; in schools and other education situations; and in offices, where they are rapidly replacing the typewriter and the filing cabinet. Much of this book deals with writing personal programs, because program writing can be fun and exciting—not to mention that programs are really what computers are all about.

Today computers have become accepted tools to be used by business and professional people, much like any other office machine. But use in the home has, so far, been restricted largely to playing video games or performing very simple tasks. There are, however, a great many ways in which the computer can play a meaningful role in the home, and even more ways in homes with children from preschool to college age.

This book deals specifically with the Texas Instruments TI-99/4A Home Computer and with the optional peripherals that are designed to work with it. While this book was being prepared, Texas Instruments announced its decision to stop manufacturing and marketing the TI-99/4A computer itself, but the company has stated that it will continue to provide software and peripherals to the million-plus owners of TI-99/4As. This book is designed to increase the enjoyment the TI-99/4A owner can derive from using the computer.

WHAT, EXACTLY, IS A COMPUTER?

Many of us tend to think of computers as some sort of superintelligent being that threatens to take over our lives. Nonsense! A computer is simply one more dumb machine that can't do a thing without human help. Computer can't think, can't philosophize, can't reason. What they can do is remember huge amounts of information, store it indefinitely, and retrieve it when needed at incredible speed. There is a saying in computer country: "*Garbage in, garbage out.*" This means simply that if we, the human operators of the machine, provide it with faulty information, it will produce faulty answers simply because it doesn't know any better: It has no way of knowing that the information it was fed was faulty in the first place. So don't be afraid of computers. They are patient beasts of burden, ready to do your bidding as soon as you know how to talk to them.

That takes us back to the original question: What, exactly is a computer? *Webster's Unabridged Dictionary* defines a computer simply as "one that computes; an automatic electronic machine for performing calculations." But computers do a good deal more than calculate. To put it simply for the moment, computers are machines designed to deal with vast amounts of information, storing it when called upon to do so, or using it in order to perform a variety of functions.

The computer is the heart of what is generally referred to as a computer *system*. The computer itself consists of two types of memory, a central processing unit, and a keyboard. We use the keyboard to communicate with the computer. But computers don't understand the complex language we use every day. In order to

communicate with the computer, we have to learn a new language, one the computer has been programmed to understand. There are a number of such computer languages (often referred to as *high-level languages*), but the only one we need to be concerned with is called BASIC, for Beginners' All-purpose Symbolic Instruction Code. (Don't let that "beginners' " fool you—BASIC is a very powerful language, capable of doing anything we may expect of it.)

When I said that the computer understands BASIC, that is, of course, not exactly correct. Technically, a computer understands nothing other than certain combinations of zeros and ones. The home computers in use today are referred to as *digital* computers because they perform all of their tasks and functions using only two digital symbols: 0 and 1. All information typed into a computer is internally translated into still another language, *machine language*, that consists entirely of zeros and ones. Every letter, digit, or symbol consists of a combination of eight zeros and ones, where the individual zeros and ones are referred to as *bits* and the combination of eight such symbols is known as a *byte*. BASIC, like other high-level languages, allows us to communicate with the computer without getting bogged down in seemingly endless strings of ones and zeros.

There are a number of versions of BASIC for different computers, though the primary terms are identical in all BASIC dialects. We shall be concerned with the dialect known as TI BASIC, which consists of just under 100 words, abbreviations, and phrases that are relatively easy to learn and understand. Most of the time, in fact, you'll probably be using only about a dozen or so.

Information typed into a computer via a keyboard is stored in a memory known as the *random access memory* or RAM. The RAM is one of two kinds of memories contained in the computer. The other is known as the *read-only memory* or ROM. It contains all the built-in functions that the computer performs automatically when called upon to do so. The RAM and ROM interact with the *central processing unit* or CPU, which consists of *microprocessors*, those little black roach-like things with their multitudinous legs that do all the actual work.

The computer itself is only able to receive information and to process it. In order for us to know what's going on, we need a means for the computer to communicate with us. That means is the *monitor*, which may be an ordinary TV set, or one of the dedicated

black-and-white or color monitors designed specifically to function with computers. The monitor is used to display what we type in, and then whatever results are produced by the computer.

Now let's take a closer look at the TI-99/4A Home Computer system, the one we'll be discussing and using throughout this book.

THE ANATOMY OF THE TI-99/4A HOME COMPUTER SYSTEM

The emphasis here must be on the word *system*, because in order for a computer to be of use, it must interact with a number of different peripherals. The entire system, the computer and its peripheral devices, is known as the *hardware*. The system we'll be using throughout this book consists of the following:

1. The computer and its keyboard
2. A color monitor
3. A peripheral extension unit
4. A single disk drive
5. A line printer
6. An extended memory board
7. Joysticks
8. A speech synthesizer
9. A modem
10. A TI EXTENDED BASIC module

We have already talked a little about the computer itself, let's take a closer look at the TI-99/4A keyboard. This consists of 48 keys, representing the digits 0 through 9; the letters of the alphabet; the usual symbols such as periods, commas, and other punctuation marks; and arithmetic function symbols such as plus, minus, and so on. Unlike a conventional typewriter, certain symbols, such as quotation marks, require that two keys be pressed at the same time. If you're a touch typist, you may find that you have to do a bit of relearning.

In addition to the usual keys, there are three that are not found on an ordinary typewriter. One is labeled FCTN, which stands for function. It must always be pressed in conjunction with another

key in order to produce the desired result. It is used to type the symbols that are printed on the front side of 16 keys, and it is also used in combination with other keys to perform some special functions that we'll be talking about on and off. Another key is labeled CTRL for control, and it is used primarily in conjunction with a word processor. The third key is labeled ENTER. Its purpose is to enter typed information into the computer RAM, and, as we shall see, it is used with great frequency.

Computers are described as having memories of so many Ks. The memory being referred to is the RAM, and the number of Ks represents the number of bytes that can be stored in the RAM at one time. Although the symbol K normally stands for 1000, in computer terms it stands for 1024 bytes. If you think of each byte as one letter, and assume that the average word consists of seven letters, a 48K RAM can accommodate up to 7022 words before running out of memory space. That sounds like a lot, but it isn't, and, with that in mind, it is easy to understand why it is important to install the Extended Memory card, which increases the computer's basic 16K memory to 48K. There is always a lot of talk among computer enthusiasts about the importance of having plenty of Ks. Don't worry too much about it: The kinds of programs the average user is likely to write rarely exceeds 48K.

The monitor can be adjusted to produce different colors and color combinations, or to present a black-on-white display and, under certain circumstances, a white- or green-on-black display. The latter is important if the computer is to be used as a word processor, as it is less tiring on the eyes than white on black or black on white. Except when using the computer to produce graphic images, or when running commercial software programs, it is advisable to leave it in its black-and-white configuration because in that mode the display is sharper and easier to read than it is in the color mode.

The peripheral extension unit is just what the name implies. Its purpose is to control the peripheral devices, and there is no need for us to concern ourselves with exactly how that is accomplished.

Built into the peripheral extension unit are slots into which certain cards must be inserted, and also built in is one disk drive slot. A word of warning: If you're going to install the disk drive and the associated card yourself, you're likely to find, as I did, that it is humanly impossible to do so by following the instructions provided by Texas Instruments. To attach the cable that connects the card to

the drive in the manner prescribed simply doesn't work. I finally succeeded by removing some screws and partially removing the top of the unit. If you're purchasing your system from a dealer, let him or her do the installation.

Also installed in the peripheral extension unit is the Extended Memory card, which increases the size of the computer's RAM from 16 to 48K, and another card that controls the operation of the line printer.

The line printer is a sort of automatic typewriter. When commanded to do so, it will print the information being displayed or stored in the computer's RAM, printing both from right to left and from left to right. There are different types of line printers available, and we'll talk about printers in greater detail in Chapter 5.

TI offers joysticks as an option. Their purpose is related strictly to playing video games available from TI in the form of modules that can be plugged into the computer.

Another option is a speech synthesizer, a rather fascinating addition that can be used to cause the computer to speak the words that are typed in, assuming that the words are contained in the 373-word dictionary the synthesizer has been taught to pronounce, or that the words can be created by using combinations of those 373 words.

A modem is another option, one that makes it possible for your computer to communicate via telephone with other computers or with data banks, such as The SOURCE, which are repositories of huge amounts of data and information. The acronym *modem* stands for *modulate/demodulate*, because the computer signals are *modulated* into the type of signals that can be transmitted over telephone lines and then, on the other end, are *demodulated* back into the kind of signals the computer is equipped to deal with. We'll discuss the subject of telecommunication in Chapter 16.

The TI EXTENDED BASIC module adds a number of important features to standard TI BASIC. It may be very important to someone anticipating serious programming, but to the average user it is more a convenience item. One drawback is the fact that once it is loaded into the computer RAM, it takes up a fair number of Ks, thus reducing the amount of RAM available for program writing.

That, then, is the complete system, the hardware we'll be using in the various chapters of this book. In addition there is all manner of software, programs that have been prerecorded on disk or in the form of plug-in modules. In Chapter 4 we'll take a look at some software that is representative of the various categories in which it is available.

2

Disks and Disk Drives

Disks, also known as *floppy disks*, *diskettes*, or *floppies*, are the medium used by computers to store data for future use. Data may also be stored on magnetic tape via cassette recorders, but using cassettes is slow and inefficient (though considerably cheaper), and for the purpose of this book we shall assume that at least one disk drive is available with your TI-99/4A Home Computer.

The disks used with your TI-99/4A are 5.25 inches in diameter. They can be bought individually or in boxes of 10, priced anywhere from less than \$2 to more than \$5 each. Personally, I have been using the cheapest disks that I could find (at \$19.90 for a box of 10) and have found them to be entirely satisfactory.

Disks may be likened to phonograph records, though they function differently. They are made of flexible magnetic material, and the read/write head in the disk drive is used to record material on the disk as well as to read material from the disk. The disk surface is divided into sectors in order to permit the retrieval of information without having to read through all the preceding material first. (This is not true of cassettes. The tape recorder must read through all recorded material consecutively in order to find the requested information.)

When you buy a disk (or a box of disks), the disk itself is enclosed within two paper covers. The outer cover is removable. Whenever a disk is not in the drive, keep it in its outer envelope to protect the exposed surfaces from damage. The inner envelope is permanent, and

the disk must *never* be removed from it. This permanent envelope has four openings. In the center is a round opening used by the drive mechanism to spin the disk in the drive. Below it is an oval opening through which the disk itself can be seen. *Do not touch the surface of the disk, and keep disks away from sources of magnetism, as they can be inadvertantly erased.* There is another small round hole in both the envelope and the disk itself called index hole. It is used by the drive to work the disk's rotation. And on the right edge there is a rectangular cutout, referred to as the read/write slot. When that slot is open, you can write to the disk. If it is covered by a piece of tape or something, the disk becomes a read-only disk: You cannot write to that disk as long as the rectangular cutout is covered.

Unlike a phonograph record, which can simply be placed on the turntable and played, disks are incapable of reading or writing data until they have been *initialized*. This initialization or *formatting* process differs for different types of computers, which is why you cannot take a disk that you've initialized on your TI-99/4A to a friend who's got another type of computer and expect it to run. It won't.

INITIALIZING YOUR DISK

Let's begin by initializing a blank disk. First, of course, turn everything on. When turning on your computer, always turn on the expansion system (the box containing the interface cards and the disk drive) first, then use the slide switch on the keyboard to turn on the computer, and then turn on the monitor. Don't try to turn on the computer before the expansion system.

At this point you may want to use the controls below the screen to adjust the color and contrast. You can select bright colors, tints, or a black-and-white display.

With your computer and its peripherals turned on, plug the *Disk Manager* module into your computer console, place a fresh disk into the drive, facing to the right with the read/write slot at the top. With the Texas Instruments title page in display, press any key. The display will change to:

```
PRESS
1 FOR TI BASIC
2 FOR ``DISK MANAGER``
3 FOR ``DISKETTEN-MANAGER``
4 FOR ``GESTION DE DISQUES``
```

Unless you want the subsequent instructions to be presented in either German or French, ignore choices 3 and 4 and press 2. First the display will change to a fancy title page reading DISK MANAGER and after a moment it will present you with a new set of choices:

```
                DISK MANAGER
1 FILE COMMANDS
2 DISK COMMANDS
3 DISK TESTS
4 SET ALL COMMANDS FOR SINGLE DISK PROCESSING
YOUR CHOICE?  1
```

The 1 after YOUR CHOICE will be flashing on and off, indicating that the computer wants to know if that is your choice or if you'd rather select something else. Right now, select **4** and press **>ENTER<** because you want all commands to be set for single-disk processing. The display remains unchanged except for the bottom line, which now reads:

SINGLE DISK PROCESSING HAS BEEN INITIALIZED

Now select DISK COMMANDS by typing 2 and >ENTER<. The display will change to:

```
                DISK COMMANDS
1 CATALOG DISK
2 BACKUP DISK
3 MODIFY DISK NAME
4 INITIALIZE DISK
YOUR CHOICE? 1
```

For the moment we're interested only in the fourth choice, which you should select by typing 4 and pressing >ENTER<. The resultant display now reads:

```
INITIALIZE NEW DISK  
MASTER DISK (1-3)? 1  
DISK NOT INITIALIZED  
NEW DISK NAME?
```

The little square signs are to remind you that the largest number of characters you can use in naming a disk is 10. The TI disk system requires that each disk be given a unique name. Names can be as simple as DISKONE, DISKTWO, or as complicated as you wish—as long as each name consists of 10 characters or less. Type your chosen name in and press >ENTER<. The display now asks a series of questions, one at a time:

```
TRACKS PER SIDE? 40  
SINGLE SIDED (Y/N) Y  
SINGLE DENSITY (Y/N) Y
```

You can press >ENTER< in answer to each of these questions unless you're sure that your disk does not conform to those specifications. The display will then show:

INITIALIZE NEW DISK. . . PLEASE WAIT

There will be a pause while the computer and the disk drive do their thing. Eventually the display will change to:

DSK 1 - DISKNAME=DISKONE
AVAILABLE = 358 USED= 0
COMMAND COMPLETED
PRESS: PROC'D, REDO, BEGIN, OR BACK

DSK1 means that the disk is in drive number 1 (you can use up to three drives with your system). The numbers after AVAILABLE and USED refer to the number of sectors that are either available for use or have been used by previously recorded programs. PRESS: and the four choices refer to certain key combinations identified by the plastic strip that came with your computer. Type FCTN 5 followed by FCTN 9 by pressing the function (FCTN) key and the number key simultaneously. This will get you back to the original title page.

Your disk is now initialized and can be used to store programs, data, or information. Remove it from the drive and be sure to write the disk name on the label so that you always know which name you've used. Always close the disk drive when it's not in use.

RECORDING PROGRAMS ON DISK

You might want to skip the rest of this chapter until after you've actually keyed in your first program, which we'll do in the next chapter. Then, with the program still in the computer, follow these steps in order to record it on your newly initialized disk:

1. Place the disk in the drive as before.
2. Type `SAVE DSK1.DEGREES`.
3. Press `>ENTER<`.

where `DEGREES` can be any program name. I have chosen it because it is the first program you'll be writing in the next chapter. Once you've done that, remove the program `DEGREES` from the computer's memory (RAM) by typing `NEW >ENTER<`. The program is now no longer in your computer, but it is stored on the disk. Let's make sure:

1. Type `OLD DSK1.DEGREES`.
2. Press `>ENTER<`.

After a few moments of clacking a whirring, the red light on the disk drive will go out and the familiar prompt (`>`) and cursor (flashing square) will be displayed. (See page 21 if you're not sure what a prompt or a cursor is.) Now type `LIST >ENTER<` and your program will again be `LISTed` on the display, ready to be `RUN`.

FINDING OUT WHAT'S ON A DISK

Later on, when you've recorded a lot of material on one or more disks, you may want to check what is on the disk and, possibly more important, how much unused space is left on the disk. To do this we use the CATALOG option which we saw displayed earlier. Take the steps that were described above to call up the DISK MANAGER display and then type **1** to select CATALOG. The display will then ask where you want the CATALOG displayed, and you can select the display screen by simply typing **>ENTER<**. What you'll then see is a list of the program titles, the length of each program in terms of sectors used on the disk, and, at the top, the number of available and used sectors.

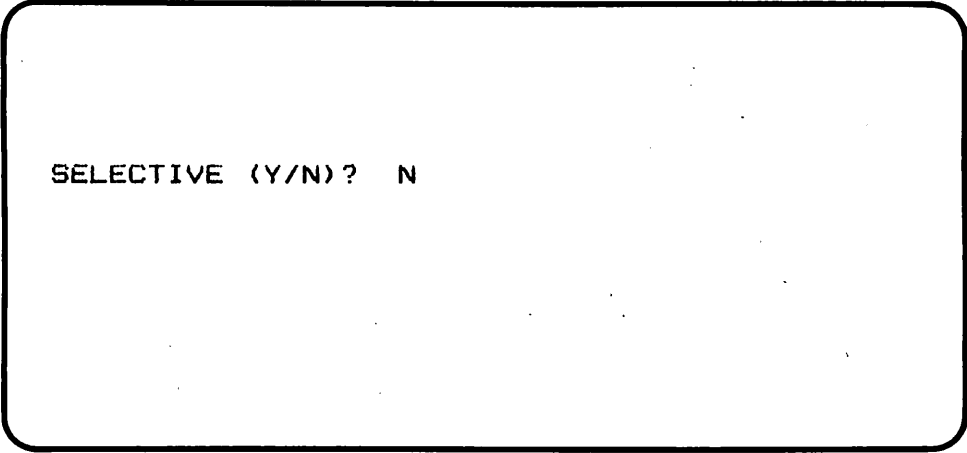
As you write more and more programs, you'll gradually get a feel for the amount of space that a given program takes up on the disk. Be sure that there is ample room when getting ready to record a program. A DISK FULL message can be disconcerting, and if you don't happen to have an initialized disk ready, you may end up losing the entire program you just painstakingly keyed into your computer. Therefore, always run the CATALOG function when you're not certain about the amount of remaining space available on the disk, and always have at least one initialized spare disk in reserve.

MAKING BACKUP DISKS

Another option that is included among the DISK COMMANDS is BACKUP DISK. The purpose of making backup disks is to protect the original against some type of damage that might result from constant use. For instance, the TI word processor that functions with your TI-99/4A consists of a plug-in module and a disk. The instruction book that accompanies the word processor suggests that you

make a backup disk before using the system, and that you then use the backup disk in your day-to-day operation, keeping the original in a safe place.

When you select the BACKUP DISK option, the display asks:



```
SELECTIVE (Y/N)?  N
```

which means that you have the choice of copying everything on the disk or simply one or another selected program. To copy the entire disk, simply press the >ENTER< key. The display will respond with the name of the disk to be copied and the message:



```
LOAD COPY DISK  
PRESS: PROC'D, REDO BEGIN OR BACK
```

Remove the master disk and insert a blank disk into the disk drive. Then press FCTN and 6 simultaneously for PROC'D. If the blank disk was not previously initialized, the display now shows:

```
DISK NOT INITIALIZED
INITIALIZE NEW DISK (Y/N)  Y
NEW DISKNAME?
```

Type in the name that you want to use for the backup disk and press >ENTER<. The display will respond with the same sequence of messages that we saw when we initialized a blank disk before. After a while the next message appears:

```
LOAD MASTER DISK
```

plus the name of the master disk and the message that tells you to again press the PROC'D keys (FCTN and 6). Now remove the

backup disk and load the master disk and then press the appropriate keys. After a moment the display will ask that you switch disks again in order to copy the first of the programs contained on the master disk onto the backup disk. That procedure is repeated until all programs have been copied, at which point the display responds with:

COMMAND COMPLETED
PRESS: PROC'D, REDO, BEGIN OR BACK

If you now press the keys for BEGIN and BACK, the display will return to the original title page.

There may also, of course, be other times when you will want to make backup disks. You might have a program that you wrote yourself that you expect to be using with considerable frequency, or you might have created some important text on your word processor that must be saved. Alternatively, you might want to combine programs from several disks onto one, though that can be accomplished by a somewhat simpler method, described below.

REARRANGING DISK CONTENTS

You may have recorded a number of entirely unrelated programs on a disk and subsequently decide that it would be more intelligent to keep related programs on the same disk. Thus you

might want one or several disks to be used entirely for educational programs, another for home- or kitchen-related programs, still others for business programs, and so on. The simplest way to accomplish this is to load one program at a time into the computer RAM, using:

```
OLD DSK1.FILENAME    >ENTER<
```

and then switch to the other disk and type:

```
SAVE DSK.1 FILENAME  >ENTER<
```

so the program will be recorded on the new disk. Then repeat that procedure for all programs that you want to record on that new disk. When all the related programs have thus been saved on the new disk, you might write the following program and save it too:

```
10 PRINT "      MENU:"  
20 PRINT "      = = = ="  
30 PRINT "FILENAME - DESCRIPTION"  
40 PRINT "FILENAME - DESCRIPTION"  
50 END
```

where FILENAME is the name of each program, using as many FILENAME - DESCRIPTION lines as you need in order to list all the programs on the disk, along with brief descriptions to remind you of the purpose of each of the individual programs. You can, of course, also call up a list of the programs by selecting the CATALOG option from the DISK MANAGER menu, but if there are more programs on the disk than can be displayed at one time, they'll scroll by too fast to be read.

Once all programs have been copied from the original disk onto several new disks, you can erase the original disk by using the INITIALIZE option, after which the disk can be used again.

3

Getting Started

Now that we have a reasonably clear idea of what the computer system is all about, let's begin to put our machine through some of its paces.

Computers can be used in two distinct fashions, usually referred to as *modes*: the *immediate mode* and the *deferred mode*. In the immediate mode the computer acts much like an everyday calculator, whereas in the deferred mode it executes instructions that have been keyed in the form of a program. For the time being, let's stick to the immediate mode as a means of getting acquainted.

IMMEDIATE MODE

To begin, turn on the expansion system, the computer, and the monitor as we did in Chapter 2. The display will show:



READY - PRESS ANY KEY TO BEGIN

Do that now, and the display will change to:

```
PRESS
1 FOR TI BASIC
2 FOR ``DISK MANAGER``
3 FOR ``DISK MANAGER``
4 FOR ``GESTION DE DISQUES``
```

For the time being we won't concern ourselves with selections 2, 3, and 4 (unless you want your instructions in German or French, you'll never use 3 and 4). Therefore, let's now press 1. The screen responds with:

```
TI BASIC READY
```

and a flashing square next to a > sign. That sign is referred to as the *prompt*, because it prompts us to key in some information. The

flashing square is the *cursor*, and it tells us where on the screen the next keyed-in character will appear.

Try typing:

1 + 1 =
and then press the >ENTER< key.

Absolutely nothing happens, because we didn't command the computer to do anything. The command we need to use in the immediate mode is PRINT. Print has nothing to do with your line printer. It simply tells the computer to PRINT the applicable information and data on the display screen. Now try:

PRINT 1 + 1 = >ENTER<

This time something does happen, but it's not what we had in mind. The display has responded with:



*INCORRECT STATEMENT

meaning that there was something wrong with what we typed. What was wrong was the use of the equal (=) sign. With that in mind, let's try once more. Type:

PRINT 1 + 1 >ENTER<

Success. The computer responds by displaying:

```
PRINT 1 + 1
      2
```

In other words, it performed the calculation and has displayed the result.

Now let's try something else. Type:

```
PRINT MARY + ANN  >ENTER<
```

The computer responds by displaying:

```
0
```

which doesn't seem to make any sense at all. Try again by typing:

```
PRINT "MARY" + "ANN"  >ENTER<
```

This time we get a message that reads:

STRING-NUMBER MISMATCH

which requires a bit of explanation. A *string* is anything, be it letters or digits, that is enclosed in quotation marks. But the computer cannot perform addition or any other mathematical function with strings, which is what is meant by the above message. Now type:

PRINT "MARY";"ANN" >ENTER<

This time the computer responds by displaying:

MARYANN

because the semicolon told the computer that there is not supposed to be a space between the two words (strings). Now type:

```
PRINT "MARY","ANN"    >ENTER<
```

using a comma instead of a semicolon. This time the result looks like this:



MARY

ANN

because the comma told the computer to space the two strings a certain distance apart.

Now, in order to illustrate the different ways in which the computer uses letters and numbers, let's try something else. But first let's get rid of all that stuff on the screen. Press the FCTN (function) key and at the same time the +/- key and watch as the display returns to the original design. Now press any key then 1, and we're back where we started. Type:

```
PRINT "MARY HAD A LITTLE LAMB"    >ENTER<
```

The result is:

MARY HAD A LITTLE LAMB

But now try:

```
PRINT "MARY SAID, "HELLO, JOE"  
WHEN HE CAME IN"  >ENTER<
```

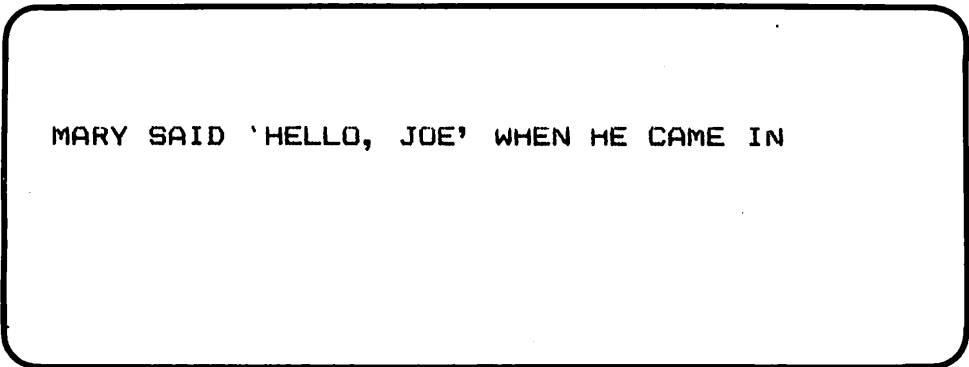
The result is:

*INCORRECT STATEMENT

because you cannot use quotation marks within quotation marks. What you can use instead are single quotation marks (apostrophes), like this:

```
PRINT "MARY SAID 'HELLO, JOE' WHEN HE CAME IN"    >ENTER<
```

This time the result is:



```
MARY SAID 'HELLO, JOE' WHEN HE CAME IN
```

because single quotation marks do not represent a command to the computer. Next, let's type:

```
A=1    >ENTER<
B=2    >ENTER<
PRINT A+B    >ENTER<
```

The computer responds with:



```
3
```

because we have told it that from now on, until we either change it or turn the computer off, it is to consider the letter A as 1 and the letter B as 2. In this case the letters A and B are referred to as *numeric variables* because they can be made to stand for any numerical value. For instance:

```
A=56/3
B=3*56
C=A+B-6
PRINT C
```

results in:



180.6666667

because the computer has performed all those calculations automatically. (Don't forget to press the >ENTER< key at the end of each line. Since that should by now have become second nature, we'll no longer mention it.) By the way, there is an alternative way that can be used to assign values to variables:

```
LET A=6
```

but the LET command is optional, and it seems silly to type something that is not needed. You may think that this business of assigning numeric values to variables (letters or letter combina-

tions) is kind of a strange way of doing things. It isn't, as you'll soon find out when we start to write some simple programs. But before that, let's look at another kind of variable. Type:

```
A$ = "MARY"  
B$ = "ANN"  
PRINT A$;B$
```

The display responds with:



MARYANN

In this case the dollar sign (\$) tells the computer that the letter represents a *string variable*, and it automatically translates that variable into the string that was assigned to it. Strings can consist of letters or digits or symbols such as commas, plus or minus signs, and so on, but they may not contain quotation marks, and they can be of any length, up to 255 characters (including blank spaces, which, to the computer, are characters). As you can imagine, using such string variables can save a lot of typing.

What we have done so far is of not much practical use. It simply helped to familiarize you with some of the very basic functions that are available to us. In practice computers are virtually never used in this immediate mode. The whole purpose of computers is to perform series of commands—programs—and to do this we must use the deferred mode. Let's start by writing a very simple program, using some of what we have already learned plus a few new concepts.

DEFERRED MODE

Programs are series of instructions that are entered in the form of individual lines. These lines must be numbered in order to tell the computer the order in which it is to execute the various commands. The lines may be numbered 1, 2, 3, and so on, or 10, 20, 30, and so on, or in any other consecutive order that appeals to you. It is good practice, however, to keep the line numbers at least 10 numbers apart, because as you start to write programs you will often find that you want to insert a line here or there, which cannot be done without renumbering (and retyping) all the remaining lines if you have left no space between line numbers.

TEMPERATURE CONVERSION PROGRAM

Our first little program is designed to convert degrees Fahrenheit to degrees Celsius and vice versa, and it demonstrates some of the basics of program writing. Let's look at it line by line:

Line 10 gives the title of the program. The REM stands for REMark, meaning that the line is simply a reminder. All REM lines are ignored by the computer.

Lines 20 and 30 assign a leading blank space followed by a string to the string variables F\$ and C\$.

Line 40, CALL CLEAR, is the command used by the TI-99/4A to clear the screen.

Line 50 prints the word MENU, a standard computer term that always refers to a list of available options.

Line 55 simply prints a line to make the display look better. The line number is 55 rather than 60 because I added the line 55 after the program had already been written.

Lines 60 and 70 print 1 F. TO C. and 2 C. TO F. Notice that the 1 is not included between the quotation marks. The reason will be explained when we get to line 90.

Line 75 is the same as line 55.

Line 80 asks you to select Fahrenheit to Celsius or Celsius to Fahrenheit by typing either 1 or 2. The INPUT command tells the computer to print the question, keyed in in quotation marks, and then to stop program execution until an answer to the question

TEMPERATURE CONVERSION PROGRAM

A simple program that converts degrees F, to degrees C, or vice versa.

```
10 REM  TEMPERATURE CONVERSION
20 F$=" DEGREES F."
30 C$=" DEGREES C."
40 CALL CLEAR
50 PRINT "MENU: "
55 PRINT "-----"
60 PRINT 1;" F. TO C."
70 PRINT 2;" C. TO F."
75 PRINT "-----"
80 INPUT "WHICH? ":WHICH
90 ON WHICH GOTO 100,200
100 PRINT
110 PRINT
120 INPUT "DEGREES F.? ":F
130 C=(F-32)/1.8
140 PRINT
150 PRINT C;C$
160 END
200 PRINT
210 PRINT
220 INPUT "DEGREES C.? ":C
230 F=C*1.8+32
240 PRINT
250 PRINT F;F$
260 END
```

has been keyed in and >ENTER< has been pressed. In TI BASIC, all INPUT commands use a colon (:) between the question and the variable that represents the answer to that question. Here the variable is WHICH. It could just as well have been simply W, or any other letter or letter combination. Once you have keyed in 1 or 2, WHICH represents your reply.

Line 90 tells the computer that if WHICH represents 1, it is to go to line 100; and if WHICH represents 2, it is to go to line 200. That is the reason for leaving the 1 and 2 outside the quotation marks. If they were inside the quotation marks, they would have become part of the string, and could not have been assigned to a numeric variable. The GOTO command simply means GO TO line number x. In TI BASIC the command can also be typed as GO TO.

Lines 100 and 110 simply place two blank lines into display to make the copy easier to read.

Line 120 asks you to key in the number of degrees Fahrenheit you

want to convert to degrees Celsius, assigning the value you type in response to that question to the numeric variable F.

Line 130 performs the required calculation and assigns the result to the variable C.

Line 140 creates a blank line.

Line 150 displays the result of the calculation along with DEGREES C., which was assigned to the string variable C\$ in line 30. You can see now why a leading blank space was included in the strings represented by F\$ and C\$. Without it the number and the word DEGREES would have run together.

Line 160 indicates to the computer that it has come to the end of the program.

Lines 200–260 are a repeat of lines 100–160, and they are used if you typed 2 in reply to the question in line 80.

In this little program we have made use of numeric as well as string variables (F and C, F\$ and C\$) and have learned to use the commands REM, CALL CLEAR, PRINT, INPUT, ON GOTO, and END. There are, of course, many more that make up the TI BASIC computer language, but for the time being we'll stick to those that are used most often.

STOPWATCH PROGRAM

Let's look at another short program that introduces a few more of the commonly used BASIC commands. The program is a timing program that acts like a stopwatch, except that it can be run at different speeds because it was originally designed for model railroaders who like to run their trains to what is referred to as scale time, where a scale hour might actually be 15 minutes.

When the program is run, it first displays its title (CLOCK) and then asks that you key in the timing speed. The number that produces the desired speed may vary among computers. With mine, typing 100 produces actual time (the smaller the number, the faster the clock, and vice versa). The display then requests that you type in the number of minutes after which you want the clock to stop, and after that you're told to press >ENTER< to start the timing process. The display then scrolls upward across the screen, reading:

```
0 Minutes and 1 Second
0 Minutes and 2 Seconds
```

and so on until it reaches the number of minutes you typed in earlier. You will notice that I have used lowercase letters here. On the screen they appear as smaller uppercase letters, but the printer automatically accepts them as lower case. (Later on we'll come back to this program and convert it to a kitchen clock that produces a loud tone when the selected time has been reached.)

Now let's look at each line of the program and see what takes place.

Lines 50–80 assign words to string variables.

Line 100 clears the screen of anything left over in the display.

Lines 110–130 represent a FOR . . . TO . . . NEXT loop; by which the computer goes around in circles, repeating the command in the center line as often as is indicated by the number after TO, in this case 10 times. Here it is used to place 10 blank lines on the screen in order to have the word "Clock" printed in the center of the screen. Such loops have lots of uses.

Lines 140–160 print the word Clock in the center of the screen between two lines. The TAB(12) command moves the first character 12 spaces in from the left edge of the screen.

Lines 170 and 180 produce two blank lines.

Line 190 asks you to key in the speed at which you want the clock to run, and assigns that value to the variable SPEED.

Lines 200 and 210 print two more blank lines.

Line 220 asks you to decide on a time limit in minutes, assigning it to the variable TIME.

Lines 230 and 240 print two more blank lines.

Line 250 requests you to press >ENTER< to start the timing process.

Lines 260–280 use another loop to place four blank lines into display. These four lines are used later on to separate the minute/second lines as they scroll up the screen.

Lines 290 and 300 use another loop to create a time lapse, the length of which is determined by the value that, earlier in the program, was assigned to the variable SPEED.

Line 310 raises the value of A by 1 each time the computer passes this line. A, in this case, represents the number of seconds that have elapsed.

Line 320 uses an IF . . . THEN command to send the computer to another line if a certain condition is met—in this case, if the value of A has reached 60. This command is used very often because it permits us to have the computer produce alternate results based on changing conditions.

Line 330 uses another IF . . . THEN command to tell the computer that if A is larger than (>) 1, then go to line 360. The purpose is to print the singular “Second” for 1 second, and the plural for more than 1 second.

Line 340 causes the string that was assigned SS\$ to be assigned to S\$, as well. The aim is to have the display read “1 second” rather than “1 seconds,” which would be annoying.

Line 350 tells the computer to skip the next line and go to line 370.

Line 360 assigns the string that was assigned to SSS\$ to S\$, now causing seconds to be read as the plural.

Line 370 causes the computer to go to line 400 if B equals 1. B, in this program, represents the number of elapsed minutes.

Line 380 is similar to line 360 for minutes.

Line 390 causes the computer to skip line 400.

Line 400 is similar to line 340.

Line 410 causes the number of minutes and seconds to be displayed. Note the blank spaces on either side of “and” inside the quotation marks.

Line 420 tells the computer to go back to line 260 and start the process all over again.

STOPWATCH PROGRAM

This program turns your computer into a clock, displaying minutes and seconds.

```
50 MM$="Minute"
60 MMM$="Minutes"
70 SS$="Second"
80 SSS$="Seconds"
100 CALL CLEAR
110 FOR X=1 TO 10
120 PRINT
130 NEXT X
140 PRINT TAB(12); "-----"
150 PRINT TAB(12); "Clock"
160 PRINT TAB(12); "-----"
170 PRINT
180 PRINT
190 INPUT "Timing speed? ":SPEED
200 PRINT
210 PRINT
220 INPUT "Stop after how many minutes? ":TIME
230 PRINT
240 PRINT
250 INPUT "To start press >ENTER< ":START$
260 FOR X=1 TO 4
270 PRINT
280 NEXT X
290 FOR PAUSE=1 TO SPEED
300 NEXT PAUSE
310 A=A+1
320 IF A=60 THEN 450
330 IF A>1 THEN 360
340 S$=SS$
350 GOTO 370
360 S$=SSS$
370 IF B=1 THEN 400
380 M$=MMM$
390 GOTO 410
400 M$=MM$
410 PRINT B;M$;" and ";A;S$
420 GOTO 260
450 A=0
460 B=B+1
470 FOR X=1 TO 4
480 PRINT
490 NEXT X
500 PRINT B;M$;" and ";A;S$
510 IF B=TIME THEN 530
520 GOTO 260
530 END
```

Line 450 returns the value of A (seconds) to 0 after 60 seconds have elapsed.

Line 460 raises the value of B (minutes) by 1 each time the computer reaches this line.

Lines 470–490 uses a FOR . . . NEXT loop to insert four blank lines to accentuate the passing of 1 minute.

Line 500 is similar to line 410.

Line 510 tells the computer to go to the END line when the value of B (minutes) equals the value of TIME, representing the previously entered time limit.

Line 520 is similar to line 420.

Line 530 tells the computer that it has reached the end of the program.

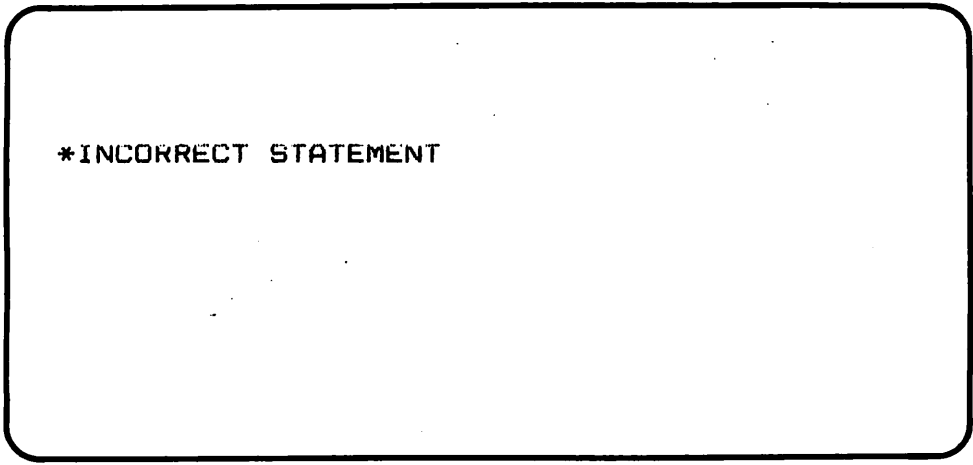
Program writing can be a lot of fun, and it can also serve a variety of practical and useful purposes. Don't be intimidated by the number and variety of commands that make up the TI BASIC language. At this stage there is no need to try to memorize them all. By far the easiest way to learn them is actually to type some (the more, the better) programs into your computer and observe the effects produced by the different commands. The programs in this book are arranged by general subject matter and also in ascending order of difficulty, eventually using all the commands that are available on your TI-99/4A. If you have a disk drive or a means of storing programs on magnetic tape, be sure to do so. Even though you may not initially envision using a particular program again, you will find, as you accumulate experience, that it is often possible to make some minor modifications in a program so that it can serve a variety of purposes.

One word of warning: Computers are extremely fussy about the manner in which information, data, and commands are typed into them. Blank spaces in the wrong place, stray punctuation marks, or the use of lowercase letters can, under certain conditions, cause programs to produce incorrect results or to refuse to work at all. Even though the TI-99/4A will accept certain commands typed in lower case, that is not true under all circumstances, and it is good practice to use upper case for all commands. Also, protected words,

the words and phrases that constitute TI BASIC, must always be used with blank spaces on either side. For instance:

FOR X=1TO10

produces an:



*INCORRECT STATEMENT

message because TO is a protected word. The line must be keyed in as:

FOR X=1 TO 10

with spaces on either side of TO.

4

Commercial Software

Although Texas Instruments is no longer manufacturing and marketing TI-99/4A Home Computer, the company publishes a catalog of the software that is available for the TI-99/4A Home Computer. Although it would be impossible to discuss and describe the hundreds of programs in detail, we shall talk about a dozen or so programs that are more or less representative of some of the various categories.

The Texas Instruments Home Computer Program Library catalog is divided into several groups: *Home Management/Personal Finance; Education; Home Entertainment; Computer Programming Aids; and Other Programs*, where the last category includes such things as engineering and statistical programs.

Most programs are contained in a plug-in module and come with an instruction booklet that explains how the program works. (A few are provided on disk.) A word of caution: Some of the booklets examined were apparently written for the earlier TI-99 and not the 4A, and they contain instructions that require special keys on the 4A. Use the key combinations shown on the plastic strip instead.

My personal feeling about software is that, with the possible exception of such highly sophisticated programs as word processors and spreadsheet programs, each program should include complete and easy-to-follow instructions and prompts that obviate the need for studying instruction books. In this area a number of the programs I have examined fall short of that criterion.

HOME MANAGEMENT/PERSONAL FINANCE

PERSONAL RECORD KEEPING

This program is designed to permit you to create any type of records and to save them on a storage device such as a cassette or disk. The program includes a means of customizing your records to fit any type of specialized need. You can use it to create an address list, an inventory, product lists with specifications, and so on (a number of typical examples are included in the instruction booklet), but there are limits to the number of characters that can be used. In order to demonstrate how the program functions, let's create a short inventory file, print it on the line printer, and save it on disk.

When the program is first activated it displays a title page followed by ONE MOMENT PLEASE . . . and then this display:

ENTER DATE

MONTH?

DAY?

YEAR? 19

which asks you to type in the date on which the record file is being created or updated. When the date has been entered, the display will change as on the following page.

PRINTING DEVICE (Y OR N)?

and after you have typed Y it continues with:

PRINTING DEVICE NAME?

requesting that you type in printer identification which, in the case of the TI-99/4A Impact Printer, is RS232. The next display is:

```
PRESS TO
1      CREATE A FILE
2      LOAD A FILE
```

Since this is the first time we're using the program and no previously created file exists as yet, let's type 1.

```
FILE NAME?
```

Type INVENTORY, which is nine characters long, the maximum allowable for a file name.

ITEM #1
ITEM NAME?

Type PROD # to reserve this first item in the record for product identification numbers. This adds the following to the display:

PRESS FOR
1 CHARACTERS
2 INTEGER
3 DECIMAL
4 SCIENTIFIC NOTATION

Your answers to these choices limit the type of data that may be entered in this particular category. If all your product numbers are simple integers, then you can select 2. If, on the other hand, your product numbers include letters, slashes (/), or any other symbols, be sure to select 1, because the integer choice will not allow you to enter a blank space, which is regarded as a character by the computer. The display now asks:

MAX # CHARACTERS?

Be sure to reserve space for a sufficient number of characters to permit future expansion, because once the limit has been established, it cannot be changed. Maybe 5 would be a good choice. When this has been keyed in, the computer asks for ITEM #2, and so on, each time asking the same series of questions. After you have entered the format for the last item, press BACK and you're presented with the MAIN INDEX:

PRESS TO
1 SEE FILE STRUCTURE
2 ADD PAGES
3 DISPLAY PAGES
4 CHANGE PAGES
5 ANALYZE PAGES
6 PRINT PAGES OR REPORT
7 SAVE DATA FILE
8 EXIT

Since we want to create a new inventory file, let's type 2. The display will now prompt you to enter your data with:

```
PAGE #          1
1 PROD#
2 DESCRIPTN
3 # ON HAND
4 REORDER #
5 MFG COST
6 LIST $
7 SHIP WT
```

Each page number represents one record. If you want to visualize the procedure, think of file cards. After you have typed in the data for the above, the display will change to PAGE #2, and so on until you have entered all your inventory data. Then type BACK, and the program will return to the MAIN INDEX, where you should now select choice #6 if you want a printout (see Figure 4-1A, B, C), choice #7 if you want to save your records on disk. You will be asked to

```
FILE:  INVENTORY
DATE:  8/15/83
TITLE: INVENTORY
```

```
INDEX
0 = PAGE #
1 = PROD #
2 = DESCRIPTN
3 = # ON HAND
4 = REORDER #
5 = MFG COST
6 = LIST $
7 = SHIP WT
```

Figure 4-1A. The printer produces a record of the file. Note the typo (B in DESCRIPTN).

FILE STRUCTURE				

ITEM	TYPE	WIDTH	DEC	
1 PROD #	CHAR	5	0	
2 DESCRIBTN	CHAR	15	0	
3 # ON HAND	INT	5	0	
4 REORDER #	INT	5	0	
5 MFG COST	DEC	6	2	
6 LIST \$	DEC	6	2	
7 SHIP WT	INT	3	0	

Figure 4-1B. It then prints the file structure.

0	1	2	3	4	5	6	7
1	10/14	PRODUCT A	78	75	33.78	65.95	7
2	10/15	PRODUCT B	48	50	15.88	29.95	3
3	10/16	PRODUCT C	114	100	5.76	9.95	2
4	10/8A	PRODUCT D	73	67	45.27	90.00	11
5	10/21	PRODUCT E	77	75	14.63	25.95	3

Figure 4-1C. And finally it prints the data you have keyed in.

enter the file name under which you want the file recorded. This file must include the name of the storage device, for instance:ql

DSK.1INVENTORY

which will cause your records to be recorded on disk for future recall, at which time you can select the data you want to inspect by page number or by item. Thus if you specify a particular item number, all pages that contain the data for that item will be displayed.

Once the program has been activated, it incorporates reasonably

easy-to-follow instructions and prompts. Within its limitations it's a versatile and useful program (though, with a bit of practice, you could write one yourself that would do the same thing).

HOUSEHOLD BUDGET MANAGEMENT

This program, too, includes fairly complete instructions and prompts, although some of the abbreviations used are a bit obscure. The program includes a demonstration that uses arbitrary figures for some 34 preselected expense and income categories (see Figure 4-2). Once you have run the demonstration program, you'll do best

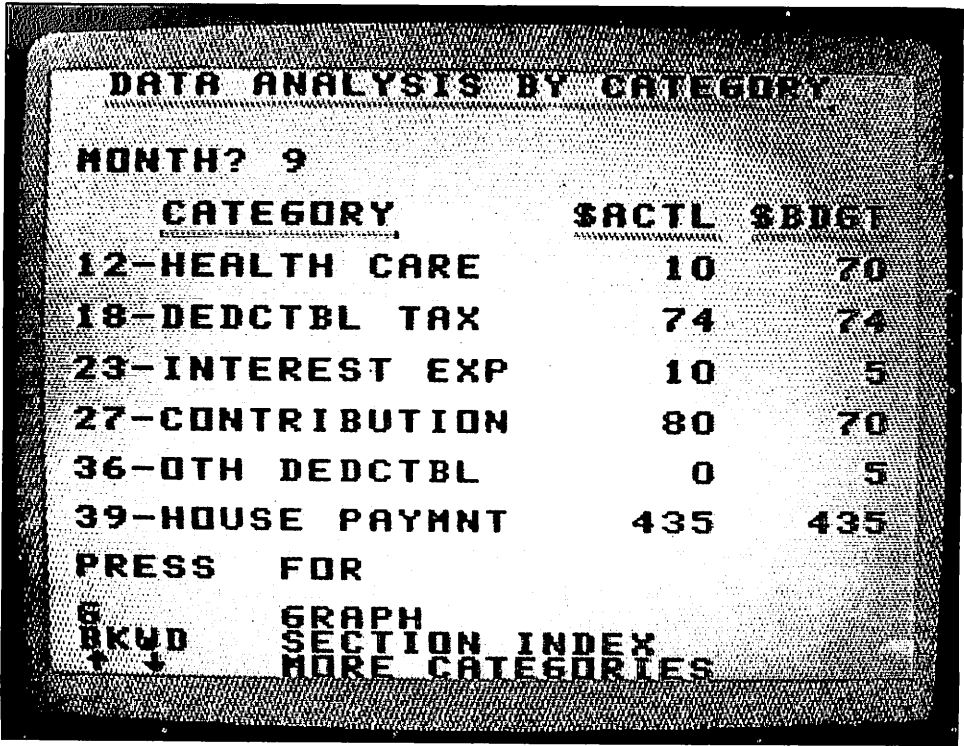


Figure 4-2. Demonstration data for the Household Budget Management program.

to start from scratch and select the first choice from the MAIN INDEX:

1 ENTER INCOME/EXPENSE

after which you will be asked to enter the category number for which you want to enter data. The trouble is that it is quite impossible to remember all those category numbers and you either have to consult the booklet, which lists them on page 14, or you have to type AID after each entry in order to find the appropriate category number for your next entry.

You can enter actual and/or projected income and expense figures, and you can set up a budget for each month. The program displays how you're doing in terms of actual income and expenditures versus the budget or projections, and it can do this in the form of tables or graphs (see Figure 4-3 on page 48). The program is a bit cumbersome to use, primarily because of the excessive number of categories. But all categories that are not applicable to your situation can be eliminated, thus reducing the number of active categories to a minimum. During the early portion of the program you're asked whether you want to go with the 34 preselected categories or not. Type N and you'll have an opportunity to select any one of a total of 99 categories relative to different types of income and almost every conceivable type of expense. I would suggest using a minimum number in order to reduce the amount of typing as well as clutter in the display. Then, when you type AID to see the category numbers, only those that you selected will be displayed.

The program might be useful for households that must adhere to a relatively stringent budget. But unless data are updated continuously, it won't do much good. The program does require a storage device such as a cassette recorder or a disk drive, because all data in the module remain unchanged and all changes that you make, along with all data that you entered, will be lost unless you record them on disk or tape.

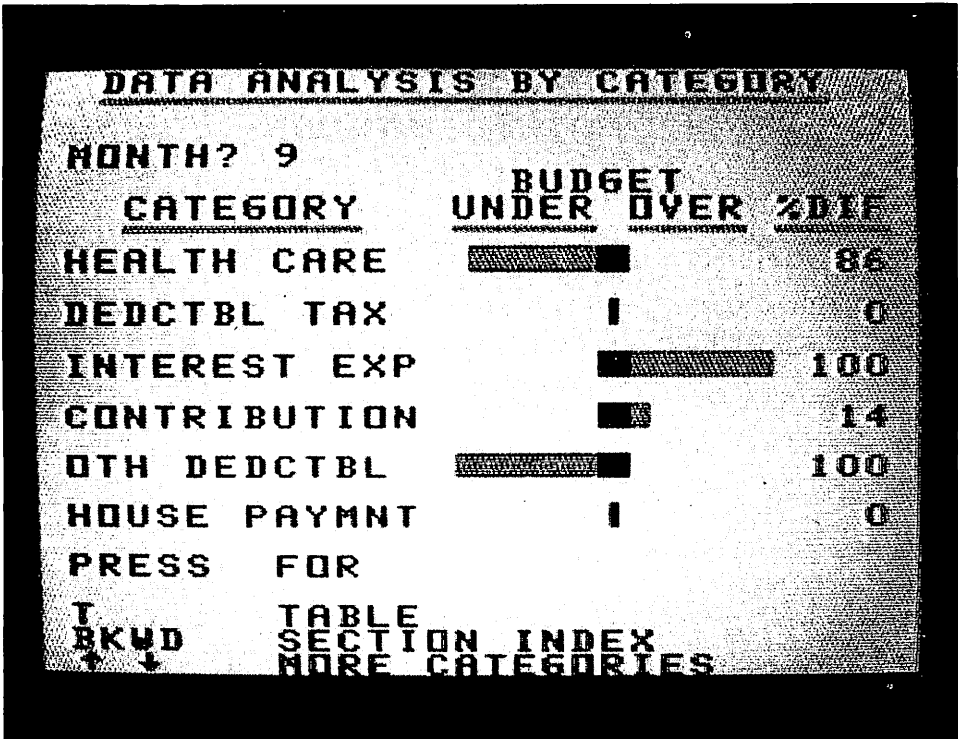


Figure 4-3. The Household Budget Management program can show graphically how you're doing.

SECURITIES ANALYSIS

This program deals with stocks, bonds, dividends, and compound interest, and its worst fault appears to be that it seems to take forever to come up with the various results. In order to use the program you must be familiar with the terms used in dealing with stocks, bonds, options, options spreads, and so on, which I am not (see Figure 4-4 if you are). I can only assume that it is reasonably valuable to people who are actively involved in the stock and bond markets.

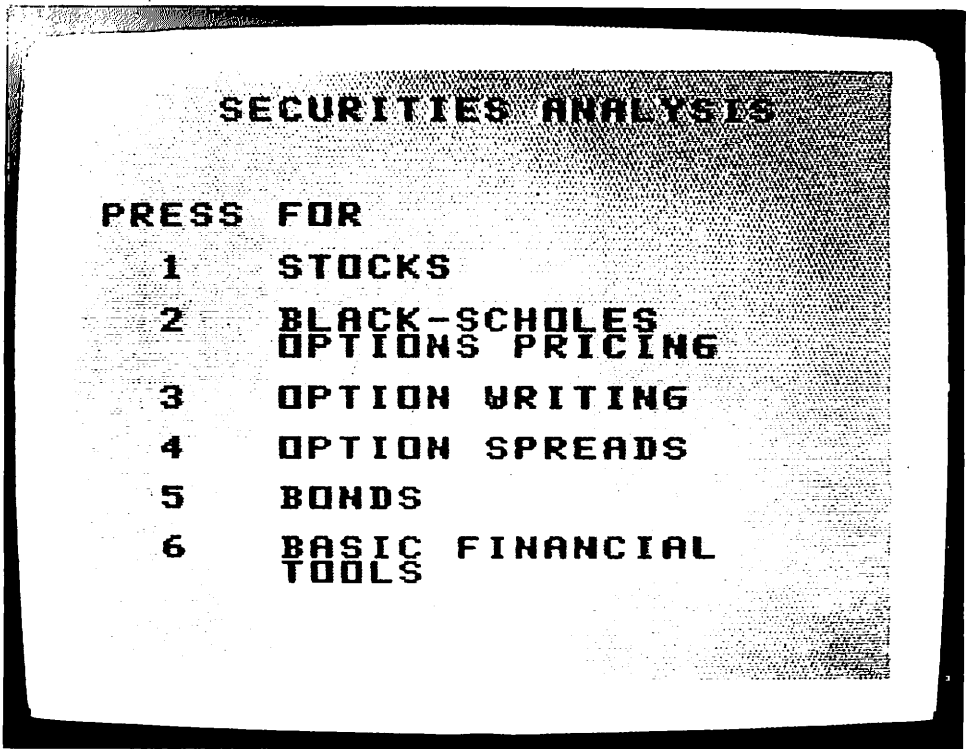


Figure 4-4. The Securities Analysis program starts by offering six choices.

EDUCATION

ADDITION AND SUBTRACTION 1

This program is designed for the very young, dealing with single-digit figures and using stylized graphics (see Figure 4-5 on page 50), music, and voice (if the optional voice synthesizer is available) to animate the program. The trouble with it is that it is extremely slow, and interaction between the computer and the student does not appear to me to be sufficiently rapid to hold the attention of any reasonably intelligent child. And once it has been activated, there is no way to exit any portion of the program and to

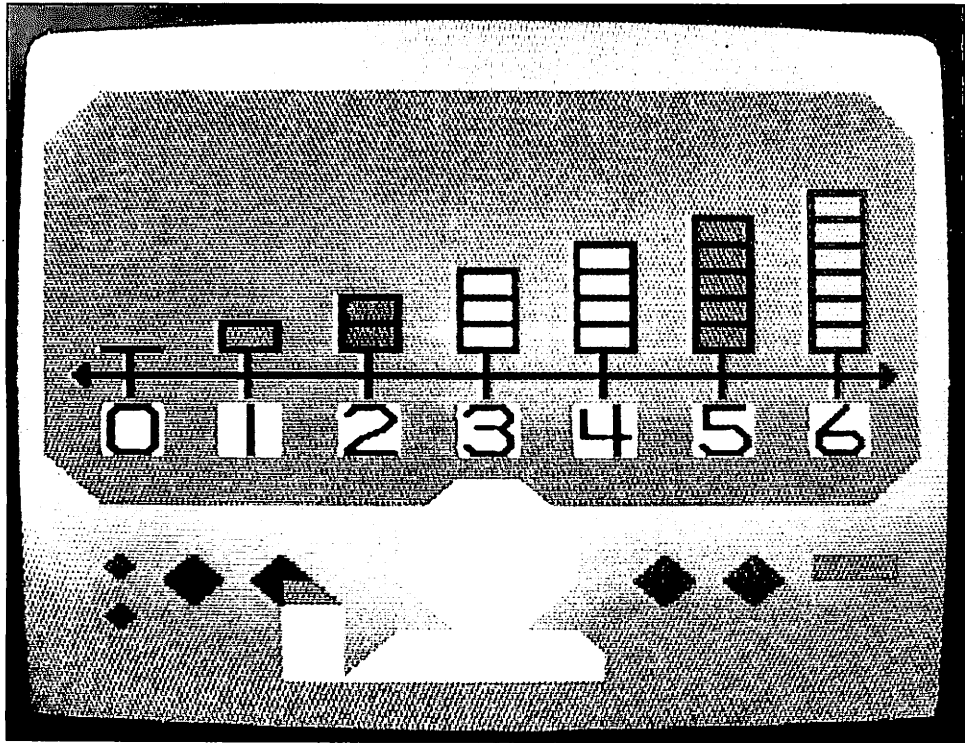


Figure 4-5. Addition and Subtraction 1 is distinguished by attractive graphics.

proceed to a more advanced portion without turning the computer off and then on again.

The program consists of nine subprograms, which are displayed at the start:

```

PRESS FOR
1      COUNTING BARS
2      GETTING READY
3      ADDITION ACTION
4      ADD ANOTHER WAY
5      SUBTRACTION ACTION
6      SUBTRACT ANOTHER WAY
7      ACROSS AND DOWN
8      ADDITION TABLE
9      SUBTRACTION TABLE
    
```

If number 1 is selected, the program will gradually (though exasperatingly slowly) go through one category after another. Furthermore, the voice keeps goading the student if after too long a period of time no answer has been typed in to a question. I like the visual and audio effects. I only wish it could be speeded up a bit.

ADDITION AND SUBTRACTION 2

This program is very similar to the one discussed above except that it deals with very slightly more advanced subjects such as three-number equations and a few two-digit figures. Here, too, the graphic, audio, and voice effects are outstanding (see Figure 4-6), but it is full of pauses during which nothing happens. Similarly, this program, too, cannot be exited except by turning the computer off.

Neither of these two programs requires the Peripheral Expan-

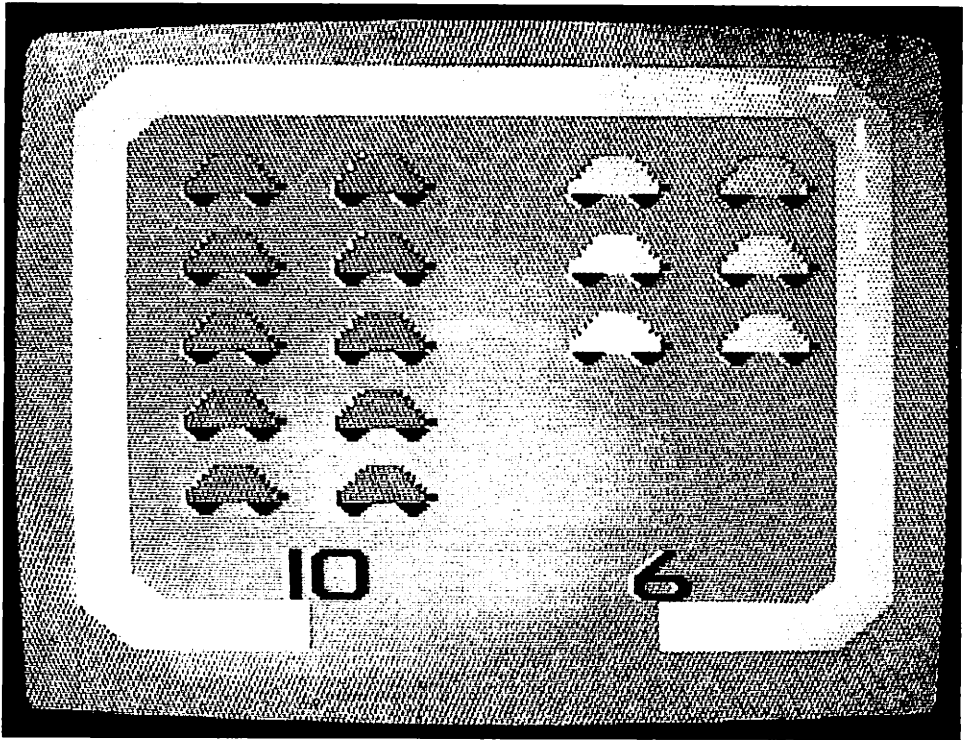


Figure 4-6. Addition and Subtraction 2 uses multinumber equations.

sion Unit, because neither calls for the use of the printer, disk drive, or the Extended Memory. Both were authored by Thomas Hartsig and were produced for Texas Instruments by Scott, Foresman and Company.

EARLY READING AND READING FUN

These two programs use the same style of graphics along with music and voice in two different types of reading lessons for the beginner. Both use very short, simple stories illustrated with stylized graphics to help the student recognize words, (see Figure 4-7). The second program, which is a little more advanced, also asks the student to think by presenting written problems with multiple-choice answers. The lettering, using upper- and lowercase letters, is



Figure 4-7. Early Reading uses simple stories.

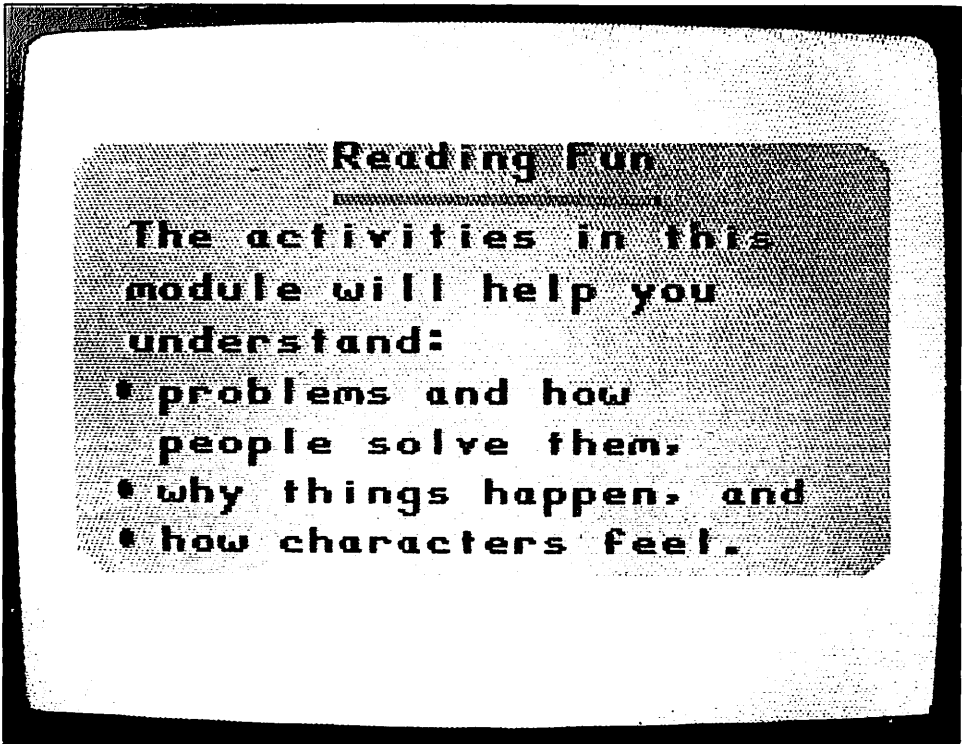


Figure 4-8. The upper- and lowercase letters are easy to read.

clear and easy to read (see Figure 4-8 on page 54). Unlike the programs discussed above, these two do not suffer from interminable pauses during which nothing happens. I believe that, depending on the student's reading ability, they will hold the child's interest, although older students with reading disabilities may object to the naivete of the subject matter.

SCHOLASTIC SPELLING, LEVEL 5

This program was produced for Texas Instruments by Scholastic Spelling, Inc. When it is first activated the display will give you a prompt:

What lesson would you like?
Press 1 to 36.

which is annoying because we don't know what those lessons are unless we study the accompanying book, where they are listed on page 3. Some lessons display words with a particular type of sound, such as short a's and long ones. In others the voice reads words and asks the student to type them. Unfortunately, some of these computer-generated words are hard to understand. Another problem is the size of the lowercase letters. Because of the relatively low resolution of the TI monitor, certain letters such as "m" and "w" are extremely hard to read (see Figure 4-9). This program, too, uses graphics, music, and, of course, voice to keep things fairly lively. The total number of words used in this program is 600, and students at the fifth level should have little trouble with them.

COMPUTER MATH GAMES II

This program was produced for Texas Instruments by Addison-Wesley Publishing Company. It uses a different and rather interesting approach to various relatively simple math problems by playing one of five games:

Your Number's up
Math Basketball
Match up

Tic-Tac-Math
Horse Race

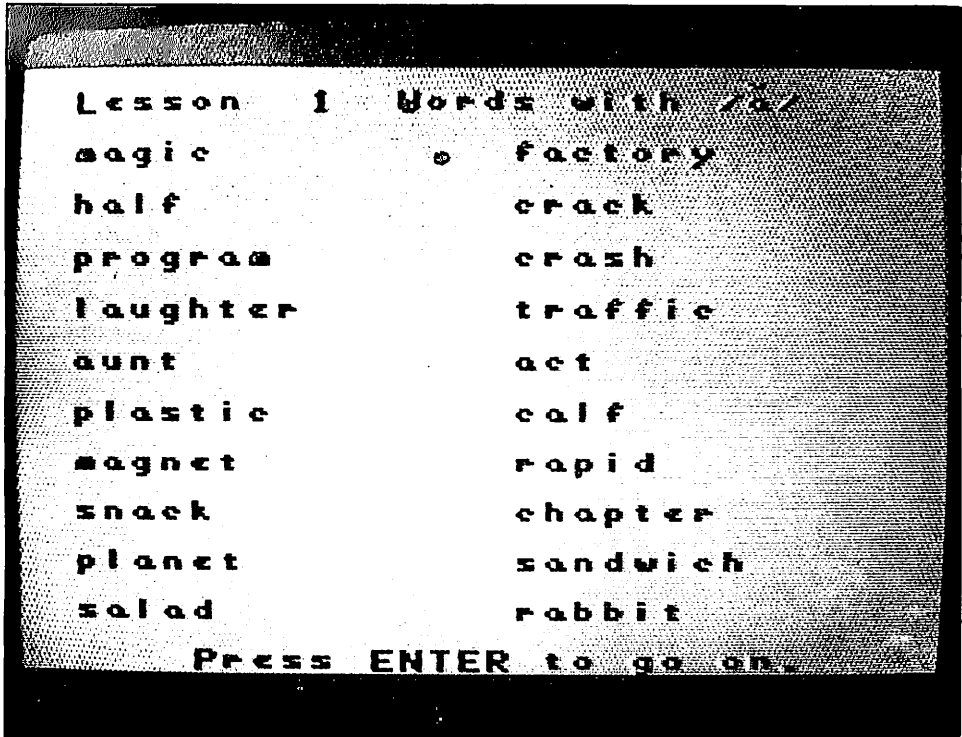


Figure 4-9. Some of the lowercase letters in this program are hard to read.

Some of these, such as the "Match up" game, are really quite difficult and require considerable concentration despite the fact that they're limited to one- or two-digit numbers (see Figure 4-10 on page 56). All are designed to be played by several players (though they can be played solo), keeping score for each player with the score being based not only on the number of right and wrong answers, but also on the time it takes a player to come up with his or her answer. The program uses pleasant graphics and music but no voice.

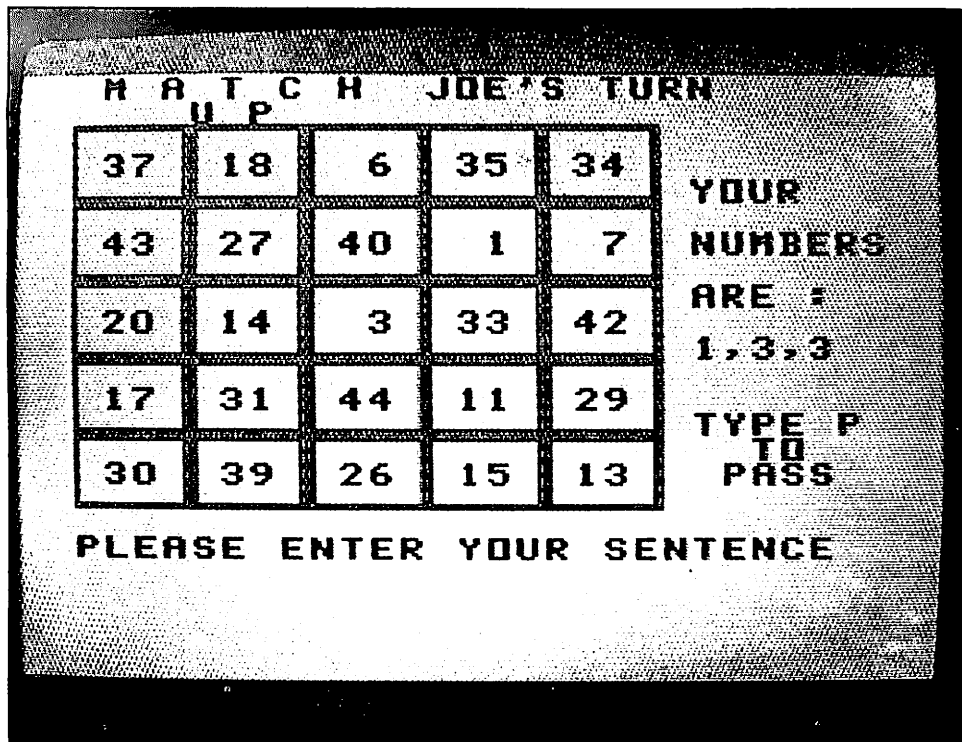


Figure 4-10. Some of the games in Computer Math Games II require considerable concentration.

ALLIGATOR MIX

Although this program looks more like an arcade game than an educational exercise, it is definitely designed to improve the ability of students to recognize whether an equation (single-digit addition or subtraction) and the related result are correct or not. It uses an alligator figure on which the result is displayed and a fish-like shape that floats toward the alligator on which the equation is displayed (see Figure 4-11). The student is to hit the space bar (or the fire button on the joystick) if the displayed result is correct, and nothing if it is wrong. The time available to the student to decide whether or not to press the space bar depends on the skill level selected and ranges from quite fast to very slow. When one of the upper skill levels is selected, it takes both considerable reading ability (with reference to numbers) and manual dexterity to come up with a reasonably good score.

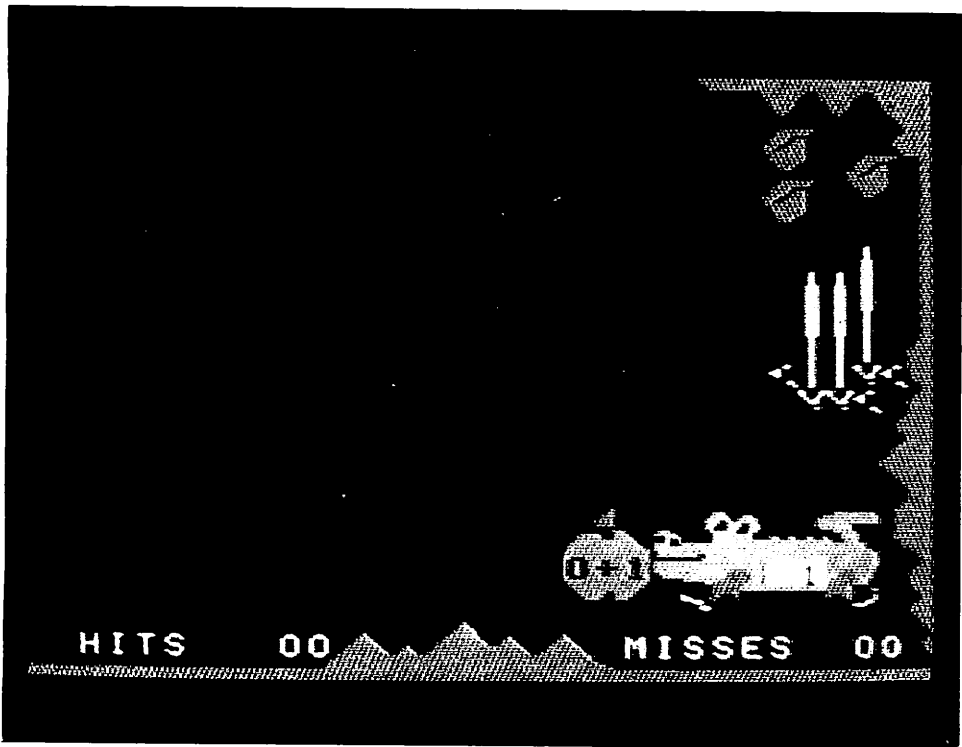


Figure 4-11. Alligator Mix combines arcade game graphics with simple math.

HOME ENTERTAINMENT

VIDEO GRAPHS

This program either displays a variety of abstract graphic patterns, or gives the player an opportunity to create his or her own graphic designs (see Figure 4-12 on page 58). The player can also interact with preprogrammed graphic patterns. In order to play this game you must first write to Texas Instruments for a keyboard overlay, which is sent free of charge (I wonder why it is not included to begin with). The program is fun for those who like to experiment with graphic designs. Others may find it less than fascinating. In the catalog it is listed under "Education," but I don't believe that it belongs there.

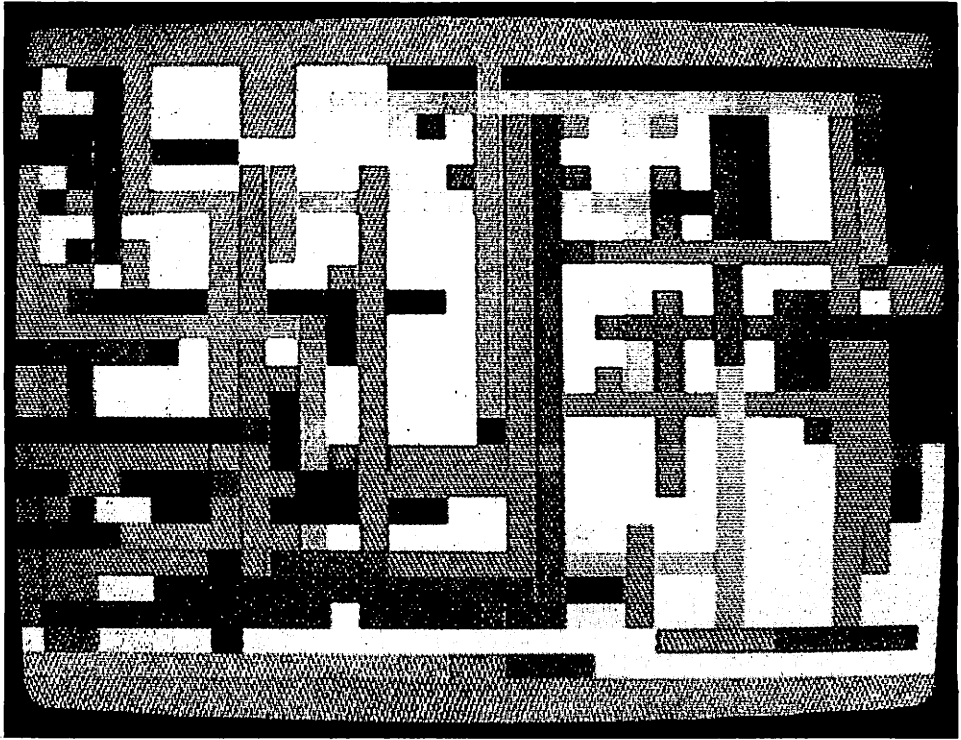


Figure 4-12. Video Graphs creates a variety of graphic images.

ALPINER

This is the only program I have selected that can be described as a real video game. The game can be played on the keyboard or with joysticks by either one or two players. It consists of a little man who is supposed to climb a bunch of mountains and who is confronted by all manner of obstacles including falling rocks, avalanches, angry bears, and so on (see Figure 4-13). The audio portion of the program is interesting in that it continuously plays a melody over which we hear two different voices, one of a man and another of a woman. The voices are obviously prerecorded and not computer-generated. In several tries I have never been able to get that little man all the way up the mountain, but then I don't really have the patience for games like this.

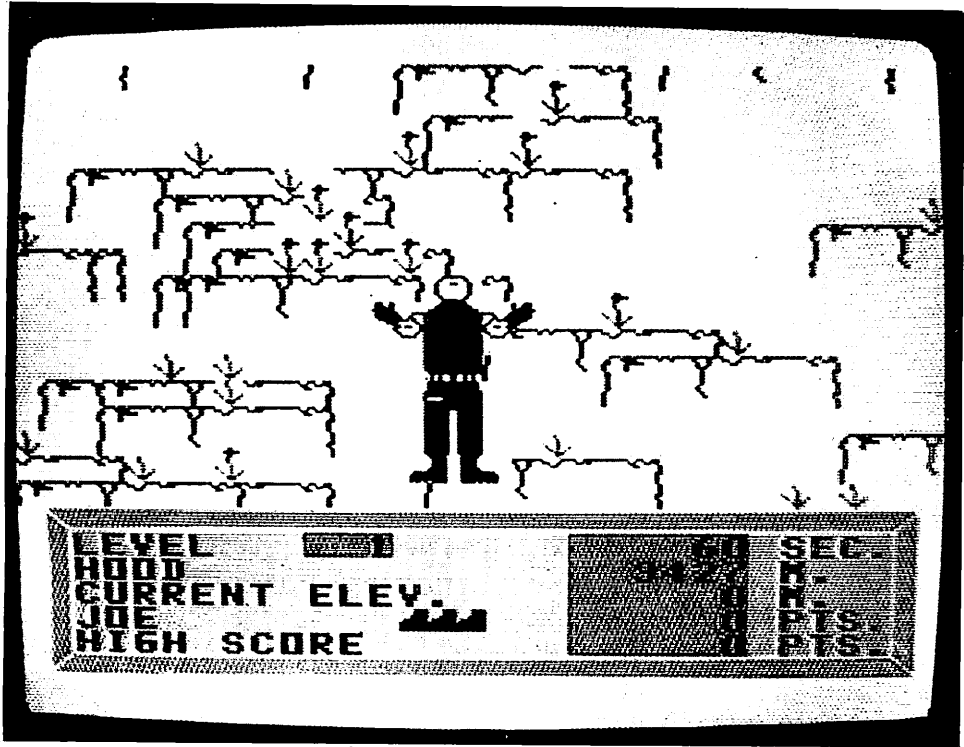


Figure 4-13. The Alpiners must climb mountains past all manner of obstacles.

Before ordering any program, stop in at your dealer's store and ask for a demonstration—otherwise you may waste a lot of time and money before you find the type of program you're actually looking for.

In the next chapter we'll look at the word processor that is designed to function with the TI-99/4A. It, too, technically falls into the category of commercial software, but it differs greatly from what we have discussed up to now.

5

Word Processing

Word processors are programs, not pieces of hardware. (Though there are so-called dedicated word processors, the kind we often see on TV in scenes that take place in newspaper offices, these are, in fact, microcomputers with built-in word processing software, designed to perform no function other than word processing.)

The purpose of a word processor is to permit you to produce text—to use your computer in place of a typewriter—because writing documents of any type on a word processor has advantages not available with typewriters. We can make corrections without using an eraser. We can delete copy or insert single words, entire sentences, or blocks of copy anywhere in a document. We can shift copy around, and we can change the format in which the final document is to be printed.

Since the whole purpose of word processing is to produce printed material, the first requirement for word processing, obviously, is that you have a line printer and that you know how to use it. Let's begin, therefore, by discussing line printers. (If you don't have a line printer and don't expect to opt for one, you may want to skip this chapter. On the other hand, even if you do not anticipate doing very much word processing, you may want to consider a line printer anyway: Several of the programs reproduced in the program sections of this book do assume the availability of a line printer, and those in turn may give you ideas for other non-word processing uses for a line printer.)

LINE PRINTERS

Line printers fall into two primary categories: *dot matrix printers* and *daisywheel printers*. Dot matrix printers produce characters that consist of closely spaced individual dots. As a general rule, dot matrix printers print from left to right and right to left at a rapid rate; with some it is possible to print all manner of graphic designs. Daisywheel printers, also referred to as *letter-quality printers*, operate on a principle that is similar to the one used by the IBM Selectric typewriter. The individual characters, instead of being contained on a metal ball (as in the IBM Selectric), are embossed on a wheel that looks much like a flower, therefore the name. The wheels can be changed to provide different type faces, and the quality of the printed material is comparable to and often better than that resulting from a good-quality typewriter. Daisywheel printers are slower than most dot matrix printers, and considerably more expensive.

Your TI-99/4A Home Computer can be used with a number of different makes and models of printers, and you might want to discuss with your dealer the make and model that best suits your needs. Texas Instruments markets the TI-99/4A Impact Printer, a relatively low-cost dot matrix printer that is capable of producing a variety of type faces and type sizes (see Figure 5-1 on page 62) as well as graphic images. The print quality is comparable to some of the best on the market; all program listings reproduced in this book were printed with that printer.

For normal use, you can operate the printer without any special commands other than the `OPEN # 1:"RS232"` or `LIST "RS232"` command that is used to access the printer from the keyboard. To produce special effects, you must use any one or more of a great many command codes that are described in some detail in the reference manual that is provided with the printer.

The printer is designed to be used exclusively with fan-fold, perforated computer paper. It does not have the friction-feed option available on some others that permits the use of ordinary paper such as stationery, envelopes, or rolls of paper without perforated edges.

Figure 5-1. The TI-99/4A Impact Printer includes a test program that prints its characters.

For first-time use, be sure to follow the set-up instructions carefully and be especially careful when loading the paper, a tricky operation that is not well illustrated in the instructions. One annoying characteristic of the printer is the fact that it is not possible to make the printer start printing at the top of a page without first running an entire blank page through.

What follows are most of the commands that you're likely to use, along with sample printouts resulting from those commands.

LIST "RS232" causes the program that is currently in the computer memory to be printed line by line.

LIST "RS232":200-500 causes lines 200–500 of the program in memory to be printed.

100 OPEN #1:"RS232"

110 PRINT #1:"Text"

120 CLOSE #1 are the program lines that are used under normal circumstances to access the printer, to print data (strings must be in quotation marks, numeric data need not be), and to disengage the printer.

CHR\$(27);CHR\$(68) is the command code that allows you to create tab settings in order to print data in a certain format (see Figure 5-2 on page 64). The "D" in line 110 can be used because the ASCII character code for D is 68; thus **CHR\$(68)** and the letter D are interchangeable. **CHR\$(10)** and **CHR\$(15)** represent the tab positions, where the numbers can be changed to represent tab positions of your choice. **CHR\$(0)** represents the end of the tab command sequence, whereas **CHR\$(9)** tells the printer to go to the next preset tab position. In most cases you'll probably be better off simply spacing the material to be printed in your program the way you want it, because such long lists of instructions are prone to typing errors.

```
100 OPEN #1:"RS232"  
110 PRINT #1:CHR$(27);"D";CHR$(10);CHR$(20);  
    CHR$(0);"NAME";CHR$(9);"CITY";CHR$(9);"STATE"  
120 CLOSE #1  
130 END
```

Figure 5-2. CHR\$(27) permits you to set tab positions.

CHR\$(14) causes characters to be printed in an enlarged format.

CHR\$(15) produces a condensed format.

CHR\$(20) cancels CHR\$(14).

CHR\$(18) cancels CHR\$(15).

CHR\$(27);CHR\$(69) causes characters to be printed in the emphasized format [CHR\$(69) = "E"].

CHR\$(27);CHR\$(70) cancels the above [CHR\$(70) = "F"].

CHR\$(27);CHR\$(71) causes characters to be double-printed [CHR\$(71) = "G"].

CHR\$(27);CHR\$(72) cancels the above [CHR\$(72) = "H"].

The short program in Figure 5-3 uses these character designation codes and shows the resulting printout.

CHR\$(27);CHR\$(75) calls up the normal-density graphics mode. [CHR\$(75) = "K"]. For an explanation of how to create different graphic effects, read the instructions in the owner's manual. Because the on/off position for each of the print wires that print

```

100 OPEN #1:"RS232"
110 PRINT #1:"NORMAL";CHR$(14);"ENLARGED";CHR$(20);
    "NORMAL"
120 PRINT #1
130 PRINT #1:CHR$(15);"CONDENSED";CHR$(14);
    "CONDENSED ENLARGED";CHR$(20);CHR$(18)
140 PRINT #1
150 PRINT #1:CHR$(27);"E";"EMPHASIZED";CHR$(27);"F"
160 PRINT #1
170 PRINT #1:CHR$(27);"G";"DOUBLE PRINT";CHR$(27);
    "H"
180 CLOSE #1
190 END

```

Figure 5-3. Different character designation codes produce various type sizes.

each individual dot must be determined in binary format and then translated into decimal format, producing graphics can be a bit tedious. If you have the time and the patience, however, it can be done. Figure 5-4 shows a program that produces a graphic design and the resulting printout.

```

100 RESTORE
110 OPEN #1:"RS232"
120 PRINT #1:CHR$(27)&"K":CHR$(12);CHR$(0);
130 RESTORE
140 FOR I=1 TO 12
150 READ A
160 PRINT #1:CHR$(A);
170 NEXT I
180 DATA 4,10,26,58,103,231,231,103,58,26,10,4
190 FOR X=1 TO 10
200 GOTO 130
210 NEXT X
220 CLOSE #1
230 END

```



Figure 5-4. Here the printer has produced a graphic design.

There are a considerable number of additional commands available for use with your printer, which create a variety of specialized and rarely used effects. All are explained in detail, in the

manual, and no useful purpose would be served by including them here.

Remember that the commands described here are applicable only to the TI-99/4A Impact Printer. Other printers may use different code combinations.

WORD PROCESSING WITH TI-WRITER

There are lots of word processing programs available, with a wide variety of sophistication (and price), but not all can be used with all types of computers. The one designed to function with your TI-99/4A Home Computer is TI-WRITER, produced and marketed by Texas Instruments.

Before discussing TI-WRITER in detail, we should mention some of the shortcomings and limitations that are inherent in the system. First, if you have done a lot of typing on a typewriter with a conventional keyboard, regardless of whether or not you're a touch typist, you'll need to get used to using the computer keyboard to type text because of the awkward positioning of the quotation marks, the question mark, and the asterisk. The upper and lower-case letters are controlled in the conventional manner, using the SHIFT key rather than the ALPHA-LOCK key. It's a matter of becoming accustomed to these features, but it may take a bit of time.

Another drawback is that the screen can display only 40 characters per line at a time. If your left and right margins are set to produce more than that number of characters per line, the screen will keep scrawling from left to right and back again, making it difficult if not impossible to read what you have written. But there is a way to get around this. If you set your left margin tab at 21 and the right margin tab at 60, all your typed lines will be 40 characters wide and it will be easy to read what you have typed. Then, after the document has been completed and has been edited and checked for errors, you can *reformat* the document to the proportions at which you want it printed. Figures 5-5A, B, C on the following pages shows a short document in the version in which it was originally typed, with 40 characters per line and the right margin not justified, as well as several ways in which it was reformatted and then reprinted.

I find the instruction book that comes with TI-WRITER to be rather awkward, and I will take you through a sufficient number of the initial steps to help you become reasonably comfortable. Let me suggest that you sit down at your computer (turned off) and follow me through, step by step, because reading without doing tends to be confusing.

ACTIVATING THE SYSTEM

Before you start it is advisable that you make a copy of the disk provided as part of the TI-WRITER system and then place the original in a safe place. That way you can be sure you always have a good disk to use to make another copy if something untoward happens to the copy disk you'll be using from now on. The original disk

The idea of a word processor is to enable the writer to type on and on without worrying about coming to the end of a line or about having to retype entire pages of copy just because something has to be deleted or some material was accidentally left out. You simply keep on typing and when you find that you need to correct a typographical error you simply type over the old copy and it is automatically replaced by the new. And, better still, you can change the format of your document to any other format when it is time to have it printed. Furthermore, because all your work is saved on disk, you don't end up with a filing cabinet full of carbon copies that tend to clutter up the place.

This copy was originally typed with 40 characters per line, single spaced and not right justified. I then used the text formatter to print it in the different versions that are reproduced here.

Figure 5-5A. The copy was typed originally with up to 40 characters per line.

The idea of a word processor is to enable the writer to type on and on without worrying about coming to the end of a line or about having to retype entire pages of copy just because something has to be deleted or some material was accidentally left out. You simply keep on typing and when you find that you need to correct a typographical error you simply type over the old copy and it is automatically replaced by the new. And, better still, you can change the format of your document to any other format when it is time to have it printed. Furthermore, because all your work is saved on disk, you don't end up with a filing cabinet full of carbon copies that tend to clutter up the place.

This copy was originally typed with 40 characters per line, single spaced and not right justified. I then used the text formatter to print it in the different versions that are reproduced here.

Figure 5-5B. Here it is reformatted to up to 80 characters per line.

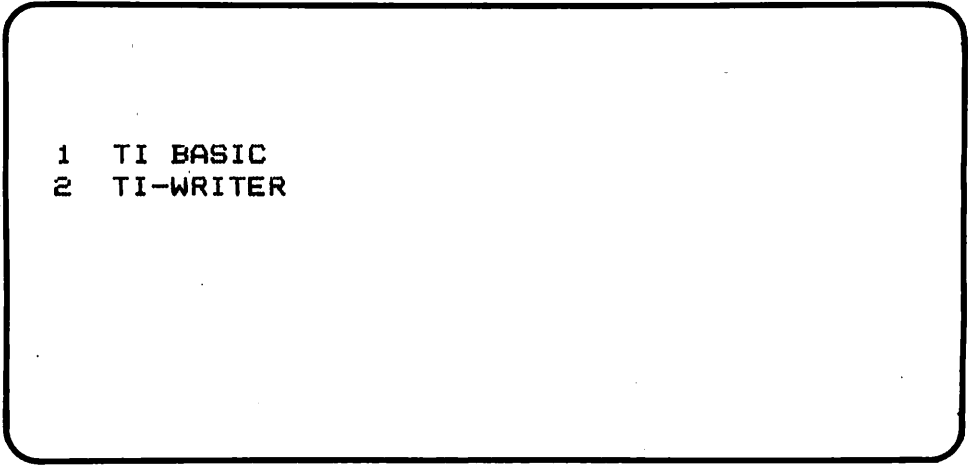
The idea of a word processor is to enable the writer to type on and on without worrying about coming to the end of a line or about having to retype entire pages of copy just because something has to be deleted or some material was accidentally left out. You simply keep on typing and when you find that you need to correct a typographical error you simply type over the old copy and it is automatically replaced by the new. And, better still, you can change the format of your document to any other format when it is time to have it printed. Furthermore, because all your work is saved on disk, you don't end up with a filing cabinet full of carbon copies that tend to clutter up the place. This copy was originally typed with 40 characters per line, single spaced and not right justified. I then used the text formatter to print it in the different versions that are reproduced here.

Figure 5-5C. And here it is changed again, to only 20 characters per line.

contains a total of seven individual programs: EDITA1, EDITA2, FORMA1, FORMA2, PRACTICE, PRACTICE1, and FORMATDOC. There is no need to copy the two practice programs onto your copy disk, as they contain sample texts that you will not be using in the future. (But you can copy them if you want to.) The procedure for copying disk is described in Chapter 2.

Now take the plastic strip that was provided with your word processor and place it in the slot above the keyboard. It will provide a ready reminder of which key combinations produce what effects. Note the red and gray dots. The red dotted line calls for the use of the CTRL key in conjunction with another key. The gray dotted line calls for the use of the FCTN key in combination with another.

Now insert the TI-WRITER module into the computer console and the copy of the TI-WRITER disk into the disk drive. Turn on your peripheral system, then the computer console and monitor. With the usual title screen in display, type any key and you'll be confronted by a menu that gives you the choice of:



```
1  TI BASIC
2  TI-WRITER
```

plus the latter in a whole bunch of different languages Type 2. The screen will display:

```
PRESS
1 FOR TEXT EDITOR
2   TEXT FORMATTER
3   UTILITY
```

We'll ignore the third choice altogether, because it deals with optional peripherals with which we're not concerned. For now, type 1, the display will change to:

```
Edit, Tabs, Files, Lines, Search, RecoverEdit
```

plus some other stuff we won't be using. Most likely your display will be in the form of white letters on a gray background, which may

not be the way you'd like your display to look. Type CTRL 3 several times in succession to observe the different color combinations in which your copy can be displayed and choose the one that seems best to you.

Now set your left and right tabs in the manner discussed above. Type T and press >ENTER< and then use the space bar to move the cursor to the left margin position, which is the 1 after the second blank in the top line and one position to the right of the 2 in the second line. When the cursor is sitting over that spot, type L and then move the cursor to the right margin position, which is the 6 in the second line, and press R and >ENTER<. Your margins are now set to a maximum line length of 40 characters. So far you have been in what is called the *command mode*, which permits you to enter commands rather than text. In this mode the lowercase letter setting does not work. To exit the command mode, type FCTN 9; the display will return to what we had before. Now type E and press >ENTER<. The top line will disappear and the cursor will locate in line 0001, two-thirds of the way to the right of the screen. Now type any copy or, if you can't be bothered to think of what to write, simply press a letter key and hold it down. As you will see, the lines fill one after another, with the letters wrapping around to the next line when the previous one is filled up. If you're typing actual words with spaces between them, then the word wrapping always takes place at the end of a word, so that words are not broken up the way they are when you type past the end of a line without the word processor.

When you have five or six lines filled with some sort of type, try this: Move the cursor, using the arrow keys (FCTN E, S, D, or X), to some place in the middle and, using the space bar, delete a couple of words. You'll see that you're left with an empty hole in the line, because the space bar produces blank spaces and blank spaces are considered characters by the computer. Now move the cursor over some letter and press FCTN 1, and you'll see that the letter disappears and all characters to the right of it on that line have moved one space to the left. Now move the cursor on top of the first character in a word and type FCTN 2; everything to the right of the cursor position will move to the next line down, giving you an opportunity to insert as many new characters or words as you like. Type in something. You now have a pretty messy-looking display, with lines of differing lengths and holes in the lines. Type CTRL 2,

and everything to the right and below the cursor will be reformatted, while text to the left and above the cursor is left unchanged. Now place the cursor over some letter and type another letter. You have replaced the previous letter with the new one.

What we have done up to this point is to use some of the basic and most used editing commands. Now let's take a look at some of the formatting functions. Type FCTN 9 to get back to the command mode and then type T and press >ENTER< in order to display the tab settings once more. Now, with the cursor on the far left tab position, type L and then, using the space bar, move the cursor all the way to the right until it rests on top of the 8. Type R and press >ENTER<, and the cursor will be located at the head of your copy. Now type CTRL 2 and watch how your typed copy is now filling each line with up to 80 characters and, as a result, your document occupies only half the number of lines it did before. Now, using the same procedure described above, change the left tab setting to 50 (the 5 on the second line) and the right setting to 60 and then reformat again (press CTRL 2). The result will be a long, skinny column with up to only 10 characters per line (unless there is a single word that is longer than 10 characters, in which case it will stick out to the right).

Next we might waste a sheet of paper by printing whatever we have on the screen. The command PF (for Print File) prints the material that is currently in display and in the computer memory. Press PF and >ENTER<, which results in this request in the top line:



PRINT FILE, enter devicename:

which asks that you type in the printer identification, which, in the case of the TI-99/4 Impact Printer is RS232. Type it and press

>ENTER< after making sure that your printer is turned on, and the printer will print your document in the form in which it was last displayed.

In most cases you will want to save what you have typed on disk rather than send it directly to the printer. To do that, first remove the TI-WRITER disk from the drive and replace it with a formatted working disk on which there is ample space. The command to use for this purpose is SF, for Save File. When you're satisfied that you want to save what you have typed, type SF and press >ENTER< to produce this display in the top line:



```
SAVE FILE, enter filename:
```

Type in any valid file name, which is the name that must be used in the future to load the file back into the computer or to cause it to be printed. But the file name must include the identification of the disk drive to be used. Here is an example:

DSK1.TEST

If you now press >ENTER< the disk drive will start whirring and after a few moments your precious prose will be preserved forever on the disk. After this, in order to return to the command mode, you must once more replace the working disk with the TI-WRITER disk, though you may continue typing without first changing disks. This

is important, because it is good practice to save your material every 10 minutes or so, just to make sure that a sudden and unexpected power interruption doesn't result in your losing pages and pages of work. Keep saving your work in bits and pieces, using the same file name, and the computer together with the disk drive will replace the previous material with the new version.

You're now ready to go to work. All the other functions and commands, and they are legion, are of only secondary importance. We'll discuss them next, one after another.

FUNCTIONS AND COMMANDS

This long list of commands and functions may at first seem intimidating. Don't let it worry you. For most day-to-day use you won't need to remember more than a few. Furthermore, many functions can be invoked by using either one of two key combinations (apparently as a convenience for touch typists).

Editing Functions

These are the functions that are available when you're operating in the *edit mode*, the mode that is invoked by typing E and that is used to type any kind of document.

Cursor Movements Commands

ARROW KEYS are the four key combinations that can be used to move the cursor to any spot on the screen within the left and right margin limits without blanking out the text across which it is moved. These key combinations include the *repeat* function, which means that holding the keys down will cause the cursor to continue to move in the selected direction. FCTN D or CTRL D moves the cursor to the right. FCTN S or CTRL S moves the cursor to the left. FCTN E or CTRL E moves the cursor up. FCTN X OR CTRL X moves the cursor down.

FCTN 7 or CTRL I moves the cursor to the next *tab* setting to the right. The cursor can also be moved by using the right arrow keys.

CTRL T moves the cursor one tab setting to the left. The left arrow keys can also be used.

CTRL V moves the cursor to the left margin position on the line on which it is currently located.

CTRL L moves the cursor to the upper left-hand corner of the screen without otherwise affecting the display.

CTRL 7 or CTRL W places the cursor over the first letter of the next word on the line on which it is currently positioned.

Insert and Delete Commands

FCTN 1 or CTRL F causes the character on which the cursor is located to be deleted, moving all text to the right of the cursor position one space to the left.

FCTN 2 or CTRL G causes the text at the cursor position to be split, moving everything to the right of the cursor down one line. This permits the insertion of a single character, a word, or entire sentences. After the additional material has been typed in, use the *reformat* function (CTRL 2 or CTRL R) to reorganize the text into the desired format.

CTRL K deletes everything to the right of the current cursor position to the end of the line on which it is located.

FCTN 3 or CTRL N deletes the entire line on which the cursor is located, moving all the text below that line one line up.

FCTN 8 or CTRL O causes a blank line to be inserted above the line on which the cursor is located. It then automatically repositions the cursor on that newly created blank line.

Paragraph Commands

CTRL 4 or CTRL J moves the cursor to the first character in the next paragraph or, if no next paragraph exists, onto the last line of the existing text.

CTRL 6 or CTRL H moves the cursor to the first character of the previous paragraph.

CTRL 8 or CTRL M inserts a carriage return and one blank line and thus causes the next copy to start a new paragraph. A funny little symbol like a tiny "c" and "r" appears on the screen. This symbol is not printed when the text is sent to the line printer.

Miscellaneous Commands

CTRL A or FCTN 4 scrolls the display on the screen 20 lines up, meaning that the 20 lines below the current display are displayed.

CTRL B or FCTN 6 is the opposite. It displays the 20 lines above the current display.

CTRL C or FCTN 9 switches from the *edit mode* to the *command mode*. It can also be used to cancel a command if it is used before >ENTER< has been pressed.

CTRL Y is the left margin release. The cursor must be moved to the left margin before this command is used. It then permits text entry to the left of the left margin.

CTRL 1 or CTRL Z can be used to recover accidentally deleted text, assuming that no other key has been pressed between this and the *delete* command.

CTRL 2 or CTRL R is the *reformat* command. The reformatting action starts at the cursor position and continues downward to the next carriage return (end of paragraph).

CTRL 3 is used to change the colors in which the screen and the text are displayed. It can be used repeatedly to show all available color combinations.

CTROL 5 replaces the line on which the cursor is positioned with the copy on the line above that position. (I can't imagine why.)

CTRL 9 or **CTRL P** tells the line printer to start a new page. It inserts funny little *pa* and *cr* symbols on the screen, which are not printed.

CTRL 0 turns the automatic wordwrap off or, when it is off, turns it back on.

FCTN = enters the *command mode* or, when used with the main menu in display, returns the title screen.

FCTN 5 scrolls the screen window 20 spaces to the right and, when the far right has been reached, it scrolls it back again to the far left.

FCTN 0 is a toggle function that removes or displays the line numbers on the screen.

Formatting Commands

Formatting commands are often called *dot commands*, because they are all preceded by a period. They should, under normal circumstances, be entered at the head of the document that is to be created, but they may also be entered later to change or adjust previously entered instructions.

.AD creates right-margin justification. It must be used in conjunction with the fill command (**.FI**).

.BP causes the printer to start a new page, regardless of the number of lines on the preceding page.

.CE (and number) centers as many lines as are indicated by the number between the left and right margins. When used without a number, it centers the next line only.

.CO (text) causes *text* to be ignored by the printer. The text may not exceed 76 character spaces in length. It can be used to insert comments that are not part of the actual document.

.FI is the *fill* command, which causes as many words as possible to be placed on a line without going beyond the right margin. It must always be used if right-margin justification is desired, or if the document is to be printed in a format that differs from the one in which it was created.

.FO (text) (%) places text at the foot of the page. The percent sign (SHIFT 5) indicates the location of consecutive page numbers, if any.

.HE (text) (%) places header text at the top of the page (line 3). The percent sign indicates the location of consecutive page numbers, if any.

.IN (and number) is used to create an indent in the first line of any paragraph. If the number is used without a + or — sign, the number represents the column number counting from the first column. If it is used with a plus or minus symbol, it is counted relative to the left margin limit.

.LM (and number) sets the left margin to the column position indicated by the number. It may be used anywhere within the document to change the left margin.

.LS (and number) controls line spacing. The default is single spacing. The number causes double spacing, triple spacing, and so on.

.NA turns off right-margin justification. It is the default condition.

.NF turns off the *fill* command (.FI) in order to permit printing portions of a document in the format in which they were originally typed.

.PA (and number) determines the page number with which numbering is to start. It can be used with a + or — sign, in which case the numbering is figured relative to previous page numbers.

.PL (and number) controls the number of lines (including blank lines) that are printed on each page. The default is 66, which is usually too long for the average 8.5-inch by 11-inch page.

.RM (and number) sets the right margin (see .LM).

.SP (and number) inserts as many blank spaces as are represented by the number. When used without a number, it inserts one blank space.

SHIFT 2 (@) overstrikes each word in front of which it is placed. To overstrike more than one word, use the symbol before *each* word.

SHIFT 6 (') can be placed in a space between two words that you want to have appear on one line. The symbol is not printed.

SHIFT 7 (&) underlines the *one* word in front of which it is placed. To underline more than one word, use the symbol before each word to be underlined.

Special Functions

What follows are a number of special functions that are available with your word processing system, but that are probably of rather limited interest to most users. They involve combining various files stored on disk to create personalized form letters for mailing lists or to otherwise permit entering variable information while a document is being printed.

File Management Commands

.IF (and file name) causes the *text formatter* to consider the calling file, the document containing the command, and the file identified by the *file name* as one document. Although it is not possible to have the called file (identified by *file name*) call up a third file, the original document can be used to call up more than one file. Here is an example:

```
.FI
.AD
.LM 4
.RM 62
.PL 55
.HE Garrison & TI &99/4A
.IF DSK1.CHAPTER 1A
.IF DSK1.CHAPTER 1B
.IF DSK1.CHAPTER 1C
```

Here the left margin for all files is set at 4 and the right margin at 62 and the text is right-justified. Each page is to contain 55 lines (including blank lines), and the header will be printed at the top of each page with consecutive page numbers, continuing through all three files, at the top right.

Form Letter Option

Here we are dealing with the creation of form letters that are to be personalized by inserting data contained in another file, or by interrupting the printing process in order to permit us to type in the code that represents the location where variable information is stored.

.ML (and file name) calls the file that was created to contain the variable data to be included in the form letter that is the calling document. If yours is a single-disk-drive system, both, the calling document and the called-up file must be on the same disk.

(number), where the number must be in the range from 1 to 99, defines the variable to be used. The variable may either be identified by number or typed in by the user.

For more detailed information on this option, study the appropriate section in the TI-WRITER Reference Guide (pages 111-113).

WORD PROCESSING WITHOUT A WORD PROCESSOR

If you're one of many who would like to use your computer to take the place of a typewriter, but the amount of writing you do doesn't seem to justify the expense of a word processing program, you do have an alternative. Granted, the method I shall describe is not suitable for long documents, but it can be used for letters, notes, recipes, or other written material of limited length.

You could, of course, simply use the PRINT #1: command before each line of text. Each line would have to be limited to the number of characters that your printer is set up to type per line, plus the line number plus space plus PRINT #1. Assuming that your line numbers consist of three digits and your printer prints 80 characters per line, the total number of characters per line would be 93, which places the last permissible character into the fourth line under the T of PRINT. Although this method is feasible, it's a bit cumbersome, but by writing a simple program, it is possible to simplify the task considerably.

THE NO-WORD PROCESSOR PROGRAM

The program uses DATA. . .READ statements, and allows you to enter as much text as your heart desires or as the RAM in your com-

puter is capable of accepting. The program first displays two blocks of copy that explain how it works. It then displays a menu:

```
1  Enter text
2  Print text
-----
Which?
```

I have included three DATA lines, starting with line 350. The lines use 80 characters per printed line plus nine characters for the line number-space-DATA-space (remember, spaces are counted as characters). If you're using commas in your text, you must enclose the entire text line in quotation marks, which are not counted as characters. If you want to use a left margin of, say, 10 characters, use opening quotation marks, then 10 blank spaces, then as many text characters as you want on the line, and then closing quotation marks. If you want your copy to appear double-spaced, use a comma at either the beginning or the end of the line (outside any quotation marks). The computer then assumes another DATA item, one that is blank, and will thus skip a line in the printout. Two such commas would produce triple spacing, and so on.

Let's look at the program line by line:

Lines 100 and 110 are REMarks.

Lines 120 and 130 clear the screen and send the computer across the three usual subroutines.

Lines 140-160 are the subroutines.

Lines 170-230 place the explanations of the program into display. These lines are optional and can be eliminated if you like.

THE NO-WORD PROCESSOR PROGRAM

```
100 REM NO-WORDPROCESSOR PROGRAM
110 REM TI EXTENDED BASIC
120 CALL CLEAR
130 GOTO 170
140 PRINT "-----" :: RETURN
150 FOR X=1 TO 8 :: PRINT :: NEXT X :: RETURN
160 INPUT "Press >ENTER< ":E$ :: RETURN
170 GOSUB 140
180 PRINT "To enter text, use DATA      lines, limiting text
    entry to as many characters as    your printer will
    print per line." :: PRINT
190 PRINT "Here is a sample for 80      characters
per per line:" :: PRINT :: PRINT
200 PRINT "300 DATA This is the time for all good men to
    come to the aid of their party. This is the" :: GOSUB
    140
210 GOSUB 160 :: CALL CLEAR :: GOSUB 140
220 PRINT "If you're using commas in a line of text, the
    line must be enclosed in quotation marks! They do
    NOT count as characters."
230 GOSUB 140 :: GOSUB 150 :: GOSUB 160 :: CALL CLEAR
240 GOSUB 140
250 PRINT 1;" Enter text" :: PRINT
260 PRINT 2;" Print text" :: GOSUB 140 :: GOSUB 150
270 INPUT "Which? ":WHICH
280 ON WHICH GOTO 350,290
290 OPEN #1:"RS232"
300 ON ERROR 340
310 READ TEXT$
320 PRINT #1:TEXT$
330 GOTO 300
340 CLOSE #1 :: END
350 DATA This is sample text entered by using the
    DATA/READ program for text entry. The
360 DATA printer is set to type 80 characters per line.
    With the line number-space-DATA-
370 DATA space the total number of characters that can be
    used per line number is 89.
```

Lines 240-270 display the menu, asking whether you want to enter text or print it.

Line 280 sends the computer to one of two line numbers, depending on your choice.

Line 290 accesses the line printer.

Line 300 is used to avoid an OUT OF DATA ERROR message when the last text line has been printed.

Line 310 READs one text line at a time.

Line 320 PRINTs that line.

Line 330 sends the computer back to line 300 to repeat the procedure for the next line.

Line 340 disengages the line printer after all text has been printed.

Line 350 and up can be used for text. The DATA lines appear at the end of the program (after END) in order to avoid having to change line numbers to accommodate lots of DATA lines.

I suggest that you save the program without the DATA lines under a file name such as WORDPRO. Then, if you want to save any of the text you have produced, save it under a different file name. That way you won't have to erase a lot of lines when you want to write something else. Later on, when the copy of your material is no longer needed, you can simply erase the file from disk or tape.

If your printer is capable of friction feeding (the TI-99/4A Impact Printer is not), you can print on letterhead and even print addresses on envelopes by arranging your line spacing accordingly.

6

TI BASIC and TI EXTENDED BASIC

TI BASIC and TI EXTENDED BASIC are two languages the TI-99/4A Home Computer can deal with. Although you need not become thoroughly familiar with all the words, phrases, commands, and statements that make up these languages, you will find a reasonably clear explanation of each, along with some examples, in this chapter.

COMMANDS AND STATEMENTS

COMMANDS AND STATEMENTS IN TI BASIC and TI EXTENDED BASIC

All commands and statements in the following list are available in both TI BASIC and TI EXTENDED BASIC. Where use of the command or statement is different in the two versions, the differences are noted. For those of you using TI EXTENDED BASIC, an additional list of commands and statements available only in this version begins on page 102.

ABS(xxx) returns the ABSsolute value, meaning the positive value of any numeric data enclosed in parentheses: ABS (123.4) produces 123.4, but ABS(-123.4) will also produce 123.4. This is useful if you want to determine the difference between two values and it requires fewer program steps to deduct the larger value from the lesser one:

```
10 A=50-100
20 A=ABS(A)
30 PRINT A           50
```

or simpler still:

```
10 A=ABS(50-100)
20 PRINT A           50
```

which illustrates that you can use simple digits, variables, or numeric calculations inside the parentheses in conjunction with the ABS function.

ASC("characters") displays the ASCII character code for the first character of the string enclosed in quotation marks. (I can't say that I have ever found this particularly useful.)

ATN(xxx) returns the arctangent or the measure of the angle, in radians, the tangent of which is represented by xxx. In order to convert radians to degrees, multiply the result by 180/PI. PI is available only in EXTENDED BASIC; in TI BASIC you must type in the values of pi, 3.14159265359.

BREAK may be used by itself or followed by one or several line numbers. When used by itself, it stops program execution when the command is encountered. When used in conjunction with one or several line numbers (separated by commas), it stops program execution at those line numbers. Program execution can be resumed by typing CON. See also UNBREAK.

BYE returns the computer to the title screen and closes all open files. It is preferable to using the QUIT (FCTN=in TI BASIC or SHIFT Q in EXTENDED BASIC) command, because the latter leaves files open, which may result in the loss of recorded data.

CALL is used in conjunction with a long list of subprograms that are part of both TI BASIC and TI EXTENDED BASIC. When the command is encountered, the computer executes the subprogram before going on to the next program line. With TI BASIC the subprograms are CHAR, CLEAR, COLOR, GCHAR, HCHAR, JOYST, KEY, SCREEN, SOUND, and VCHAR; and with TI EXTENDED BASIC the subprograms, in addition to the above, are CHARPAT, CHARSET, COINC, DELSPRITE, DISTANCE, ERR, INIT, LINK, LOAD, LOCATE, MAGNIFY, MOTION, PATTERN, PEEK, POSITION, SAY, SPGET, SPRITE, VERSION. All these subprograms are explained in detail at the end of this chapter (page 107). The CALL statement can be used as an immediate execution command or as a statement in a program. It can also be used to invoke subprograms that you write yourself, but in that case it cannot be used in the immediate mode.

CHR\$(xx) returns the character represented by the ASCII character code number enclosed in the parentheses. It is the reverse of the ASC function.

CLOSE #xx closes a data file with the file number xx that was previously OPENed. Although open files are closed automatically if you edit the program or if you use any of the commands BYE, RUN, NEW, OLD, SAVE, or LIST (*to a device, such as the printer*), it is always good practice to use the CLOSE statement when access to the file is no longer required. Several of the programs reproduced in this book include examples of the use of OPEN and CLOSE.

CONTINUE, abbreviated as CON, restarts program execution after it has been interrupted by a BREAK statement within the program, or by typing SHIFT C in TI EXTENDED BASIC or FCTN 4 in TI BASIC.

COS(xxx) returns the trigonometric cosine of a radian number or a numeric variable representing such a number. In order to obtain the cosine of a number representing degrees, multiply that number by $\text{PI}/180$ ($\text{PI}=3.14159265359$. PI is available in EXTENDED BASIC only.)

DATA makes it possible to store long lists of alpha-numeric data that can be **READ** subsequently when called upon to do so by the program. Individual **DATA** items must be separated by commas, and commas may not be used within such items, unless the entire item is enclosed in quotation marks:

```
100 DATA 1,2,3,4,5,6
200 DATA "10,000",ABC,"ABC Company, Inc."
```

See also the related statements **READ** and **RESTORE**.

DEF is used to permit you to **DEFine** your own function. It must be used in conjunction with function identifiers, followed by an equal sign (=), followed by the desired result:

```
10 DEF SALARY = HOURS*PAY
```

defines the variable **SALARY** as representing the result of multiplying the value assigned to **HOURS** by the value assigned to **PAY** whenever it is encountered in a program.

DELETE, when used in conjunction with the identifier for the device and the name of a file, causes that file to be erased:

```
DELETE "DSK1.TIFILE"
```

in the immediate execution mode, or:

```
10 INPUT "File name?":FN$
20 DELETE FN$
```

in the deferred execution mode.

DIM is used to reserve space in the computer memory for arrays, either numeric or string. It must be used in conjunction with the array name and a numeric expression (integer). The number of such numeric expressions is limited to three in **TI BASIC** and to 10 in **TI EXTENDED BASIC**. For more details on the use of **DIM**, see the explanation of arrays in Chapter 7.

DISPLAY is used differently in the two versions of BASIC. In TI BASIC it is identical to the PRINT statement except that it can only be used to write to the screen and not to any other device. In TI EXTENDED BASIC it can be used in conjunction with AT, BEEP, ERASE ALL, and SIZE to control the placement of data on the screen:

10 DISPLAY AT (10,3):X

places the value represented by X into the tenth row starting at the third column:

10 DISPLAY ERASE ALL:X

causes the screen to be cleared of all data before the value of X is displayed in the left bottom corner of the screen:

10 DISPLAY AT (A,B) SIZE(D)BEEP:A\$

displays the string assigned to the variable A\$ in the row that corresponds to the value assigned to the variable A starting at the column number represented by the variable B, limiting its length in terms of characters to the number represented by the variable D while at the same time sounding a beep (assuming that the volume on the monitor is turned up).

END tells the computer that the end of the program has been reached.

EOF stands for End Of File and is used in conjunction with a file number to determine whether the end of that file has been reached:

10 IF EOF(1) THEN 1000

tells the computer to go to line 1000 upon reaching the last item in file number 1.

EXP is used in conjunction with a numeric expression in parentheses to produce the exponential value:

```
10 X=EXP(15)
```

will produce 3269017.372.

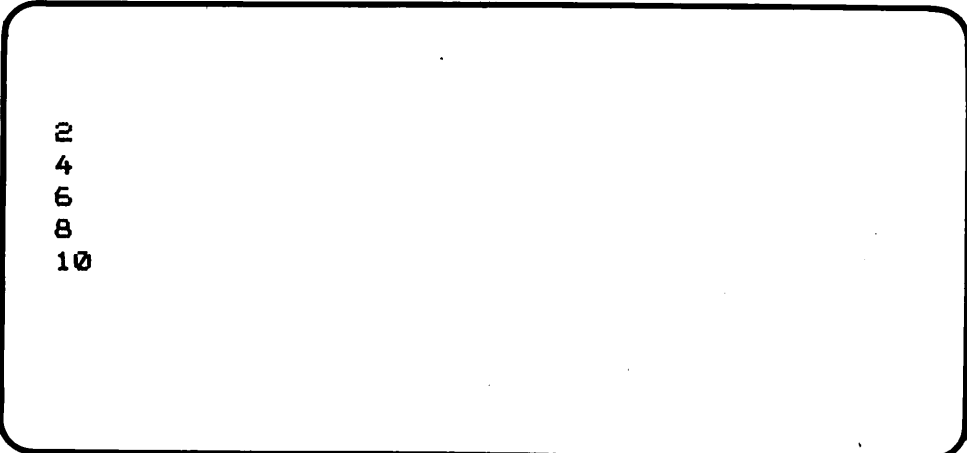
FOR. . .TO (STEP). . .NEXT causes the computer to go around in circles until a certain condition has been met (a *loop*):

```
10 FOR X=1 TO 10
20 PRINT
30 NEXT X
```

will cause 10 blank lines to be placed into display before program execution continues:

```
10 FOR X=1 TO 10 STEP 2
20 PRINT X
30 NEXT X
```

will cause the program to print:



```
2
4
6
8
10
```

before continuing program execution.

```
10 FOR X=1 TO 100
20 READ A$
30 IF A$="SMITH" THEN GOSUB 50 ELSE GOTO 40
40 NEXT X
50 READ B$,C$
60 PRINT B$," ";A$," ";C$
70 RETURN
```

causes the computer to search through up to 100 DATA items until it finds the name SMITH. At that point it goes to a subroutine to READ the strings assigned to B\$ and C\$ and then to print them (JOHN SMITH, CHICAGO), after which it returns to the loop.

GOSUB is used in conjunction with a line number and the RETURN statement to send the computer to a subroutine and then to RETURN it to the line number that follows the GOSUB statement:

```
10 GOSUB 1000
```

sends the computer to line 1000 to execute all following line numbers until a RETURN statement is encountered, after which it goes back to line 20.

GOTO is used in conjunction with a line number to tell the computer to go to that line number and to continue execution from there. The difference between GOSUB and GOTO is that with the latter the computer does not go back to where it came from unless another GOTO statement sends it to another line number.

IF...THEN...ELSE is used to specify an action based on whether or not a condition has been met. The ELSE portion is optional:

```
10 IF A=25 THEN 100
or
10 IF A=25 THEN 100 ELSE 200
or
10 IF A=25 THEN GOSUB 100
or
10 IF A=25 THEN GOSUB 100 ELSE GOSUB 200
```

In the first example the computer is sent to line 100 IF A represents the value of 25; if not, the computer goes on to the next line. In the next example the computer goes to either one of two lines depending on the value of A. In the third example the computer goes to a subroutine if the condition is met and then returns to the next line after 10. In the fourth example the computer goes to either one of two subroutines depending on the value of A.

INPUT halts program execution until data have been keyed in and >ENTER< has been pressed. The statement is used to assign numeric values or string expressions to numeric or string variables:

```
10 INPUT "Your name? ":NAME$
20 INPUT "Your phone number? ":PN or
10 PRINT "Your name?"
20 INPUT NAME$
30 PRINT "Your phone number?"
40 INPUT PN
```

In the first two examples, the prompt line appears along with the INPUT statement, followed by a colon and then the variable. In the last two examples, the prompt is represented by a PRINT statement and the INPUT statement is on a separate line, producing a question mark (?) on the screen to ask you to key in the data to be assigned to the variables.

INPUT (with files) is confusing, because it refers to reading data FROM data files INTO the computer RAM. For more details on statements related to data files, see the data file information in Chapter 7.

INT is used in conjunction with a numeric expression enclosed in parentheses and returns the INTeger of that numeric expression:

```
10 A=INT(123.45)
```

produces 123.

```
10 A=INT(1.99999)
```

produces 1.

```
10 A=INT(123/4)
```

produces 30, although the actual result of this numeric expression is 30.75.

LEN is used in conjunction with a string expression to determine the number of characters in a string:

```
10 PRINT LEN("ABC")
```

prints 3.

```
10 XX=LEN("ABCD")
```

assigns 4 to the variable XX.

```
10 PRINT LEN(" ")
```

prints 1, but

```
10 PRINT LEN(" ")
```

prints 0.

LET is optional. In practice it is nearly always omitted. The purpose of the statment, whether used or omitted, is to assign values to variables:

```
10 X=14
```

assigns 14 to X.

```
10 X=1<3
```

assigns -1 to X, because 1 is indeed smaller than 3.

```
10 X=3<1
```

assigns 0 to X, because 3 is not smaller than 1.

```
10 X$ = "Harry"
```

assigns Harry to X\$.

```
10 X = 10/2
```

assigns 5 to X.

```
10 X = A + B
```

assigns the sum of the values assigned to A and B to X.

LIST causes program lines to be displayed or printed. Used without line numbers it LISTs all program lines from the lowest to the highest. Used with line numbers it lists those lines.

```
LIST
LIST 250
LIST 250 -
LIST -250
LIST 250-500
LIST "RS232"
LIST "RS232" 250-500
```

In the first instance, all program lines are displayed. In the second, line 250 is displayed. In the third, all lines above 250 are shown. Next all lines up to line 250 are shown. In the fifth example, lines 250 through 500 displayed. In the sixth example, all program lines are printed by the line printer. And in the last example, lines 250 through 500 are printed by the line printer.

LOG returns the natural logarithm of a numeric expression as long as that expression is greater than zero:

```
10 A = LOG(15)
```

assigns 2.708050201 to the variable A.

```
10 A=LOG (3*5)
```

does the same.

```
10 A=LOG(X/Y)
```

also does the same if X equals 45 and Y equals 3.

NEW clears the screen and the computer memory of all previously stored data, preparing it to accept a NEW program. Be sure to SAVE anything that you don't want to lose before using NEW.

NEXT See FOR. . .TO.

NUMBER, abbreviated NUM, causes the computer to create line numbers automatically: NUM starts numbering with 100, 110, and so on; NUM 1000 starts with 1000, 1010, and so on; NUM 1000, 100 starts with 1000, 1100, 1200, and so on.

OLD loads a program from disk or cassette into the computer RAM. It must be used with the name of the device and the name of the program to be loaded: OLD "DSK1.PROGRAM" loads the program named PROGRAM from the disk in drive 1 into RAM.

ON GOSUB is used to send the computer to a subroutine if the variable used represents a certain number:

```
10 FOR X=1 TO 5
20 ON X GOSUB 100,200,300,400,500
30 NEXT X
```

sends the computer consecutively to five different subroutines. The value of the variable (X) must start with 1. Thus:

```
10 FOR X=5 TO 10
20 ON X-4 GOSUB 100,200,300,400,500
30 NEXT X
```

would have to be used to make sure that the first value controlling the GOSUB statement is 1.

ON GOTO works the same way as **ON GOSUB**, except that the computer goes to a branch rather than a subroutine.

OPEN opens an already existing data file or creates a new file in conjunction with programs designed to interact with files recorded on disk. For details see the section on *file programs* in Chapter 7.

OPTION BASE 0 or **OPTION BASE 1** sets the lowest acceptable subscript of an array to either zero or one. For more on the subject, see the section on *arrays* in Chapter 7.

POS assigns the position of a certain string character within a string to a numeric variable:

```
10 X=POS("ADAM","A",1)
```

assigns the value of 1 to the variable X, because the computer was told to search the string starting with the first character (1), and the letter A was the first character encountered.

```
10 X=POS("ADAM","A",2)
```

assigns the value of 2 to the variable X, because the computer was told to start the search with the second character (2), and the letter A was the second character found.

```
10 X=POS("ADAM","A",4)
```

assigns the value of 0 to the variable X, because the computer was told to start the search at the fourth character (4), and no letter A was found.

PRINT causes data to be displayed on the screen:

```
10 PRINT 123
```

displays 123

RANDOMIZE is used in conjunction with the RND function to produce an unpredictable sequence of numbers. If it is followed by a numeric expression or a variable representing a numeric expression, the same sequence of numbers will result each time.

READ returns the items listed in DATA lines. See DATA.

REM stands for REMark and is used to enter explanations into programs; all REM lines are ignored by the computer.

RESEQUENCE, abbreviated RES, can be used to change the line numbers throughout a completed and debugged program to regular intervals. Do not use this command unless you're sure that no further editing is needed, because you're likely to have trouble finding what you're looking for. RES can be used with and without line numbers: RES changes line numbers to 100, 110, 120, and so on; RES 1000 changes line numbers to 1000, 1010, 1020, and so on; RES 1000, 100 results in 1000, 1100, 1200, and so on; RES, 100 results in 100, 200, 300, and so on.

RESTORE is usually used to make sure that the items contained in DATA lines are READ from the beginning. But there are, in fact, several ways in which the statement can be used:

```
10 RESTORE
10 RESTORE 250
10 RESTORE #1
10 RESTORE #2,REC 5
10 RESTORE #3,REC A
```

The first example causes items in DATA lines to be READ from the beginning. The next causes them to be READ starting at line 250. The third sets the next record to be the first record in file #1 that will be used with the next PRINT, INPUT, or LINPUT statement. In the fourth example, record number 5 (the sixth record) in file #2 will be used; and in the last example, the record number in file #3 is the number assigned to the variable A.

RETURN See GOSUB, ON GOSUB, and ON ERROR.

RND returns a random number that is greater than zero but less than one, which is why it is nearly always used in conjunction with some type of numeric expression. Unless it is used in combination with RANDOMIZE, it always returns the same number sequence.

```
10 X=INT (RND*16)+1
```

assigns some number between 1 and 16 to the variable X.

```
10 X=INT (RND*(Y-Z+1))+Y
```

produces a random number between the values assigned to the variables Y and Z.

RUN is usually used by itself to start program execution. It can also be used with a line number to start program execution at that line number.

SAVE is used to SAVE a program on disk. It must be used in conjunction with the name of the disk drive to be used and the name under which the program is to be SAVED:

```
SAVE DSK1.PROGRAM
```

simply records the program on the disk in drive 1. When writing programs, especially long ones, it is always a good idea to SAVE them often during the process of program writing in order to avoid inadvertant loss of your work. There are two other ways in which the statement can be used in TI EXTENDED BASIC only:

```
SAVE DSK1.PROGRAM,PROTECTED  
SAVE DSK1.PROGRAM,MERGE
```

In the first example the program called PROGRAM is saved in such a way that it can be RUN, but it cannot be edited, LISTed or SAVED again. In the second example the program is recorded in a manner that permits MERGEing it with a program in the computer RAM. See MERGE, in the listing for TI EXTENDED BASIC.

SEG\$ stands for SEGment of a string. It is used to return a string that consists of a portion of another string:

```
10 A$=SEG$("JOHN DOE",1,4)
```

assigns JOHN to the string variable A\$, because the computer was told to use four characters starting with the first character (1,4).

```
10 A$ = SEG$("JOHN DOE",6,3)
```

assigns DOE to the variable by reading three characters, starting with the sixth (6,3). Remember that blank spaces are counted as characters.

SGN results in 1 if the numeric expression used is positive, 0 if it is zero, and -1 if it is negative. (Seldom used.)

SIN returns the sine of a radian value. To convert degrees to radians, multiply the degrees by $\text{PI}/180$ if you're using TI EXTENDED BASIC, or by $3.14159265359/180$ if you're using TI BASIC.

SQR is used in conjunction with a numeric expression and returns the positive square root:

```
10 X = SQR(9)
20 PRINT X
```

prints 3

STOP terminates program execution. It is interchangeable with **END**, except that it cannot be used after subprograms.

STR\$ converts a numeric expression to a string:

```
10 A$ = STR$(110.7)
```

changes 110.7 into a string expression that cannot be used in calculations. See also **VAL**.

TAB(x) causes data to start at a column determined by the number in parentheses or, in TI EXTENDED BASIC, the value of a numeric variable in parentheses:

```
10 PRINT TAB(8); "DATA"
```

or

```
10 A = 8
```

```
20 PRINT TAB(A); "DATA"
```

causes the word DATA to start in the eighth column.

TAN produces the trigonometric tangent of a radian expression. If the angle is in degrees, the degrees must be multiplied by $\text{PI}/180$ in TI EXTENDED BASIC or the value of $\text{PI}(3.14159265359)$ divided by 180 in TI BASIC.

TRACE displays the program lines along with all other program data when the program is being run. A useful aid when editing.

UNBREAK either removes all BREAKpoints or only those represented by a line-number list.

UNTRACE turns off the TRACE action.

VAL(string) produces the numeric value of a string that consists of digits:

```
10 X = VAL("123")
```

assigns 123 to the numeric variable X.

ADDITIONAL COMMANDS AND STATEMENTS FOR TI EXTENDED BASIC ONLY

The following commands and statements are available only with TI EXTENDED BASIC.

ACCEPT is an alternative to the INPUT command that permits placing the input data in a specific manner and position in the display. The word can be used with a number of options, which may be in any consecutive order. The functions of ACCEPT are best explained by examples:

```
10 ACCEPT AT(8,3):DATA$
```

will place the typed-in data, whether numeric or string, into the eighth row, starting at the third column from the left. DATA\$ or DATA (or any other character or character combination) represents the variable to which the keyed-in data are assigned.

```
10 ACCEPT ERASE ALL:X
```

will clear the screen of all previously displayed data and assign the keyed-in data to the variable X while displaying the value of X at the bottom left corner of the screen.

```
10 ACCEPT VALIDATE("YN"):Y$
```

limits the characters that can be typed in and assigned to the string variable Y\$ to Y or N. Typing any other character causes the computer to go on to the next line without assigning anything to the variable Y\$.

```
10 ACCEPT AT(A,B)SIZE(C)BEEP VALIDATE(DIGIT,"XYZ"):W$
```

combines a number of the available options. It is assumed that A and B are numeric variables, where A represents the row number and B the column number where data will be displayed. The SIZE option limits the number of characters that may be entered to the numeric value assigned to C. The BEEP option causes a beep to sound (assuming that the volume on the monitor is turned up) before data may be typed in. The VALIDATE option limits the characters that may be entered to digits or the letters X, Y, and Z with the keyed-in data assigned to the string variable W\$.

The options available for use in conjunction with VALIDATE, which limits the type of input, are UALPHA, which permits all uppercase letters; DIGIT, which permits digits only; NUMERIC, which

permits digits and +, -, E, comma, and decimal point; and *string*, which permits the characters that make up the string and are enclosed in quotation marks. When any of these options is used in conjunction with **VALIDATE**, it must be enclosed in parentheses.

DISPLAY USING specifies the format in which a numeric expression is to be displayed. In most instances it is used to limit the number of displayed decimals. For example:

```
10 DISPLAY AT(A,B):USING "###.##":X
```

limits the number of decimals to two and the number of digits that can be used to the left of the decimal point to three. Thus, if the variable X represents 123.4567, the displayed figure will be 123.45.

IMAGE is used in conjunction with a string to control the way numbers are displayed on the screen or printed by the line printer. The use of **IMAGE** is very complicated and only rarely used; if you're interested, check the details in the TI EXTENDED BASIC manual.

LINPUT permits the assignment of an entire line or record to a single string variable. This is another seldom used statement; you can consult the manual for details.

MAX displays the greater of two numeric expressions or assigns it to a numeric variable. For example:

```
PRINT MAX (5,10)
```

displays 10, and

```
10 X=MAX(Y,Z)
```

assigns the greater of the two values to X. Similarly,

```
10 X=MAX (-15,-36)
```

assigns -15 to X.

MERGE permits you to MERGE a program already recorded on disk with a program that is currently in the computer RAM. Be sure that the program lines used in the already-recorded program are not the same as those used in the program currently in RAM: Otherwise, the new line numbers will replace the old. This command can be used only with disks, and in order to work, the previous program must have been SAVED using the MERGE option:

SAVE "DSK1.PROGRAM",MERGE

from which it is then recalled and MERGED with the current program by using:

MERGE DSK1.PROGRAM

MIN is the opposite of MAX.

ON BREAK can be used in two ways: ON BREAK STOP and ON BREAK NEXT, where the first has the same effect as BREAK whereas the second causes the computer to ignore a BREAK statement, assuming that that statement is used in conjunction with a line number.

100 BREAK 250

ON ERROR determines what the computer does when an ERROR message is encountered: ON ERROR STOP stops program execution; ON ERROR 2000 sends the computer to line 2000, which must be the beginning of a subroutine that ends with the RETURN statement.

ON WARNING can be used in three ways:

ON WARNING PRINT
ON WARNING STOP
ON WARNING NEXT

The first causes the warning message to be displayed, after which the program execution continues. The second prints the message

and halts program execution. The third ignores the message and continues program execution.

PI stands for 3.14159265359. For instance, the following figures the volume of a ball with a radius of 3:

```
10 BALL=PI*(4/3)*4^3
```

assigns 268.0825731 to the variable BALL.

PRINT USING limits the number of digits that will be displayed:

```
10 PRINT USING "The total is $ ####.##":6543:1234
```

will result in:

```
The total is $6543.12
```

REC must be used in conjunction with a file number in parentheses, and it returns the number of the RECOrd that will be accessed in an OPENed file at the next use of a PRINT, INPUT, or LINPUT statement. Since records in files are numbered starting with zero, number 3 is actually the fourth one.

RPTS causes repetitions of a string, with the number of repetitions controlled by a numeric expression. For example:

```
10 A$=RPT$("HENRY",3)
```

results in:

```
10 A$=RPT$("HENRY",3)  
HenryHenryHenry
```

SIZE, used in the immediate mode, displays the number of unused bytes; if the Memory Expansion Peripheral is attached, it returns the available bytes in the stack and in memory. **SIZE** produces either:

```
11840 BYTES FREE
```

```
11840 BYTES FREE
```

```
224488 BYTES OR PROGRAM SPACE FREE
```

with the numbers depending on the amount of program currently in RAM.

SUB must be used as the first statement in a user-written subprogram. For details consult your manual.

SUBEND is used to mark the end of a subprogram.

SUBEXIT is used to exit a subprogram before its end.

SUBPROGRAMS

Both versions of BASIC include a number of built-in subprograms that can be **CALL**ed by using the **CALL** command along with the name of the subprogram. In this section we look at these programs in some detail, though I suggest that you also consult your manuals to be sure you have a clear understanding of the functions and input requirements of those subprograms you intend to use.

The subprograms are divided into two sections. The first section deals with those that are available in both versions of BASIC, and the second section deals with those that are available only in TI EXTENDED BASIC.

Several of the subprograms dealing with the graphics capability of the computer refer to SPRITES. Sprites are graphic characters that can be manipulated in many different ways. Because the subject is of considerable interest, especially to younger computer users, I have devoted to the subject a large chunk of a separate chapter, in which all of the sprite-related subprograms are explained in detail and sample programs illustrate how they function.

Some programs listed in the program sections of this book make use of subprograms and provide examples of how they can best be utilized.

SUBPROGRAMS AVAILABLE WITH TI BASIC AND TI EXTENDED BASIC

The following subprograms are available for both versions of TI BASIC. A list of additional subprograms available only for TI EXTENDED BASIC begin on page 122.

CALL CHAR must be used in conjunction with a character code and a pattern identifier, with the latter in quotation marks and both together in parentheses. For example:

```
CALL CHAR(32,"FFFFFFFFFFFFFFFF")
```

converts the ASCII character code (32) that actually stands for a blank space to a graphic character represented by those 16 Fs. All graphic character identifiers must consist of 16 characters, and that can get a bit confusing. The reason is this: The character position on the screen is made up of 64 dots, which are arranged in an eight-by-eight grid (see Figure 6-1). Each character in the string expression is used to control the pattern of dots that make up the block, and these characters must be in the hexadecimal code, where 0 1 2 3 4 5 6 7 8 9 A B C D E F, represent the numbers from 0 through 15. I have found it very time-consuming and dif-

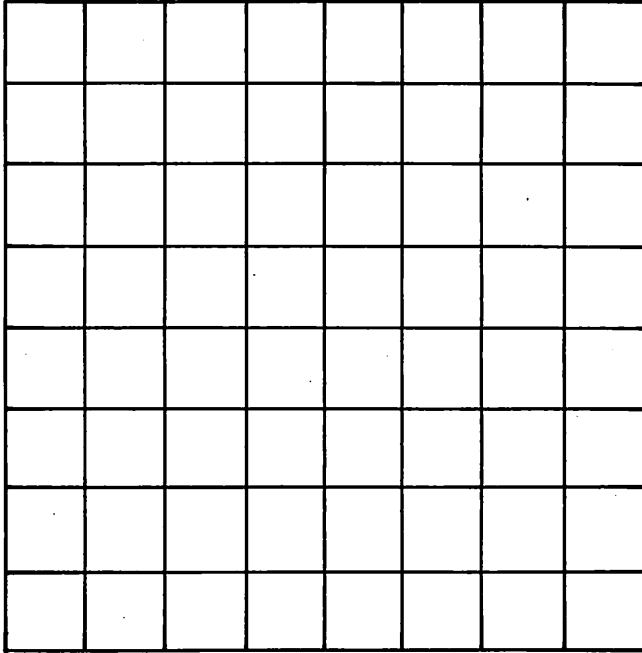


Figure 6-1. Characters are composed of an eight-by-eight grid of dots.

difficult to figure out the appropriate number-character combinations that will produce certain results. For that reason it might be a good idea to produce a subroutine that assigns various frequently used patterns to a string variable that can then be used with all manner of graphic programs:

```
1000 BLOCKS$ = "FFFFFFFFFFFFFFFF"
1010 TRIANGLE$ = "0103070F1F3F7FFF"
1020 SQUARE$ = "FF818181818181FF"
```

and so on. Then, instead of having to type all those 16 characters each time you use the CALL CHAR statement, you can replace it with the string variable:

```
10 CALL CHAR(33,SQUARE$)
```

which is likely to avoid a lot of typing errors. Figure 6-2 (see page 110) shows how the block of 64 dot positions is divided into two

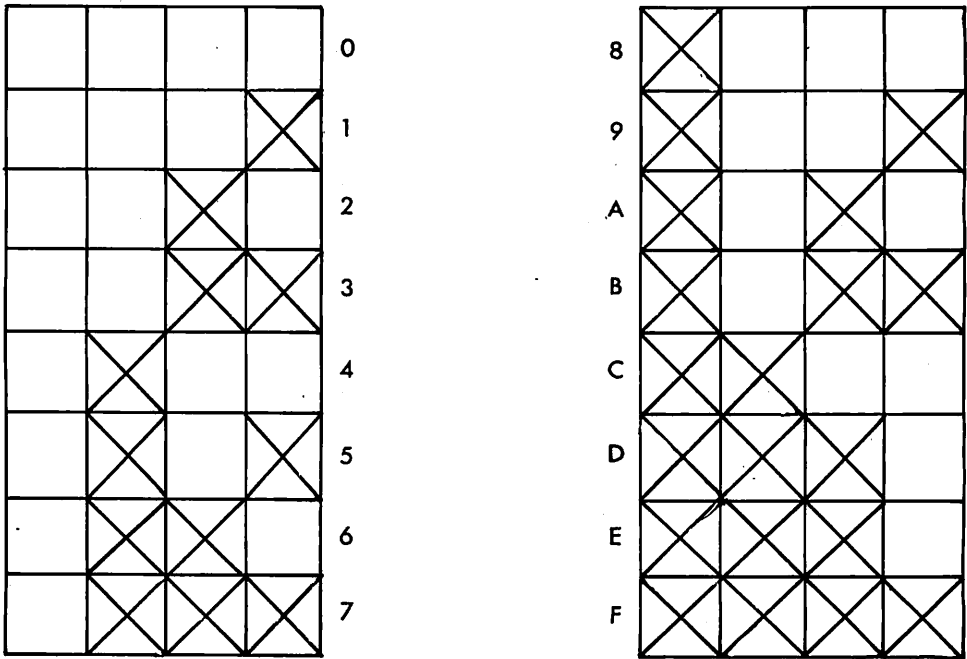


Figure 6-2. The 64-dot block is divided into two halves, with one hexadecimal character representing each line of four dots.

halves, each consisting of four horizontal and two vertical positions. Each group of four horizontal positions is controlled by one of those 16 hexadecimal characters. Thus, it takes two of those characters to control each horizontal line of eight dot positions. The characters that produce the different available combinations of black (on) and white (off) dots are shown in the illustration. Figure 6-3 shows a number of likely patterns and the 16-character combinations that will produce these patterns.

The CALL CHAR subprogram, by itself, only defines the shapes to be used. In order to cause those shapes to be displayed, we have

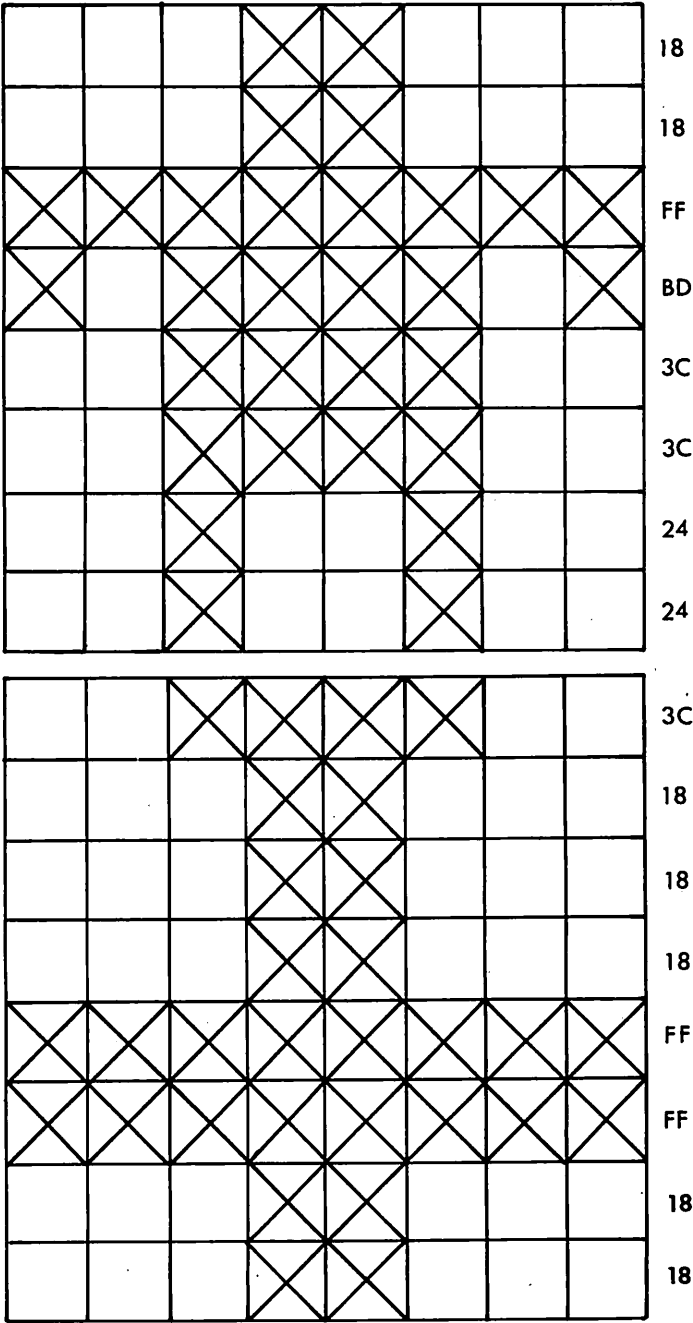


Figure 6-3. Patterns created by using 16 hexadecimal numbers.

to use some of the other graphic subprograms that are designed to tell the computer where to display the shape and what color combinations to use. Before we go into that, type this short program just for the fun of it:

```
100  A=5
110  B=5
120  GOSUB 230
130  CALL CLEAR
140  FOR X=1 TO 15
150  CALL CHAR(33, BLOCK$)
160  CALL CHAR(33, SQUARE$)
170  CALL COLOR(1,9,6)
180  CALL VCHAR(A,B,33)
190  A=A+1
200  B=B+1
210  NEXT X
220  GOTO 220
230  SQUARE$="FF818181818181FF"
240  BLOCK$="FFFFFFFFFFFFFFF"
250  RETURN
```

Now type RUN and watch what happens. A staircase develops, consisting alternately of solid blocks and open squares.

CALL CLEAR may be used in the immediate and deferred modes. Either way, it CLEARS the screen, leaving the prompt and cursor in the bottom left-hand corner. It should be used with some frequency when developing a program in order to keep the screen reasonably uncluttered and easy to read. Here is a good way to achieve a cosmetically attractive display in either version of BASIC:

```
10 CALL CLEAR
20 PRINT TAB(8); "Program title"
30 FOR X=1 TO 10
40 PRINT
50 NEXT X
60 INPUT "Press >ENTER< ":E$
```

will place the program title more or less centered on the screen with the prompt to press >ENTER< displayed on the left side of the bottom line. The CALL CLEAR command fills the screen with the ASCII character 32. If that character is defined to represent something else, such as some sort of graphic symbol, it will print that symbol instead (see Figure 6-4). For instance:

```
10 CALL CHAR(32,"OFOFOFOFOFOFOFOF"  
20 CALL CLEAR  
30 GOTO 30
```

will fill the screen with black and white (or, if you're in color, black and varied colored) vertical lines until you press FCTN 4 (CLEAR) to stop the program.

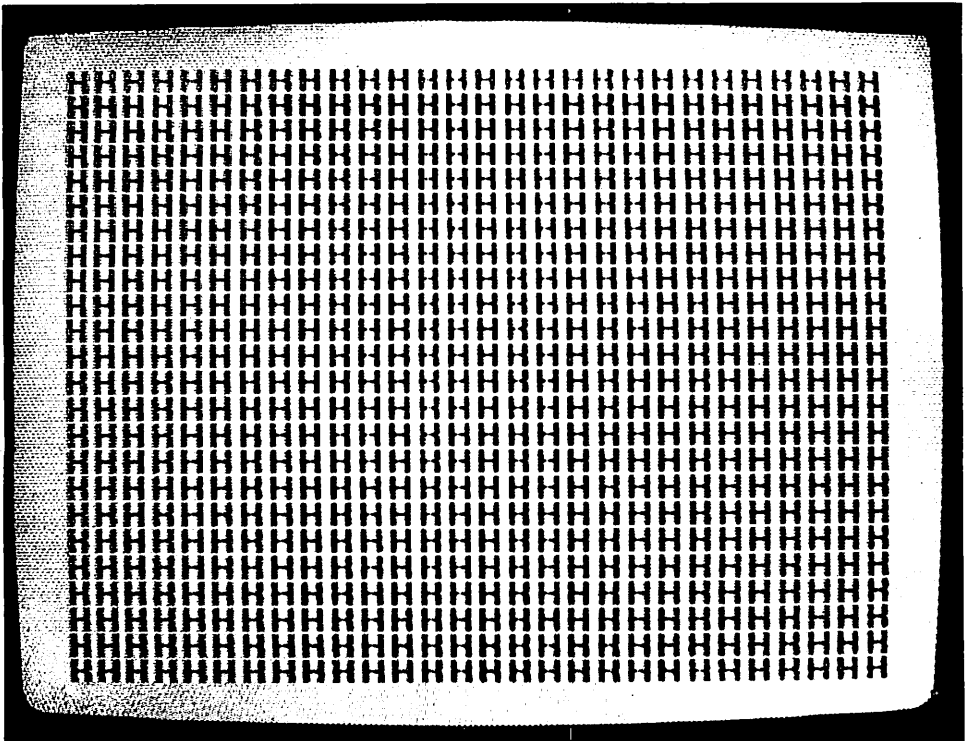


Figure 6-4. If the ASCII character code 32 is redefined as representing a letter or graphics symbol, CALL CLEAR fills the screen with that symbol.

CALL COLOR is the subprogram that determines the colors that will be used when the graphic shapes are displayed on the screen. The command is used with three parameters, enclosed in parentheses:

CALL COLOR (1,16,14)

where the first number is referred to as the *character-set number*, meaning that the character to be displayed falls into one of 16 groups of ASCII character codes:

1	32-39
2	40-47
3	48-55
4	56-63
5	64-71
6	72-79
7	80-87
8	88-95
9	96-103
10	104-111
11	112-119
12	120-127
13	128-135
14	136-143
15	144-151
16	152-159

(To determine which characters fall into each of those groups of seven ASCII character codes, see the list of ASCII character codes in Appendix I.) The second number in parentheses is the *foreground color*, the color in which the selected character itself will be displayed. The third number is the *background color*. Each character appears, theoretically at least, on a square block of color. That color is the background color. If transparent is selected, then the character appears with an invisible background, meaning that it is displayed against the color of the screen. (For control over screen color, see CALL SCREEN). To see how all of this works, and in order to be able to do a bit of experimentation, type this short program:

```
100 GOSUB 150
110 CALL CLEAR
120 CALL COLOR(A,B,C)
130 CALL VCHAR(D,E,F,G)
140 GOTO 140
150 INPUT "CHARACTER SET # ? " :A
160 INPUT "FOREGROUND COLOR? " :B
170 INPUT "BACKGROUND COLOR? " :C
180 INPUT "PLACE ON ROW # ? " :D
190 INPUT "AT COLUMN # ? " :E
200 INPUT "USE CHARACTER # ? " :F
210 INPUT "REPEAT HOW OFTEN? " :G
220 RETURN
```

By typing different numbers and RUNning the program over and over, you can find out what color combinations work for you, and which ones are less than satisfactory. Here is the list of available colors and the number code associated with each:

- 1 transparent
 - 2 black
 - 3 medium green
 - 4 light green
 - 5 dark blue
 - 6 light blue
 - 7 dark red
 - 8 cyan
 - 9 medium red
 - 10 light red
 - 11 dark yellow
 - 12 light yellow
 - 13 dark green
 - 14 magenta
 - 15 gray
 - 16 white
-

CALL GCHAR is a subprogram of limited usefulness. It is used with three parameters in parentheses; as:

```
CALL GCHAR(10,15,X)
```

and causes the ASCII character code of the character displayed in the tenth row in column number 15 to be assigned to the numeric variable X. If you then add a PRINT X command, that ASCII character code will be displayed on the screen.

CALL HCHAR and **CALL VCHAR** are two subprograms that control the horizontal and vertical position in which a selected character is displayed on the screen. Unless one or both of these commands are used, no display results. Both commands are used in conjunction with either three or four numeric expressions or variables representing those expressions:

```
CALL HCHAR(A,B,C,D)  
CALL VCHAR(A,B,C,D)
```

where A is the row number, B is the column number, C is the ASCII character code, and the optional D represents the number of times you want the character repeated, either horizontally or vertically or both. Here is a short program designed to demonstrate the effect of these two commands:

```
100 GOSUB 180
110 CALL CLEAR
120 CALL SCREEN(S)
130 CALL COLOR(CS,FG,BG)
140 CALL HCHAR(A,B,C,D)
150 CALL VCHAR(E,F,G,H)
160 INPUT "PRESS >ENTER< ":E$
170 GOTO 100
180 INPUT "HORIZONTAL ROW? ":A
190 INPUT "HORIZONTAL COL.? ":B
200 INPUT "ASCII CHAR. CODE? ":C
210 INPUT "REPEAT HOW OFTEN? ":D
220 INPUT "VERTICAL ROW? ":E
230 INPUT "VERTICAL COL.? ":F
240 INPUT "ASCII CHAR. CODE? ":G
250 INPUT "REPEAT HOW OFTEN? ":H
260 INPUT "SCREEN COLOR? ":S
270 INPUT "CHARACTER SET? ":CS
280 INPUT "FOREGR. COLOR? ":FG
290 INPUT "BACKGR. COLOR? ":BG
300 RETURN
```

RUN the program a number of times, entering different values for all the variables, and watch what happens. It's the best way to get a feel for how all this works.

CALL JOYST causes input to the computer to be controlled by the Wired Remote Controller, commonly referred to as the JOYSTick. It can be used with commercially available game programs and with certain types of graphic or sound programs that you might write yourself. Since it must be used in conjunction with other subprograms that we haven't discussed yet, we'll get back to it when those associated subprograms are discussed. Suffice it to say for now that the optional unit consists of two controllers, each with a movable handle (joystick) and a red bar referred to as the fire button. The unit is plugged into the nine-prong receptacle on the *left* side of the keyboard console (not in the back, where there is an identical outlet), and it can be attached at any time without affecting programs or other data in memory.

CALL KEY is a complicated and difficult-to-understand program that assigns certain codes to a variable, depending on which key has been pressed. To get an idea, try this:

```
10 CALL KEY(0,A,B)
20 IF B=0 THEN 10
30 PRINT A,B
40 GOTO 10
```

If you now type RUN, absolutely nothing happens because the variable B records whether a key has been pressed. If no key has been pressed, the value of B remains zero. Now type A B C D E F; the display will respond with:

65	1
66	1
67	1
68	1
69	1
70	1

where the numbers on the left side of the screen represent the ASCII codes for the letters you typed, having been assigned, one after another, to the variable A. The 1's on the right represent the number of key strokes that you made since the last time the computer encountered the CALL KEY statement. The statement can also be used to split the keyboard into two nearly identical halves, which can then be used by two persons to control the movement of characters on the screen. It can also be used in conjunction with CALL JOYST to affect the effect of pressing the fire button. There are other uses, such as converting the keyboard to PASCAL. For

more details, consult the TI BASIC User's Reference Guide and some of the sample programs included with the SOUND and GRAPHICS commands.

CALL SCREEN works similarly to **CALL COLOR**, except that only one number or variable is required in parentheses; as in:

CALL SCREEN(X),

representing the color you want the screen to be. The same color set shown earlier is available for this command. You might want to change the above program by adding two lines:

```
115 CALL SCREEN(S)
215 INPUT "SCREEN COLOR? ":S
```

in order to be able to experiment with different screen colors in combination with the previously used colors.

CALL SOUND permits the use of the computer to produce sounds and, if used by someone with a degree of musical talent, actually to play melodies. (Be sure that the volume on the monitor is turned up.) The statement is used in conjunction with three or more numeric expressions or numeric variables. Try this as a demonstration:

```
10 INPUT "DURATION? ":D
20 INPUT "FREQUENCY? ":F
30 INPUT "VOLUME? ":V
40 CALL SOUND(A,B,C)
50 GOTO 10
```

For the inputs you can use 1 through 4250 for DURATION, where every 1000 represents about 1 second. For FREQUENCY you can use any number from 110 to 44733 for actual tones or -1 through -8 for different types of noises. For VOLUME you can use any number from 0 to 30, where 0 is the loudest and 30 the quietest. After you have experimented in order to learn the effects of different values for each of the three categories, you might try this:

```
10  TONE=110
20  FOR SCALE=1 TO 38
30  CALL SOUND (500,TONE,0)
40  TONE=TONE+55
50  NEXT SCALE
60  END
```

When you type RUN, the speaker plays a series of 38 ascending notes.

The CALL JOYST command can be used in conjunction with SOUND programs. A program in the little booklet that comes with the joysticks permits you to use the Remote Control Device to control the sounds you hear. I think it's more trouble than it's worth, but it might give you some ideas about how to use this combination of functions:

```
100 A$="CDEFGABC"
110 DIM NOTE(9),M(8,8)
120 FOR I=1 TO 9
130 READ NOTE(I)
140 NEXT I
150 DATA 262, 294, 330, 349, 392, 440,
      494, 524, 40000
160 FOR I=1 TO 9
170 READ X,Y,INDEX
180 M(X,Y)=INDEX
190 NEXT I
200 DATA 8,4,1,8,8,2,4,8,3
210 DATA 0,8,4,0,4,5,0,0,6
220 DATA 4,0,7,8,0,8,4,4,9
230 CALL JOYST(1,X1,Y1)
240 CALL JOYST(2,X2,Y2)
250 X1=X1+4
260 Y1=Y1+4
270 X2=X2+4
280 Y2=Y2+4
290 CALL SOUND(-1000,NOTE(M(X1,Y1),0),NOTE(M
      (X2,Y2),0)
300 PRINT SEG$(A$,M(X1,Y1),1);SEG$(A$,M(X2,
      Y2),1
310 GOTO 230
```

When this program is RUN, nothing happens until you operate the joysticks. Both can be used simultaneously to produce harmony (or disharmony). In line 100 a string of the available names of the notes to be played is assigned to the string variable A\$, which is later broken down into its segments (using SEG\$) in order, in line 300, to display the note(s) being played on the screen. In lines 120-150, the frequencies of the C scale are assigned to an array called NOTE. Another array, called M, is established in lines 160-220, using the subscripts 0-8. Nine values, 1-9, are READ and stored in M at locations that correspond to the positions of the joysticks, and are then incremented by 4. These values are then used in line 290 to select

the proper frequency from the NOTE array. Lines 230 and 240 cause the joysticks to be activated. The X and Y values are adjusted in lines 250–280 to correspond to locations in the M array. Line 290 causes the tones to be generated, and line 300 displays the names of the notes on the screen. Line 310 sends the computer back to line 230 to permit you to play as long as you like. The SOUND and the GRAPHICS programs appear to be rather difficult to explain which may be why it is hard to understand the explanations in the different instruction manuals. A number of programs that make use of these subprograms are included in the program section, and the line-by-line explanations there will probably be more useful than a lot of complicated theory.

A list of the numbers associated with musical tone frequencies is contained in both the TI BASIC and the TI EXTENDED BASIC manuals.

SUBPROGRAMS AVAILABLE IN TI EXTENDED BASIC ONLY

CALL CHARPAT is similar to CALL GCHAR. It is used with two parameters in parentheses, the first of which is the character code that was assigned to a character created by the CALL CHAR program. The second must be a string variable designed to receive the 16-character code that was used to create that character. Thus, if that character was a solid square block, the string variable will represent "FFFFFFFFFFFFFFFF", which can be PRINTed if you so desire.

CALL CHARSET simply restores the standard characters and standard colors, negating previous commands to the contrary. It is used by itself with no parameters.

CALL COINC stands for COINCide. It is a program that is used in conjunction with CALL MOTION and similar subprograms discussed below. It is designed to detect an impending or actual collision between several sprites; some examples of its use can be found in Chapter 11.

CALL DELSPRITE deletes a previously created sprite. It is used with one or several numeric expressions or numeric variables that represent sprites, or with ALL to delete all sprites:

```
CALL DELSPRITE(#1,#5)  
CALL DELSPRITE(ALL)
```

CALL DISTANCE is used in conjunction with two sprite numbers or one sprite number and a row/column identifier plus a numeric variable in parentheses. It returns the square of the distance between two sprites or between one sprite and a spot on the screen.

CALL ERR is used in conjunction with either two or four numeric variables in parentheses, representing the *error code*, the *error type*, the *error severity*, and the *line number* of the last error to have been encountered. It is used in conjunction with ON ERROR.

CALL INIT is used along with LINK, LOAD, and PEEK to access assembly language subprograms. It prepares the computer and the Memory Expansion unit. For details see the manual.

CALL LINK is used in combination with INIT, LOAD, and PEEK to access assembly language subprograms. See the manual for details.

CALL LOAD See LINK and INIT.

CALL LOCATE causes one or several sprites to be LOCATED at a specific row/column position on the screen. It is used in conjunction with sprite number(s) and row/column parameters in parentheses.

CALL MAGNIFY, used with the magnification factor in parentheses, causes sprites to be magnified. The available magnification factors are 1, 2, 3, and 4.

CALL MOTION is used in conjunction with a sprite number and numeric variables representing row and column velocities, in parentheses. In terms of row velocity, a positive number moves the sprite down and a negative number moves it up. In terms of column

velocity, a positive number moves the sprite to the right, and a negative number moves it to the left. Combinations of the two can be used to move sprites at any angle.

CALL PATTERN is designed to permit changing the character pattern of a sprite without affecting any of its other characteristics. It is used in conjunction with the number of the sprite and the character value, in parentheses.

CALL PEEK See INIT, LINK, and LOAD, as well as the manual.

CALL POSITION, used with sprite number(s) and numeric variables in parentheses, assigns the row/column position of a named sprite to the numeric variables.

CALL SAY is used to activate the speech synthesizer. It is used with a word string in parentheses, where the word(s) must be included in the available word list. For more details, see Chapter 8.

CALL SPGET can be used to assign word sounds to string variables. For details, see Chapter 8.

CALL SPRITE is the program that creates all those SPRITEs we've been talking about. A sprite is a graphic design that has a certain color and can be made to appear anywhere on the screen. Sprites must be numbered from 1 to 28 in order to be manipulated. For details, see Chapter 11.

CALL VERSION, used with a numeric variable in parentheses, assigns a number that represents the version of BASIC that is currently in use (100 represents EXTENDED BASIC).

7

A Primer to Personal Program Writing

In chapter 3, we wrote two simple programs in order to get a bit of hands-on experience with the keyboard and to observe the effects of a few commands and statements. In this chapter we'll gradually go several steps further to explore some of the more sophisticated approaches to program writing and to look at some of the tricks and shortcuts that are available to us.

Before we begin, there is one important distinction we must make between writing programs in TI BASIC and in TI EXTENDED BASIC. In TI BASIC every statement must be on a separate line, whereas in TI EXTENDED BASIC it is possible to group several statements on one line, separating them by double colons (::). To illustrate this difference, here are two versions of an identical sequence of program steps:

```
100 FOR X=1 TO 10  
110 PRINT  
120 NEXT X
```

is the form in which this sequence of three statements, constituting a loop, would have to be written in TI BASIC.

100 FOR X=1 TO 10::PRINT::NEXT X

is the same series of statements in TI EXTENDED BASIC. Actually, it makes no difference which format you use, unless you want your program listing to be printed by a line printer. In that case you may want to avoid excessively long printouts by grouping statements on single lines (the maximum number of characters per line, including blank spaces, is 140). On the other hand, it is often a lot easier to figure out what a program is about if there are not too many statements on the same line, not to mention that editing and debugging tends to become more difficult. For that reason, many of the programs in this book use separate lines for every statement despite the fact that they could have been reproduced more concisely by combining statements.

In order to write any program, we must first have a clear idea of what we want our program to accomplish. Many authors like to emphasize the importance of developing some graphic representation, known as a *flowchart*, to aid in creating a visual "blueprint" for the organization of the program. Personally, I find that flowcharts are sometimes more trouble than they are worth, and I never use them. But that's my way of doing things. You may feel different.

Basically, programs fall into two primary categories. Some are designed to store data, representing for all practical purposes an electronic file cabinet or card file. The other category consists of programs to perform calculations and, thus, provide answers to (usually) mathematical questions. Within those two categories there are various subcategories. Some programs simply require you to type in certain data. After that they calculate and then produce an answer. Others are designed to activate the printer and to print out selected data. Still others interface with the disk drive(s) and record data, which then becomes available the next time that program is executed. In addition, certain programs include decision-making statements that cause results to be based on whether or not a certain condition has been met. Let's look at a few examples.

A SIMPLE DATA-BASE PROGRAM

We'll begin with a simple data-base program. (We've written this one in TI EXTENDED BASIC to save space. If you have only TI

A SIMPLE DATA BASE PROGRAM

```
100 CALL CLEAR:GOSUB 1000
110 PRINT TAB(5);"This is a simple":PRINT
120 PRINT TAB(5);"Data-Base program.":GOSUB 1000
130 GOSUB 1100
140 INPUT "Press >ENTER< ":E$
1000 PRINT "-----":RETURN
1100 FOR X=1 TO 10:PRINT:NEXT X:RETURN
```

BASIC you will have to reformat the program to put each statement on a separate line before you can run it. Try it—it will be good practice.)

Up to this point the program causes its two-line title to be displayed more or less in the center of the screen between two dashed lines with the prompt "Press >ENTER<" in the bottom left corner. In line 100 the screen is cleared and the computer is sent to a subroutine (line 1000) that prints a dashed line. We use a subroutine rather than a direct command here because this dashed line is used many times to make the display cosmetically more attractive, and it saves a lot of typing to be able simply to go to the subroutine. In line 110 the TAB(5); statement causes the text line to start in the fifth column, thus centering it on the screen. The PRINT at the end of the line causes a blank line to be inserted between the two text lines. Line 120 is similar to the previous line except for the GOSUB 1000 statement at the end, which causes another dashed line to be displayed. In line 130 the computer is sent to another subroutine, one that is also used frequently. It places 10 blank lines on the screen, positioning the two lines of copy along with the two dashed lines screen center. Line 140 is used to stop program execution until you press >ENTER<, because otherwise the computer would go right on and the two lines of text would disappear before they could be read. Lines 1000 and 1100 are the two subroutines we've already discussed. Figure 7-1 on page 128 shows what the screen looks like at this point.

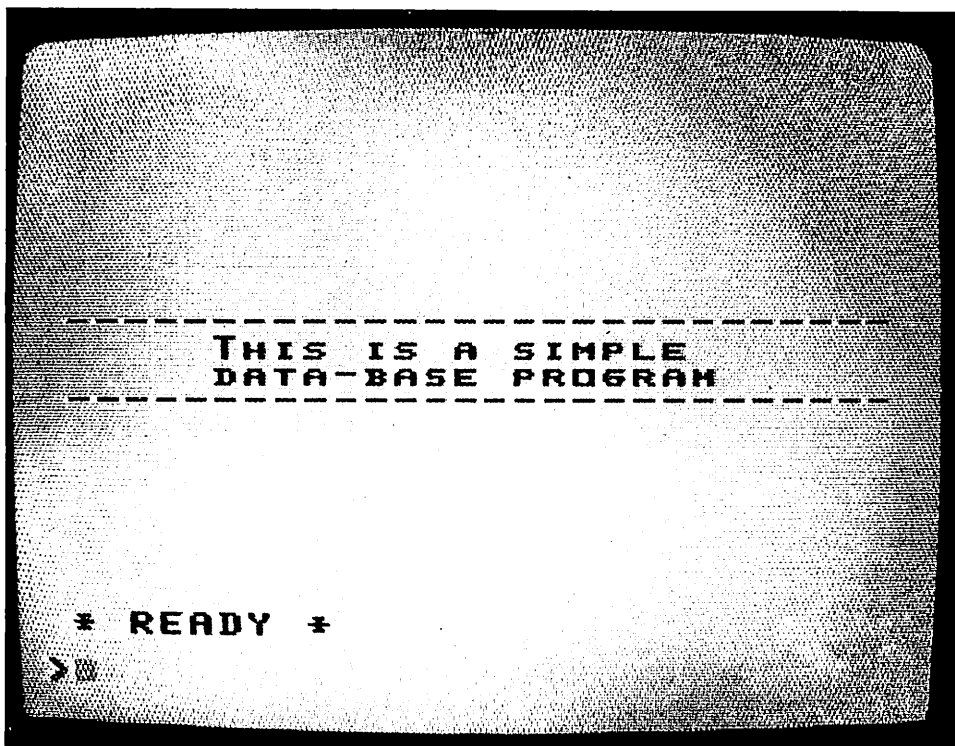


Figure 7-1. The display at this stage of our simple data-base program.

Now let's go on and write another portion of the program.

```
145 CALL CLEAR
150 GOSUB 1000::PRINT "To find data associated"::PRINT
160 PRINT "with a product, type"::PRINT
170 PRINT "the product number"::GOSUB 1000
180 PRINT "Which product number?":GOSUB 1100
190 INPUT PN
```

This time we ask the computer to display instructions to the user. In line 145 we make sure that the last line of the previous display is erased before new text is displayed. We use a line number ending on 5 because this line was entered after the subsequent lines had already been typed. Line 150 first uses the dashed-line subroutine and then prints the first line of text, adding a blank line

for readability. Lines 160 and 170 print the next two lines and then add another dashed line. Line 180 asks you to determine which product number you're interested in, and it uses the other subroutine to place the four lines of copy into the center of the screen. Line 190 places a question mark (?) into the bottom left corner of the screen and causes the computer to stop program execution until you type in a number.

```

200 INPUT "Printout? (Y/N) ":YN$
210 FOR Y=1 TO 50
220 RESTORE::A=A+1::READ N,PR$,Q,RU,LP
230 IF N=PN THEN 300
240 IF A=50 THEN END
250 NEXT Y
300 CALL CLEAR::GOSUB 1000
310 PRINT "Product number . . . . .";N
320 PRINT "Description . . . . .";PR$
330 PRINT "Quantity on hand . . .";Q
340 PRINT "Reorder number . . . . .";RO
350 PRINT "List price . . . . .";LP::GOSUB 1000::
PRINT::PRINT
360 IF YN$="Y" THEN 400 ELSE 370
370 INPUT "Another product or quit? (P/Q) ":PQ$
380 IF PQ$="P" THEN 150 ELSE END
400 OPEN #1:"RS232"
410 PRINT #1:"Product number....";N
420 PRINT #1:"Description .....";PR$
430 PRINT #1:"Quantity on hand . . .";Q
440 PRINT #1:"Reorder number....";RO
450 PRINT #1:"List price.....$";LP
460 PRINT #1:"-----":PRINT #1
470 CLOSE #1
480 GOTO 370

```

This time we've used a whole bunch of new statements. Let's look at them line by line. Line 200 asks if you want the data sent to the line printer. Line 210 starts a loop that is limited to 50 turns, because we're dealing with 50 (or fewer) product numbers. If the program must accommodate more than 50 products, the number will have to be increased accordingly. In line 220 the RESTORE statement is used to make sure that the items in the DATA block (which we have yet to type in) are READ from the beginning. Then the value of A is increased by 1 during each turn through the loop

and next the computer is told to READ five items that are contained in the DATA block. Line 230 checks if the value assigned to the numeric variable N is equal to the value that you typed in and that was assigned to the numeric variable PN, in which case the computer is told to go to line 300. If the values assigned to the two numeric variables don't match, the computer goes to line 240, which checks the value of the numeric variable A to see if all products have been READ. If they have, the computer is told that the END of the program has been reached. If not, the computer goes to line 250, which causes it to repeat the loop.

Lines 300–360 are the section of the program that causes the READ data for the selected product number to be displayed between two dashed lines; the two PRINTs at the end of line 350 move the display up from the bottom of the screen. Line 360 asks if you want the results sent to the line printer, in which case the computer is told to go to line 400. Otherwise it is told to go to line 370 to determine whether you want to exit the program or select another product number. Depending on the answer, it then returns to line 150 to permit you to select another product number or simply quits the program. Line 400 activates the line printer. Lines 410–450 print the data at the line printer. Line 460 prints a dashed line and adds two blank lines in order to separate this group of data from the next. Line 470 deactivates the line printer, and line 480 sends the computer to line 370 to determine whether you want to continue or quit.

Now the only thing left to do is to type in the product data:

```
500 DATA 1, PRODUCT A, 300, 250, 75.95
510 DATA 2, PRODUCT B, 200, 250, 15.75
520 DATA 3, PRODUCT C, 175, 150, 88.77
```

and so on for up to 50 products. The first item represents the product number, the second the product description, the third the quantity on hand, the fourth the reorder level, and the fifth the list price. When all entries have been made and you have tested the program to make sure it functions the way it is supposed to, you might type RES for RESEQUENCE and >ENTER< and the computer will

automatically change the line numbers to start at 100 and continue at intervals of 10, adjusting the various GOSUB, GOTO, and IF . . THEN . . ELSE line numbers automatically. Figure 7-2 shows a printout of the program listing after the line numbers have been adjusted and Figure 7-3 is a printout of two products that resulted when the printout option was selected.

```

100 CALL CLEAR :: GOSUB 480
110 PRINT TAB(5); "This is a simple" :: PRINT
120 PRINT TAB(5); "Data-Base program" :: GOSUB 480
130 GOSUB 490
140 INPUT "Press >ENTER< ":E$
150 CALL CLEAR
160 GOSUB 480 :: PRINT "To find data associated"
    :: PRINT
170 PRINT "with a product, type" :: PRINT
180 PRINT "the product number" :: GOSUB 480
190 PRINT "Which product number?" :: GOSUB 490
200 INPUT PN
210 INPUT "Printout? (Y/N) ":YN$
220 FOR Y=1 TO 50
230 RESTORE :: A=A+1 :: READ N,PR$,Q,RO,LP
240 IF N=PN THEN 270
250 IF A=50 THEN END
260 NEXT Y
270 CALL CLEAR :: GOSUB 480
280 PRINT "Product number....";N
290 PRINT "Description.....";PR$
300 PRINT "Quantity on hand..";Q
310 PRINT "Reorder number....";RO
320 PRINT "List price.....$";LP :: GOSUB 480 ::
    PRINT :: PRINT
330 IF YN$="Y" THEN 360 ELSE 340
340 INPUT "Another product or quit? (P/Q) ":PQ$
350 IF PQ$="P" THEN 160 ELSE END
360 OPEN #1:"RS232"
370 PRINT #1:"Product number....";N
380 PRINT #1:"Description.....";PR$
390 PRINT #1:"Quantity on hand..";Q
400 PRINT #1:"Reorder number....";RO
410 PRINT #1:"List price.....$";LP
420 PRINT #1:"-----" :: PRINT
    #1

```

(continued)

```
430 CLOSE #1
440 GOTO 340
450 DATA 1,PRODUCT A,300,250,75.95
460 DATA 2,PRODUCT B,200,250,15.75
470 DATA 3,PRODUCT C,175,150,88.77
480 PRINT "-----" :: RETURN
490 FOR X=1 TO 10 :: PRINT :: NEXT X :: RETURN
```

Figure 7-2. The listing simple data-base program after the lines have been renumbered using the RESEQUENCE command.

```
Product number.... 1
Description.....PRODUCT A
Quantity on hand.. 300
Reorder number.... 250
List price.....$ 75.95
-----

Product number.... 2
Description.....PRODUCT B
Quantity on hand.. 200
Reorder number.... 250
List price.....$ 15.75
-----

Product number.... 1
Description.....PRODUCT A
Quantity on hand.. 300
Reorder number.... 250
List price.....$ 75.95
-----
```

Figure 7-3. A sample printout resulting from running our simple data-base program.

There are many ways in which a data-base program like this can be refined, and more sophisticated versions are included in the program section. The principles, however, remain the same.

A SIZE/PRICE COMPARISON PROGRAM

Now let's look at another type of program, one that performs some quite different functions. This sample program designed to accept a number of inputs and then perform some simple calculations in order to produce a desired result. Let's say, for the moment, that we want to determine whether the "large economy size" is actually more economical than regular or smaller sizes of a certain product, such as soap powder. (Again, we've combined statements to save space; if you do not have TI EXTENDED BASIC you will have to put each statement on a separate line if you want to run the program.)

A SIZE/PRICE COMPARISON PROGRAM

```

100 CALL CLEAR::GOSUB 1000
110 PRINT TAB(5);"This program compares sizes":PRINT
120 PRINT TAB(5);"and prices of products.":GOSUB 1000
130 GOSUB1200:GOSUB 1100
140 CALL CLEAR::INPUT "Number of units in size 1?":U1::
    PRINT
150 INPUT "Price of size 1?          $":P1::PRINT
160 INPUT "Number of units in size 2? ":U2::PRINT
170 INPUT "Price of size 2?          ":P2
180 CALL CLEAR
190 UP1=P1/U1::UP2=P2/U2
200 IF UP1<UP2 THEN 210 ELSE 240
210 PER=(UP2-UP1)/UP2*100::PER=INT(PER*10+.5)/10
220 PRINT "Size 1 is ";PER;"% cheaper":PRINT
230 GOSUB 1100::CALL CLEAR::GOTO 300
240 PER=(UP1-UP2)/UP2*100::PER=INT(PER*10+.5)/10
250 PRINT "Size 2 is ";PER;"% cheaper":PRINT
260 GOSUB 1100::CALL CLEAR
300 INPUT "Another calculation? (Y/N)  ":YN$
310 CALL CLEAR
320 IF YN$="Y" THEN 140 ELSE END
1000 PRINT "-----"
    ::RETURN
1100 PRINT::INPUT "Press >ENTER< ":E$:RETURN
1200 FOR X=1 TO 10::PRINT::NEXT X:RETURN

```

This program calculates the percentage difference in price between two sizes of the same product or, for that matter, two versions of a product produced by different manufacturers. Lines 100–130 print the purpose of the program, using cosmetic subroutines in the same manner as we did before. Lines 140–170 ask you to enter the number of units (ounces, pounds, or whatever) in each package, and the two prices. Line 190 determines the price per individual unit for both packages, assigning the result to the numeric variables UP1 and UP2. In line 200 the computer checks which package has the larger unit price, and if the unit price of package 1 is smaller than the other, the computer is told to go to line 210, otherwise (ELSE) to line 240. Line 210 deducts the unit price for package 1 from that of package 2, divides the result by the unit price of package 2, and multiplies the result by 100 to arrive at a percentage. It then uses an often-used standard formula to round the result off to one decimal. The formula is:

$$X = \text{INT}(X * 10 + .5) / 10 \text{ or } X = \text{INT}(X * 100 + .5) / 100$$

where the number of zeros (10, 100, etc.) determines the number of displayed decimals. Line 220 displays the result, and line 230 tells the computer to go to line 300 to find out whether you want to run another calculation or quit. Lines 240–260 are a repeat of the previous lines, used if package 2 proves to be the cheaper of the two. Line 300 asks if you want to repeat, and line 320 tells the computer where to go based on the answer. Lines 1000–1200 are subroutines.

STANDARD FORMULAS

Time Calculations

In the previous program we used a formula for rounding off numeric values to a predetermined number of decimals. (I could have used the PRINT USING command available with EXTENDED BASIC, but for most purposes I find the formula method easier.) There are a number of other standard formulas that are also used fairly often to make the computer perform certain functions. One has to do with the performance of calculations using hours, min-

utes, and seconds. Since minutes and seconds are not decimal values, they must be converted to decimals for this purpose and then reconverted. The manner of accomplishing this depends on the format in which the time values are entered into the computer. The two most common are:

```
100 INPUT "Hours?      ":H
110 INPUT "Minutes?    ":M
120 INPUT "Seconds?    ":S
```

and

```
100 INPUT "Time (H.MMSS) ":T
```

In the first version we leave the hour figure alone but convert the minutes and seconds to decimal values like this:

```
130 M=M / .6 / 100::S=S / .6 / 10000
140 TIME=H+M+S
```

in which case the numeric variable TIME becomes the decimal equivalent of the three time values that were entered separately. After the calculation has been performed, the time value must be reconverted:

```
200 TIME1=INT(TIME)::TIME2=(TIME-TIME1)*100
210 MIN=INT(TIME2)
220 SEC=(TIME2-MIN)*100*.6
230 MIN=MIN*.6
240 PRINT H;" hours, ";MIN;" minutes and ";SEC;" seconds"
```

which produces the conventional values for hours, minutes, and seconds.

If the second type of input is used, a value such as 2 hours, 30 minutes, and 30 seconds would be entered as 2.3030, which will have to be separated into its component parts before it can be converted:

```
110 H=INT(T)::M1=(T-H)*100 / .6::M=INT(M1)
120 S=(M1-M)*100 / .6::M=M / 6
130 TIME=H+(M / 100)+(S / 10000)
```

This results in the decimal equivalent of the original time value, ready to be used for calculations. To reconvert it to conventional time values, use the same formula that was described above.

Compound Interest

Another formula frequently used in programs that deal with financial matters determines compound interest:

$$FV = PV * (1 + (INT / 100)) ^ CP$$

where FV stands for FUTURE VALUE, PV for PRESENT VALUE, INT for INTEREST, and CP for the number of COMPOUNDING PERIODS. In order to work correctly, the annual interest rate must be adjusted to the interest rate representing the compounding periods. For instance, if the annual interest rate is 7.45% and the interest is compounded daily, then the value of INT in the formula is:

$$INT = 7.45 / 365.25$$

because each year, in view of the fact that each fourth year is a leap year, has 365.25 days.

Temperature Versus Altitude

Here is a formula that is primarily of interest to the pilots among my readers. Temperature fluctuations affect aircraft performance, a fact that is of particular importance during the landing and take-off phases of flight. Aircraft need longer runways on hot days than they do on cold days, and the climb capability is affected by air temperature. The phenomenon is referred to in aviation as the effect of *density altitude*, which refers to the theoretical elevation or altitude adjusted for the prevailing temperature conditions. Here is a short program that uses the standard formula to perform that adjustment:

```
100 INPUT "Actual elevation or altitude? ";ALT
110 INPUT "Temperature in degrees F.?      ";TF
120 TC=(TF-32)/1.8
130 DALT=(145426*(1-(((288.15-ALT*.001981)/288.15), 5.2563/
  ((273.15+TC)/288.15)) .235))
140 DALT=INT(DALT)
150 PRINT "Density altitude= ";DALT;" feet.":END
```

Here line 120 uses a standard formula to convert degrees Fahrenheit to degrees Celsius and line 130 uses a long and complicated formula to convert actual elevation or altitude to density altitude. Although this conversion may be of limited interest to many readers, it does illustrate the fact that we always need to know the correct mathematical formulas to use whenever we expect the computer to perform some type of calculation.

FILE PROGRAMS

Now let's get back to some more programming. Many times it is desirable that our programs retain previously input or calculated information for future use, so that you don't have to key in previous totals or data. Programs that perform this function are referred to as *file programs*, because they create separate files that are used to store these data for future use. There are two types of such file programs available to us. One is referred to as *sequential* and the other as *random access* (which, for some strange reason, is referred to as *RELATIVE* in TI BASIC). Sequential files, when accessed, read data from the beginning, one after the other. Random-access files, when accessed, permit the computer to go directly to a given record that consists of the desired data.

In order to make use of this function, we must have at least one disk drive on line, because the data are stored directly on disk and are not retained as part of the original program. (A cassette tape recorder may also be used.) The commands and statements used to write to or access the data files are OPEN #1, PRINT #1, INPUT #1,

and CLOSE #1, with the number representing the number of the file to be used. The OPEN #1 command must be used in conjunction with the DISK NAME and the FILE NAME:

100 OPEN #1:"DSK.TIONE.FILE NAME"

and its purpose is to OPEN the desired file and thus get it ready to accept further commands. The PRINT #1 command must be used with the numeric or string variable that represents the data you want to preserve by recording them on the disk:

110 PRINT #1:A

or

PRINT #1:A\$

which sends the value assigned to the numeric variable A or the string assigned to the string variable A\$ to the disk. The opposite is INPUT #1, which is used in the same format, but retrieves data that were previously recorded and places them into the computer's memory, ready to be displayed on the screen, sent to the line printer, or otherwise manipulated. The CLOSE #1 statement is used by itself, and it must always be used to CLOSE the file when no further data are to be recorded or retrieved. You can perform both recording and retrieval between one pair of OPEN / CLOSE commands, but I believe that it is better practice to CLOSE the file after one type of action is finished and to reOPEN it before the next function is performed. Here is a simple sample (using combined statement lines):

```
100 CALL CLEAR::A$="BEGIN"
110 INPUT "First run? (Y/N) ":FR$
120 IF FR$="Y" THEN 130 ELSE 200
130 INPUT "Last total? ":LT
140 OPEN #1:"DSK.TIONE.TOTALS"
150 PRINT #1:LT
```

(continued)

```
160 CLOSE #1::IF A$="END" THEN END
200 OPEN #1:"DSK.TIONE.TOTALS"
210 INPUT #1:LT
220 CLOSE #1
230 PRINT "The last total was      ":LT::PRINT::PRINT
240 PRINT "Do you want to":PRINT
250 PRINT 1,"add?":PRINT 2,"deduct?":PRINT
260 INPUT "Which?      ":WHICH
270 ON WHICH GOSUB 300,400
280 A$="END":GOTO 140
300 INPUT "Add how much?      ":ADD
310 LT=LT+ADD::RETURN
400 INPUT "Deduct how much?   ":DED
410 LT=LT-DED::RETURN
```

In the first line we assign the string BEGIN to the string variable A\$, because we need to use it later on in lines 160 and 280. Lines 110 and 120 are needed because, if this is the first run, a new file must be created because no data file called TOTALS exists as yet. If this is the first run, you're asked to enter the last total (of anything) in line 130, and lines 140-160 then create a file and record the value assigned to the numeric variable LT. In line 160 the computer also checks on the string that is assigned to A\$ and, since the answer is negative, the computer goes to the next line, reOPENing the file, reading the value assigned to LT, closing the file, and then, in line 230, printing that value. Lines 240-260 ask whether you want to add or deduct, and line 270 sends the computer to one of two line numbers. Line 280 assigns a new string, END, to the string variable A\$, and sends the computer back to line 140 to again reOPEN the file and record the results produced in lines 300-410 in the data file. Lines 300 and 310 are used if you want to add to the last total, and lines 400 and 410 come into play if you want to deduct from the last total. In each case the computer is sent back to line 280.

Although this little program may not be of much practical use, it clearly illustrates the basics of the file management function. There are, of course, more complicated ways to deal with files, but most of these are beyond the needs of the average home programmer.

What we just created is a *sequential* data file. Whenever no additional instructions are given, the computer automatically assumes *sequential* as the type of file. It also assumes that the

lengths of the various records that will be stored in the file may be *variable* in size. If you want your file to be of the random-access type, the statement must include the word **RELATIVE**, in which case the automatic assumption is that all records will be of the same length, though you may add the statement **FIXED** as reminder. You may also include a numeric expression to stipulate the maximum number of characters in each record:

OPEN #1:"DISK.TIONE.FILE NAME",RELATIVE,FIXED 20

The convention used for **RELATIVE** files is explained in some detail (including several available options) in the User's Reference Guide. The trouble is that the use of **RELATIVE** (random-access) files is considerably more involved in TI BASIC than in other BASIC dialects; unless you insist on becoming proficient at programming such files, I would suggest you stick to the simpler version we have described. The advantage of random-access files is that they operate faster, but since the difference can be measured in fractions of seconds, it is of little practical consequence.

Furthermore, sequential files can be used to retrieve data in random-access fashion simply by asking the computer to retrieve the values assigned to a certain variable. The computer then reads the data stored in the file from the beginning until it encounters the indicated variable, which is then retrieved and moved into the computer memory for display or manipulation.

ARRAYS

In this last section of this chapter we'll look at one of the more obscure, but also more useful, functions available to us. The function is the use of arrays, and it requires a bit of explanation. Put succinctly, *arrays* are collections of variables arranged in a way that allows easy use in computer programs.

Array Demonstration Programs

Arrays are nearly always used in combination with FOR...NEXT loops, and the best way to demonstrate how the function works is to start by explaining how data are entered into an array. Array Demonstration Program #1 is a short program that describes one way in which numerical data might be entered.

ARRAY DEMONSTRATION PROGRAM #1

A numeric array containing 10 numbers is created using a FOR...NEXT loop.

```
100 CALL CLEAR
110 FOR X=1 TO 10
120 READ A(X)
130 NEXT X
140 DATA 10,9,8,7,6,5,4,3,2,1
150 INPUT "SUBSCRIPT ":S
160 PRINT
170 PRINT TAB(12);A(S)
180 B(1)=A(S)*3
190 PRINT
200 PRINT TAB(12);B(1)
210 PRINT
220 GOTO 150
```

Line 110 sets up the loop that will make 10 passes before continuing with program execution.

Line 120 causes the computer to READ the numeric data in the DATA line (140), assigning each to the numeric variable A and the subscript (in parentheses) of the value of X. When all 10 passes through the loop are completed, the 10 numbers will have been assigned to the numeric variables as follows:

A(1)=10	A(2)=9	A(3)=8	A(4)=7	A(5)=6
A(6)=5	A(7)=4	A(8)=3	A(9)=2	A(10)=1

Line 150 asks you to input a subscript number from 1 to 10.

Line 170 causes the value assigned to A(S) to be displayed.

Line 180 multiplies the value assigned to A(S) by 3 and assigns the result to the numeric variable B(1).

Line 200 displays the result, and line 220 starts the process all over again.

As you can see, if we had not used the array, we would have to assign the 10 values to 10 different numeric variables, using up a lot of programming steps and thus slowing program execution. But arrays can also be used to store strings, using more or less the same routine. Array Demonstration Program #2 is a short program that demonstrates how this works. Here we're using the array to contain five names.

ARRAY DEMONSTRATION PROGRAM #2

A string array created in the same manner as in Figure 7-4.

```
100 CALL CLEAR
110 FOR X=1 TO 5
120 READ A$(X)
130 NEXT X
140 DATA JOHN DOE,MARY SMITH,FRANK JONES,PAUL GARRISON,
    JANE BROWN
150 INPUT "SUBSCRIPT? ":S
160 PRINT
170 PRINT A$(S)
180 PRINT
190 GOTO 150
```

Line 110 sets up a loop that will make five passes through the loop.

Line 120 causes the computer to READ the strings contained in the DATA line (140), assigning each to the string variable A\$ and the subscript X in parentheses.

Line 150 asks you to key in a subscript number from 1 to 5.

Line 170 displays the string represented by the string variable A\$(S), and line 190 goes back to the beginning.

Here we would have had to assign the five names to five different string variables, which wouldn't have been too bad. But if the DATA block had contained 50 or 500 names, the array capability would have simplified matters considerably.

In Array Demonstration Program #3 we use a somewhat different way of entering data into an array. Instead of using data stored in a DATA block, we use the numbers generated by the loop itself.

ARRAY DEMONSTRATION PROGRAM #3

A FOR . . . NEXT loop generates the numeric data for the array.

```
100 CALL CLEAR
110 FOR X=10 TO 100 STEP 10
120 M(X/10)=X
130 NEXT X
140 INPUT "SUBSCRIPT? " : S
150 PRINT
160 PRINT M(S)
170 PRINT
180 GOTO 140
```

Line 110 sets up a loop that assigns the numbers from 10 to 100 to the numeric variable X in increments of 10.

Line 120 assigns the value of X to the numeric variable M with the subscript (X / 10) in order to have the subscript numbers start with 1.

Line 140 asks you to type in a subscript number from 1 to 10. Line 160 displays the value assigned to M(S), which will be 10, 20, 30, . . . ,100.

Array Demonstration Program #4 found on the next page uses still another method of entering data into arrays. Here, instead of using a FOR . . . NEXT loop, we create a loop using the GOTO statement, permitting us to enter an unlimited number of arbitrary variables into the two arrays.

ARRAY DEMONSTRATION PROGRAM #4

Using a GOTO statement, an unlimited number of numeric and string data can be entered into the two arrays.

```
100 CALL CLEAR
110 INPUT "NUMBER? ":N0
120 PRINT
130 INPUT "STRING? ":ST$
140 PRINT
150 A=A+1
160 B=B+1
170 ARR1(A)=N0
180 ARR2$(B)=ST$
190 PRINT ARR1(A)
200 PRINT
210 PRINT ARR2$(B)
220 PRINT
230 INPUT "PRESS >ENTER< ":E$
240 GOTO 100
```

Line 110 asks you to key in a number that is assigned to a numeric variable.

Line 130 asks you to key in a string, assigning it to a string variable.

Lines 150 and 160 increase the values of the numeric variables A and B by 1 each time the lines are encountered.

Lines 170 and 180 set up two arrays that accept the previously input data.

Lines 190 and 210 display the variables assigned to the two arrays.

Line 240 returns you to the beginning to enter additional data.

Array Demonstration Program #5 is a short program that demonstrates the practical use of arrays, showing quite clearly how they can be useful in effectively shortening and, thus, speeding up a program. Here we have three sets of data. The first is a list of eight product descriptions. The second shows the list prices of these products. And the third lists the manufacturing cost for each product. In the conventional way we would have ended up with eight string variables and 16 numeric variables. The purpose of the program is to compute the changes in manufacturing cost and list price, assuming that there is an X% increase in material and/or

labor cost. This would have meant a total of 16 calculations with the results assigned to an additional 16 numeric variables. By using three arrays, all this is changed.

ARRAY DEMONSTRATION PROGRAM #5

A short program in TI EXTENDED BASIC that makes practical use of three arrays.

```

100 REM ARRAY SAMPLE PROGRAM
110 REM TI EXTENDED BASIC
120 CALL CLEAR
130 INPUT "Percent change? ":PER :: PER=PER/100
140 RESTORE
150 OPTION BASE 1
160 OPEN #1:"RS232"
170 FOR P=1 TO 8 :: READ P$(P):: NEXT P
180 FOR L=1 TO 8 :: READ LL(L):: NEXT L
190 FOR C=1 TO 8 :: READ CC(C):: NEXT C
200 FOR X=1 TO 8 :: PRINT #1:P$(X);LL(X);TAB(20);CC(X)::
    PRINT #1 :: NEXT X
210 IF E$="END" THEN END ELSE 220
220 FOR L=1 TO 8 :: LL(L)=LL(L)+LL(L)*PER :: NEXT L
230 FOR C=1 TO 8 :: CC(C)=CC(C)+CC(C)*PER :: NEXT C
240 PRINT #1:"-----"
    "
250 E$="END" :: GOTO 200
260 DATA PRODUCT A,PRODUCT B,PRODUCT C,PRODUCT D,PRODUCT
    E,PRODUCT F,PRODUCT G,PRODUCT H
270 DATA 100,56.34,11.06,97.35,56.93,25,75.11,5.85
280 DATA 55,29,6,49,30,13,38,3

```

Line 130 asks you to key in the percentage increase in manufacturing cost, assigning the figure to the numeric variable PER and then dividing that figure by 100 for use in the subsequent calculations.

Line 140 assures that the items in the DATA block are READ from the top.

Line 150 sets the lowest subscript number in the arrays to 1. If this statement is left out, the lowest number is always 0.

Line 160 accesses the line printer.

Lines 170-190 use three FOR...NEXT loops to create three arrays: P\$(P) for the product descriptions, LL(L) for the list prices, and CC(C) for the manufacturing cost figures.

Line 200 uses another FOR. . .NEXT loop to print the eight sets of data stored in the three arrays.

Line 210 sends the computer to line 220 if E\$ does not equal "END". If it does, the program stops.

Lines 220 and 230 use two more FOR. . .NEXT loops to perform the calculations for each of the eight product categories.

Line 240 prints a dashed line to separate the old figures from the new.

Line 250 assigns "END" to the string variable E\$ and sends the computer back to line 200, where the previously used loop is used once more, this time printing the new totals.

Lines 260-280 are the DATA block that contains the eight product descriptions, the list prices, and the manufacturing cost figures. Many additional items could be added to these lines, requiring only that the 8 in the FOR. . .TO. . .NEXT loops be changed to reflect the total number of DATA items.

Figure 7-4 is a printout that resulted from the above program when a figure of 20 was used for the "percentage change" input. The top half represents the data stored in the DATA block, and the bottom half shows the new totals after 20% has been added.

The trouble with using arrays is that they make programs very difficult to decypher. Somehow things get terribly confusing, and it is always hard to figure out what's actually going on since, on paper at least, one lonely variable, the one used to identify the array, is used, in fact, to represent a large number of data. For this reason I have avoided using arrays in the program sections of this book, since my intention was to make these programs as easy to read and understand as possible.

Still, if you want to do a lot of program writing of your own, you should experiment with arrays, not only because they shorten and speed up programs, but also because they tend to reduce the chance of typing errors.

PRODUCT A	100	55
PRODUCT B	56.34	29
PRODUCT C	11.06	6
PRODUCT D	97.35	49
PRODUCT E	56.93	30
PRODUCT F	25	13
PRODUCT G	75.11	38
PRODUCT H	5.85	3

PRODUCT A	120	66
PRODUCT B	67.608	34.8
PRODUCT C	13.272	7.2
PRODUCT D	116.82	58.8
PRODUCT E	68.316	36
PRODUCT F	30	15.6
PRODUCT G	90.132	45.6
PRODUCT H	7.02	3.6

Figure 7-4. The printout that resulted from running the program.

8

Your Computer Talks Back

If you have the optional Speech Synthesizer, you can make the computer talk back to you. It's not only fun, it can also be used to produce some interesting and educational exercises. Before we start, let's introduce ourselves. Enter:

```
100 CALL SPGET("HELLO",H$)
200 CALL SPGET("THERE",T$)
300 CALL SPGET("I",I$)
400 CALL SPGET("AM",M$)
500 CALL SPGET("HOME",O$)
600 CALL SPGET("COMPUTER",C$)
700 CALL SPGET("HOW",W$)
800 CALL SPGET("ARE",R$)
900 CALL SPGET("YOU",Y$)
1000 CALL SAY(H$,T$,I$,M$, "A T I NINE NINE FOUR
      A",O$,C$,W$,R$,Y$)
```

and type RUN. It will take a few seconds, so be patient and wait for the computer to talk to you. (Be sure that the volume at the monitor is turned up.) Eventually an unemotional computer-generated voice will respond with "Hello there, I am a TI-99 4A Home Computer. How are you?", but entirely without punctuation or inflection. In writing this program we have used the two primary commands available with the speech function:

```
CALL SAY(---)  
CALL SPGET(---)
```

The first causes the voice synthesizer to be activated and to read whatever is written between the parentheses, though the total sentence should not exceed 126 characters. The second is comparable to the more familiar LET in that it permits us to assign frequently used words to string variables. As you will have noticed in looking at the above program, actual words or sentences must be enclosed in quotation marks, and when several string variables are used next to one another, they must be separated by *two* commas, whereas only *one* comma is needed between a string variable and a word or phrase enclosed in quotation marks.

The computer is capable of pronouncing only a limited number of words; the available words are listed in Appendix II at the back of the book. If you use a word that is not in its vocabulary, the computer will spell it rather than pronounce it. For instance:

```
CALL SAY("THE WEATHER TODAY IS BEAUTIFUL")
```

will produce the following:



"THE WEATHER TODAY IS BEAUTIFUL"

with only the words "the" and "is" actually being read as words and the others being simply spelled. Because the dictionary is rather limited, it is difficult to create meaningful speeches. On the other

hand, the computer's proclivity for spelling is a useful function in some other ways. Here is an example:

```
100 CALL CLEAR
110 PRINT "How do you spell the word "::
120 PRINT "that is the opposite of "
125 PRINT
130 PRINT " FLOOR?"
140 FOR X=1 to 10
150 PRINT
160 NEXT 10
170 INPUT "Spell the word . . . ":W$
180 FOR PAUSE=1 TO 100
190 NEXT PAUSE
200 CALL SAY("CEILING")
210 END
```

Here we have a simple test that combines two problems that must be solved. The first is knowing what the opposite of a certain word might be, and the second involves knowing how to spell it. The computer waits until you type in your answer, after which it waits for approximately 1 second before spelling the word. In Chapter 10 I have listed a number of programs that make use of this function in a variety of different combinations with other types of problems.

If the word to be spelled is actually in the dictionary, we have to let the computer know that we want it spelled by using the following convention:

```
CALL SAY("C,O,M,P,U,T,E,R")
```

which causes the word to be spelled despite the fact that it is included among the available words.

There is a way to modify verbs and nouns that are included in the word list by adding "s" to either verbs or nouns, or "ing" or "ed" to verbs. The routine involved in adding these suffixes is so complicated that it seems doubtful that it would be used often. The commands involved are CALL DEFS, CALL DEFING, and CALL DEFED; and the routine often involves truncating the word to be modified. The TI EXTENDED BASIC manual includes a rather lengthy

sample program (pages 206-210) that explains these functions.

There is a way in which we can make the speech synthesizer say just about anything (though some words tend to be hard to understand). To accomplish this we need something called the *Terminal Emulator II* Command Module, the primary purpose of which is its use with the optional modem.

The Terminal Emulator II, which is plugged into the computer in place of the TI EXTENDED BASIC module, has certain speech rules programmed into it, making it possible to use any English words as well as proper names and so on. The manner of use differs considerably from that which we have discussed so far. To cause the synthesizer to talk, the OPEN statement must be used, followed by a file number, the file name, and file attributes, if any. Here is an example:

```
100 CALL CLEAR
110 OPEN #1:"SPEECH",OUTPUT
120 PRINT #1:"THE NEXT IS A MATHEMATICS PROBLEM"
130 PRINT #1:"WHAT IS THE RESULT OF"
140 PRINT #1:"FOUR TIMES ONE HUNDRED AND TWENTY
    FIVE"
150 PRINT #1:"THE RESULT IS"
160 PRINT #1:4*125
170 END
```

When you now type RUN, the voice speaks the four lines of text and, when it comes to the fifth line, instead of reading the equation, it says "five zero zero," because whenever an equation is included in speech commands, the computer will automatically perform the calculation and the voice then pronounces the result.

In order to improve the speech quality, it is possible to insert punctuation marks—question marks and such—but they are treated as numerical characters and must, therefore, be used outside of quotation marks: "WORD", "WORD" with a blank space after commas and periods. Furthermore, it is possible to change the pitch of the voice (high, medium, low), and the slope characteristic, which controls the rate at which the pitch changes in spoken phrases. Both use numbers, 0 through 63 for pitch, where 0 is hardly audible, 1 is a whisper, and 63 is nearly a shout, and from 0 to 255 for slope numbers, with the best slope numbers usually being the integer of 10% of the pitch number times 32. The default values

are a pitch number of 43 and a slope number of 128 ($4 * 32$). The convention for use is:

// xx yyy

where xx is the pitch number and yyy is the slope number, used with a mandatory space between the two. In addition, the module includes inflection symbols, using ^, underline, and >. These symbols must be used ahead of the word to be affected and are included within the quotation marks. In order to become familiar with their effect, try different combinations. And last, there are "allophones," which refer to the special pronunciation of certain letters or letter combinations. There are a total of 125 such allophones programmed into the module; the list is on pages 41 and 42 of the Terminal Emulator II instruction book. To use this method of constructing words from individual allophone components, the routine is:

```
100 CALL CLEAR
110 OPEN #1:"SPEECH",OUTPUT
120 OPEN #2:"ALPHON",INTERNAL
130 PRINT #1:&CHR$(xxx)&CHR$(yyy)
```

where the numbers in parentheses represent the allophone sounds. In order to use more than one such sound with a PRINT statement, there must be a & sign between them, as otherwise the computer will read and speak only the first sound.

You can also use the INPUT statement in speech programs. For instance:

```
100 CALL CLEAR
110 OPEN #1:"SPEECH",OUTPUT
120 INPUT "WHAT DO YOU WANT ME TO SAY?-> ":A$
130 PRINT #1:A$
140 GOTO 120
```

will halt program execution until you type in whatever you want the computer to say.

Obviously, the capability of using the speech synthesizer in this manner immensely improves its usefulness. All manner of educa-

tional as well as fun programs can be written this way, and although use of the allophone portion might seem complicated, at first, it becomes relatively simple with a bit of practice. And remember, unless you're a stickler for excellence in computer-generated pronunciation, you don't need to use that capability at all.

The Terminal Emulator II is not terribly expensive (under \$50), and if you're planning to use the speech synthesizer for some serious programming, it is a virtual necessity.

9

Programs for the Home

When we talk about using a home computer for anything other than playing video games, we usually think first that it would be nice to use it to keep our checkbook balanced. But then it soon becomes obvious that this may be more trouble than it's worth for the financial activities of the average household. If, on the other hand, those financial activities include expenses divided into several different categories, such as personal as well as tax-deductible business expenses, then a relatively simple computer program can be used to keep a continuing record, simplifying the eventual task of determining the taxes that must be paid.

In this chapter we'll examine a number of programs using the instructions and functions discussed in the previous chapters that can be of practical use in the home. The first is the kind of checkbook-keeping program we just discussed. Another deals with a simple data-base program designed to keep a permanent record of names, addresses, phone numbers, and birth dates of family, friends, and acquaintances. Still another is designed to perform the conversion and calculation chores that confront us when we try to convert a cookbook recipe that might have been written for a party of six or eight to serve a smaller or larger group. Also included is a program designed to simplify the task of filling out the Schedule C tax form for self-employed persons. Other programs determine the day of the week for any date, past, present, or future; provide assistance in making vacation or other travel plans; help to control a rigid budget; and so on.

CHECKBOOK PROGRAM

Let's look at the checkbook program first. It falls into the category known as *file programs* that we discussed in Chapter 7. As written, the program requires that you have one disk drive and that data be recorded on a disk on which there is sufficient empty space to accommodate the data files to be created.

CHECKBOOK PROGRAM

This checkbook-keeping program is a sequential file program that stores input data in separate data file called BALANCE.

```

100 REM CHECKBOOK/TI99/4A
200 GOSUB 1000
210 PRINT "A program that keeps": :
220 PRINT "your checkbook balanced."
230 GOSUB 1100
240 GOSUB 1000
250 INPUT "First run? (Y/N) ":FR$
260 IF FR$<>"Y" THEN 350
270 PRINT
280 INPUT "Current balance? $":DD
281 BW=0
282 PW=0
290 OPEN #1:"DSK.TIONE.BALANCE"
300 PRINT #1:DD
301 PRINT #1:BW
302 PRINT #1:PW
310 CLOSE #1
350 GOSUB 1000
360 OPEN #1:"DSK.TIONE.BALANCE"
370 INPUT #1:DD
371 INPUT #1:BW
372 INPUT #1:PW
380 CLOSE #1
390 PRINT "Last balance= $";DD
391 PRINT
392 PRINT "Business expense to date= $";BW: :
393 PRINT "Personal expense to date= $";PW: :
400 GOSUB 1100
410 GOSUB 1000
420 PRINT "Your choice:": :
430 PRINT "-----"
440 PRINT 1;" Deposits.": :
450 PRINT 2;" Withdrawals."
460 PRINT "-----"

```

(continued)

```
470 INPUT " Pick one ":DW
480 IF DW=1 THEN 500 ELSE 600
500 GOSUB 1000
510 INPUT "Amount deposited? $":D
520 DD=DD+D
525 IF D=0 THEN 900
530 GOSUB 800
550 GOTO 410
600 GOSUB 1000
610 INPUT "Amount withdrawn? $":W
620 DD=DD-W
621 GOSUB 1300
625 IF W=0 THEN 900
630 GOSUB 800
650 GOTO 410
800 GOSUB 1000
810 PRINT "After last entry,": :
820 PRINT "key in 0": :
830 GOSUB 1100
840 RETURN
900 OPEN #1:"DSK.TIONE.BALANCE"
910 PRINT #1:DD
911 PRINT #1:BW
912 PRINT #1:PW
920 CLOSE #1
930 GOSUB 1000
940 PRINT TAB(12);"End."
950 END
1000 CALL CLEAR
1010 FOR X=1 TO 10
1020 PRINT
1030 NEXT X
1040 RETURN
1100 PRINT
1110 INPUT "Press >ENTER< ":Y$
1120 RETURN
1300 GOSUB 1000
1301 INPUT "Business or personal? (B/P) ":BP$
1310 IF BP$="B" THEN 1350
1320 PW=PW+W
1330 RETURN
1350 BW=BW+W
1360 RETURN
```

When the program is run, it first displays its purpose:

A program that keeps
your checkbook balanced.

It then asks:

First run? (Y/N)

because only the first time the program is used will you be required to enter the last balance. During all future runs the last balance will

be displayed automatically. If this is the first run, the program continues with:

Your choice:

-
- 1 Deposits.
 - 2 Withdrawals.
-

If, on the other hand, the program has been run before and therefore has stored previously entered information, it first displays:

Last balance= \$xxx.xx
Business expense to date= \$xxx.xx
Personal expense to date=\$xxx.xx

after which it goes to the previous display, asking whether you want to enter deposits or withdrawals. It is then ready to accept as many deposit or withdrawal items as you want to make, asking, in the case of withdrawals:

Business or personal? (B/P)

because the computer performs the necessary calculations internally so that eventually, after the last entry, it can display the results in terms of the current balance and the totals for personal and business expenditures. Now let's examine the program line by line to be sure that each statement and command is clearly understood.

Line 100 is a REMark line that identifies the program but is ignored by the computer.

Line 200 goes to a subroutine that is used repeatedly to clear the display screen.

Lines 210 and 220 display the purpose of the program.

Line 230 goes to another subroutine used to tell you to press the >ENTER< key in order to continue program operation.

Line 250 asks if this is the first time the program is run. Your answer, either Y for yes or N for no, is then assigned to the string variable FR\$.

Line 260 checks whether you typed in Y for yes or N (or any other letter) for no, telling the computer to skip the next few lines and go to line 350 if this is *not* the first run.

Line 270 uses the PRINT command with nothing after it to place a blank line into the display.

Line 280 asks you to key in the current balance, assigning the reply to the numeric variable DD. If this figure is in excess of \$999.99, do *not* use commas (10000.00, not 10,000.00), because numeric variables can consist only of digits and decimal points and no other symbols.

Lines 281 and 282 make sure that no leftover values are assigned to the numeric variables BW and PW.

Line 290 OPENS a data file named TIONE.BALANCE. Since this is the first run, the command means that a data file by that file name is being newly created.

Line 300 uses the PRINT statement to enter data into that newly created file. Those data will be the value of DD that you have just keyed in, and the values of zero for BW and PW.

Line 310 CLOSEs the data file. This is important. Whenever such a data file has been OPENed, it must be CLOSEd before program execution can continue.

Line 350 again uses the subroutine to clear the screen. This is the line to which the computer is sent if this is *not* the first run of the program.

Lines 360 and 372 OPEN the data file again, using the INPUT statement to ready the computer to display the data stored in that file on the monitor screen by placing them into the computer's RAM.

Line 380 CLOSEs that data file.

Line 390 causes the last balance and the expense data to be displayed.

Lines 400 and 410 use the subroutines to ask you to press >ENTER< and then to clear the screen.

Lines 420-470 display the menu of the types of entries you may make, asking you to pick one by typing either 1 or 2. Your answer is then assigned to the numeric variable DW.

Line 480 examines the value assigned to DW. If it is 1, the computer is told to go to line 500; otherwise, using the ELSE command, it is told to go to line 600.

Line 500 clears the screen.

Line 510 asks you to key in the amount of a deposit, assigning that figure to the numeric variable D.

Line 520 adds the value of D(current deposit) to the value of DD (last balance) to produce the new balance, assigning it again to the numeric variable DD.

Line 525 checks whether the keyed-in amount was zero, indicating that the entry was the last deposit to be entered. If so, the computer is told to go to line 900.

Line 530 sends the computer to a subroutine that tells you to key in zero after the last entry.

Line 550 sends the computer back to line 410 to ask if you want to key in more deposits or withdrawals.

Line 600 clears the screen again.

Line 610 asks you to key in the amount of a withdrawal, assigning that figure to the numeric variable W.

Line 620 deducts the value of W (withdrawal) from the value of DD (last balance), assigning the result to the variable DD.

Line 625 checks whether the last amount keyed in was zero, in which case the computer is sent to line 900.

Line 630 sends the computer to the subroutine in lines 800-840.

Line 650 sends the computer to line 410 for another choice of entering either deposits or withdrawals.

Lines 800-840 are the subroutine discussed above.

Lines 900-912 OPEN the data file and use the PRINT command to record the keyed-in data in that file.

Line 920 CLOSEs the data file before the computer continues with program execution.

Lines 930-950 cause the word END to be displayed on the screen and, in line 950, tell the computer that it has reached the end of the program.

Lines 1000-1040 are the subroutine used to clear the screen and place some blank lines into the display.

Lines 1100-1120 are the subroutine asking that you press >ENTER<.

Lines 1300-1360 are the subroutine that asks whether the withdrawal is for personal or business expenses. In line 1310 the answer assigned to the string variable BP\$ is examined; if the value of BP\$ equals B (business), the computer is told to go to line 1350. In line 1320 the value of W (withdrawal) is added to the value of PW (personal expenses) and the result is again assigned to PW. In line 1330 the computer is sent back to where it came from. In line 1350 the value of W is added to the value of BW (business expenses) and is then again assigned to BW. And in line 1360 the computer is told to RETURN to where it came from.

DAY OF THE WEEK PROGRAM

Now let's look at a different kind of program, one that does not create data files while it is being run. This program performs some rather esoteric computations along the way. The purpose of the program is to determine the day of the week for any date, past (though after 1584, the start of the Gregorian calendar), present, or future. If you type in a date (2 10 1918), the display will respond with the day of the week (Sunday). This program has its practical uses in determining dates for meetings, parties, vacations, and so on.

DAY OF THE WEEK PROGRAM

This program determines the day of the week for any date after 1584.

```
100 REM DAY OF THE WEEK/TI99/4A
110 CALL CLEAR
120 PRINT "This program determines": :
130 PRINT "the day of the week": :
140 PRINT "for any date after 1584,"": :
150 PRINT "the start of the": :
160 PRINT "Gregorian Calendar"
170 GOSUB 1000
180 GOSUB 800
190 CALL CLEAR
200 INPUT "Month (1 or 2 digits) ":M
210 PRINT
220 INPUT "Day (1 or 2 digits) ":D
230 PRINT
240 INPUT "Year (4 digits) ":Y
250 GOSUB 1000
260 ON M GOTO 270,290,310,330,350,370,390,410,430,
    450,470,490
270 E=D
280 GOTO 510
290 E=D+31
300 GOTO 510
310 E=D+59
320 GOTO 510
330 E=D+90
340 GOTO 510
350 E=D+120
360 GOTO 510
370 E=D+151
```

(continued)

```

380 GOTO 510
390 E=D+181
400 GOTO 510
410 E=D+212
420 GOTO 510
430 E=D+243
440 GOTO 510
450 E=D+273
460 GOTO 510
470 E=D+304
480 GOTO 510
490 E=D+334
500 GOTO 510
510 Z=(Y-1584)/Y
520 A=INT(Z)
530 Z=Z-A
540 IF Z=0 THEN 550 ELSE 560
550 E=E+1
560 B=(Y-1)/4
570 B=INT(B)
580 C=(Y-1)/100
590 C=INT(C)
600 F=(Y-1)/400
610 F=INT(F)
620 G=E+Y+B-C+F
630 H=G/7
640 I=INT(H)
650 I=H-I
660 I=I*7
670 CALL CLEAR
680 IF I<1.2 THEN 820
690 IF I<2 THEN 700 ELSE 710
700 IF I>1.5 THEN 840 ELSE 710
710 IF I<3.2 THEN 720 ELSE 730
720 IF I>3 THEN 860 ELSE 730
730 IF I<4 THEN 740 ELSE 750
740 IF I>3.5 THEN 880 ELSE 750
750 IF I<5.5 THEN 760 ELSE 770
760 IF I>5 THEN 900 ELSE 770
770 IF I<6 THEN 780 ELSE 790
780 IF I>5.6 THEN 920 ELSE 790
790 IF I=0 THEN 940
800 INPUT "Press >ENTER< ":Y$
810 RETURN
820 PRINT M; ", "; D; ", "; Y; " is Sunday"
830 GOTO 960
840 PRINT M; ", "; D; ", "; Y; " is Monday"
850 GOTO 960
860 PRINT M; ", "; D; ", "; Y; " is Tuesday"
870 GOTO 960
880 PRINT M; ", "; D; ", "; Y; " is Wednesday"
890 GOTO 960

```

(continued)

```
900 PRINT M;"", ";D;", ";Y;" is Thursday"
910 GOTO 960
920 PRINT M;"", ";D;", ";Y;" is Friday"
930 GOTO 960
940 PRINT M;"", ";D;", ";Y;" is Saturday"
950 GOTO 960
960 PRINT : :
970 PRINT : :
980 PRINT "          End."
990 END
1000 PRINT "-----"
1010 RETURN
```

When you run this program it first displays its purpose;

This program determines
the day of the week
for any date after 1584,
the start of the
Gregorian calendar

after which it asks that you type in:

Month (1 or 2 digits)
Day (1 or 2 digits)
Year (4 digits)

and, after a brief hesitation during which the computer performs the necessary calculations, the display responds with something like:

2, 10, 1918 is Sunday

followed by:

End.

After I wrote this program I used the RESEQUENCE command, which automatically renumbers all program lines to intervals of 10 (unless otherwise commanded), also automatically adjusting the different GOTO, GOSUB, and IF...THEN...ELSE lines to the revised line numbers.

Let's look at the program line by line:

Line 100 identifies the program.

Line 110 clears the screen.

Lines 120-160 place the purpose and description of the program into display.

Line 180 sends the computer to the subroutine asking you to press >ENTER<.

Line 190 clears the screen.

Lines 200-240 ask you to key in the month, day, and year for which you want to determine the day of the week.

Line 250 sends the computer to a subroutine that causes a line to be displayed below the previous lines of text.

Line 260 uses the ON GOTO command to send the computer to one of 12 line numbers, depending on the month in question.

Lines 270-500 add the appropriate number of days to the date, based on the selected month. Here it is determined that the 10th of February, for instance, is the 41st day of the year. In each instance the computer is sent on to line 510.

Lines 510-660 perform the actual calculations, taking account of the fact that every fourth year is a leap year.

Line 670 again clears the screen.

Lines 680-790 examine the value of the variable I relative to some figures that are part of the established mathematical formula determining the day of the week.

Lines 800 and 810 are the subroutine described above.

Lines 820-940 display the result of the calculation, each time sending the computer on to line 960.

Line 950 is superfluous, because it sends the computer to line 960, which is the next line.

Lines 960 and 970 place some blank lines into the display.

Line 980 places the word END into display.

Line 990 tells the computer that the end of the program has been reached.

Lines 1000 and 1010 are the subroutine that causes a dashed line to be displayed.

It is possible to make the program repeat itself if several dates are to be examined. The lines that must be changed or added are:

```

980 INPUT "Examine another date? (Y / N) ":YN$
990 IF YN$="Y" THEN 190 ELSE 1020
1020 PRINT ::
1030 PRINT "      End."
1040 END

```

These lines eliminate the need to run the program again from the beginning, saving a certain amount of time.

TRAVEL PROGRAM

Our next program deals with a variety of details that are usually associated with making plans for travel and vacations. The program is designed to determine the cost of any trip and the comparative costs, using four different means of transportation: automobile, commercial airline, bus, or private aircraft. In each category it displays the total time en route as well as cost for one-way, round-trip, and for the entire trip including food and lodging.

TRAVEL PROGRAM

Travel comparison program.

```

10 REM  TI EXTENDED BASIC
100 REM TRAVEL COMPARISON
110 D$="Distance to travel (miles)?      "
120 T1$="Approx.time to destination="
130 T2$=" hours and "
140 T3$=" minutes."
150 FL$="Food/lodging en route?          $"
160 GT0$="Grand total (one way)=          $"
170 GTR$="Grand total (round trip)=       $"
180 FT$="Final total (entire trip)=       $"
190 CA$="Cost,travel to airport?          $"
200 TA$="Time,travel to airport?          "
210 NP$="Number of paying passengers?    "
220 CT$="Cost,each ticket,one way?       $"
230 CD$="Cost,travel from airport?       $"
240 TD$="Time,travel from airport?       "
250 CR$="Cost,rental car (if any)?       $"

```

(continued)

```
260 CRR$="Total,car,round trip="          "$"
270 TRO$="Time,car,one way="
280 CRA$="Total,airlines,round trip="      "$"
290 TOA$="Time,airlines,one way="
300 FTA$="Final total,airlines="          "$"
310 FTC$="Final total,car="              "$"
320 DISPLAY AT(5,1)ERASE ALL:"This program analyzes" ::
330 DISPLAY AT(7,1):"travel costs for different"
340 DISPLAY AT(9,1):"means of transportation"
350 GOSUB 2460
360 GOSUB 2480
370 GOSUB 2440
380 DISPLAY AT(5,1):"Do you plan to travel by:"
390 DISPLAY AT(6,1):"-----"
400 DISPLAY AT(7,1):1,"Automobile"
410 DISPLAY AT(9,1):2,"Airlines"
420 DISPLAY AT(11,1):3,"Bus"
430 DISPLAY AT(13,1):4,"Private plane"
440 DISPLAY AT(14,1):"-----"
450 DISPLAY AT(15,1):5,"Exit program"
460 DISPLAY AT(16,1):"-----"
470 DISPLAY AT(17,1):"Which?"
480 INPUT WHICH
490 GOSUB 2440
500 ON WHICH GOTO 510,880,1360,1910,2510
510 INPUT D$:MILES
520 INPUT "Average speed (incl.stops)?"    ":SPEED
530 INPUT "Aver.cost of gas per gal?"      "$":GAS
540 INPUT "Aver.miles per gallon?"         ":MPG
550 INPUT "Annual car expense(exc.gas)?"   "$":YEAR
560 INPUT "Aver.annual miles driven?"     ":ANNUAL
570 GOSUB 2460
580 GOSUB 2400
590 TIME=MILES/SPEED
600 TIME1=INT(TIME)
610 TIME2=(TIME-TIME1)*.6*100
620 TIMES=TIME1
630 TIME6=TIME2
640 PRINT T1$;TIME1;T2$;TIME2;T3$
650 GAS=GAS*MILES/MPG
660 GAS=INT(GAS*100+.5)/100
670 PRINT "Fuel cost="                    "$":GAS
680 YEAR=YEAR/ANNUAL*MILES
690 YEAR=INT(YEAR*100+.5)/100
700 PRINT "Other car costs="              "$":YEAR
710 TOTAL=GAS+YEAR
720 GOSUB 2460
730 PRINT "Total car costs="              "$":TOTAL
740 GOSUB 2460
750 INPUT FL$:FOOD
760 TOTAL1=TOTAL+FOOD
770 GOSUB 2460
```

(continued)

```

780 PRINT GTO$;TOTAL1
790 TOTAL2=TOTAL1*2
800 PRINT GTR$;TOTAL2
810 TRIP=TOTAL2+MOTEL
820 PRINT FT$;TRIP
830 TRIPCAR=TRIP
840 GOSUB 2460
850 GOSUB 2480
860 GOSUB 2440
870 GOTO 380
880 INPUT D$;MILES
890 INPUT "Scheduled air time?           ":TIMEX
900 INPUT CA$;TAXI
910 INPUT TA$;GROUND
920 INPUT "Wait time at airport?         ":AIRPT
930 INPUT NP$;PERSONS
940 INPUT CT$;TICKET
950 INPUT CD$;TAXI1
960 INPUT TD$;GROUNDX
970 INPUT CR$;CAR
980 GOSUB 2460
990 GOSUB 2400
1000 TIMEX1=INT(TIMEX)
1010 TIMEX2=(TIMEX-TIMEX1)/.6
1020 TIMEX=TIMEX2+TIMEX2
1030 GROUND1=INT(GROUND)
1040 GROUND2=(GROUND-GROUND1)/.6
1050 GROUND=GROUND1+GROUND2
1060 GROUNDX1=INT(GROUNDX)
1070 GROUNDX2=(GROUNDX-GROUNDX1)/.6
1080 GROUNDX=GROUNDX1+GROUNDX2
1090 AIRPT1=INT(AIRPT)
1100 AIRPT2=(AIRPT-AIRPT1)/.6
1110 AIRPT=AIRPT1+AIRPT2
1120 TIME=TIMEX+GROUND+GROUNDX+AIRPT
1130 TIME1=INT(TIME)
1140 TIME2=(TIME-TIME1)*.6*100
1150 TIME2=INT(TIME2)
1160 TIME7=TIME1
1170 TIME8=TIME2
1180 PRINT T1$;TIME1;T2$;TIME2;T3$
1190 COST=TAXI+(PERSONS*TICKET)+TAXI1
1200 PRINT GTO$;COST
1210 TCOST=(COST*2)+CAR
1220 TOTAL5=TCOST
1230 PRINT GTR$;TCOST
1240 GOSUB 2460
1250 GOSUB 2480
1260 GOSUB 2440
1270 PRINT FTC$;TRIPCAR
1280 TRIP=TCOST+MOTEL
1290 GOSUB 2460

```

(continued)

```
1300 PRINT FT$;TRIP
1310 TRIPAIR=TRIP
1320 GOSUB 2460
1330 GOSUB 2480
1340 GOSUB 2440
1350 GOTO 380
1360 INPUT D$;MILES
1370 INPUT "Scheduled bus time?          ":TIMEX
1380 INPUT "Cost,travel to bus terminal?  $":TAXI
1390 INPUT "Time,travel to bus terminal?  ":GROUND
1400 INPUT "Waiting time at terminal?     ":AIRPT
1410 INPUT NP$;PERSONS
1420 INPUT CT$;TICKET
1430 INPUT "Cost,travel from terminal?    $":TAXI1
1440 INPUT "Time,travel from terminal?    ":GROUNDX
1450 INPUT FL$;FOOD
1460 INPUT CR$;CAR
1470 GOSUB 2460
1480 GOSUB 2400
1490 GROUND1=INT(GROUND)
1500 GROUND2=(GROUND-GROUND1)/.6
1510 GROUND=GROUND1+GROUND2
1520 GROUNDX1=INT(GROUNDX)
1530 GROUNDX2=(GROUNDX-GROUNDX1)/.6
1540 GROUNDX=GROUNDX1+GROUNDX2
1550 AIRPT1=INT(AIRPT)
1560 AIRPT2=(AIRPT-AIRPT1)/.6
1570 AIRPT=AIRPT1+AIRPT2
1580 TIME=TIMEX+GROUND+GROUNDX+AIRPT
1590 TIME1=INT(TIME)
1600 TIME2=(TIME-TIME1)*.6*100
1610 TIME2=INT(TIME2)
1620 COST=TAXI+(TICKET*PERSONS)+TAXI1+FOOD
1630 PRINT T1$;TIME1;T2$;TIME2;T3$
1640 HOUR1=TIME1
1650 HOUR2=TIME2
1660 GOSUB 2460
1670 PRINT GTO$;COST
1680 TCOST=(COST*2)+CAR
1690 TOTAL6=TCOST
1700 PRINT GTR$;TCOST
1710 TRIP=TCOST+MOTEL
1720 GOSUB 2460
1730 PRINT FT$;TRIP
1740 TRIPBUS=TRIP
1750 GOSUB 2460
1760 PRINT CRR$;TOTAL2
1770 PRINT CRA$;TOTAL5
1780 PRINT TRO$;TIME5;T2$;TIME6;T3$
1790 PRINT TOA$;TIME7;T2$;TIME8;T3$
1800 GOSUB 2460
1810 GOSUB 2480
```

(continued)

```

1820 GOSUB 2440
1830 PRINT FTC$;TRIPCAR
1840 PRINT FTA$;TRIPAIR
1850 GOSUB 2460
1860 PRINT FT$;TRIPBUS
1870 GOSUB 2460
1880 GOSUB 2480
1890 GOSUB 2440
1900 GOTO 380
1910 INPUT D$;MILES
1920 INPUT "Av.cruise speed (nm)?           ":SPEED
1930 SPEED=SPEED/1.15089
1940 INPUT "Cost,operation/hour?           $":COST
1950 INPUT CA$;TAXI
1960 INPUT TA$;GROUND
1970 INPUT CD$;TAXI1
1980 INPUT TD$;GROUNDX
1990 INPUT CR$;CAR
2000 GOSUB 2460
2010 GOSUB 2400
2020 TIME=MILES/SPEED
2030 COST=(COST*TIME)+TAXI+TAXI1+CAR
2040 TIME1=INT(TIME)
2050 TIME2=(TIME-TIME1)*.6
2060 TIME=TIME1+TIME2+GROUND+GROUNDX
2070 TIMEX=INT(TIME)
2080 TIMEXX=(TIME-TIMEX)*.6*100
2090 TIMEXX=INT(TIMEXX)
2100 PRINT T1$;TIMEX;T2$;TIMEXX;T3$
2110 GOSUB 2460
2120 COST=INT(COST*100+.5)/100
2130 PRINT GTO$;COST
2140 TCOST=(COST*2)-CAR
2150 PRINT GTR$;TCOST
2160 TRIP=TCOST+MOTEL
2170 GOSUB 2460
2180 PRINT FT$;TRIP
2190 GOSUB 2460
2200 GOSUB 2480
2210 GOSUB 2440
2220 PRINT CRR$;TOTAL2
2230 PRINT CRA$;TOTAL5
2240 PRINT "Total cost,bus,round trip=      $";TOTAL 6
2250 PRINT TRO$;TIMES;T2$;TIME6;T3$
2260 PRINT TOA$;TIME7;T2$;TIME8;T3$
2270 PRINT "Time by bus, one way=          ";HOUR1;T2$;
      HOUR2;T3$
2280 GOSUB 2460
2290 GOSUB 2480
2300 GOSUB 2440
2310 PRINT FTC$;TRIPCAR
2320 PRINT FTA$;TRIPAIR

```

(continued)

```
2330 PRINT "Final total, bus=                $";TRIPBUS
2340 GOSUB 2460
2350 PRINT "Final total,private plane=      $";TRIP
2360 GOSUB 2460
2370 GOSUB 2480
2380 GOSUB 2440
2390 GOTO 380
2400 INPUT "Food/lodging at destination?    $":MOTEL
2410 GOSUB 2460
2420 PRINT ::
2430 RETURN
2440 CALL CLEAR
2450 RETURN
2460 PRINT "-----"
2470 RETURN
2480 PRINT ::
2490 INPUT "Press >ENTER<   ":Y$
2500 RETURN
2510 GOSUB 2440
2520 PRINT "End."
```

When the program is activated it displays the menu:

Do you plan to travel by:

-
- 1 Automobile
 - 2 Airlines
 - 3 Bus
 - 4 Private plane
-

5 Exit program

Which?

After that, depending on the selection made, it asks all the pertinent questions relative to the information needed in order to perform the calculations. If several different means of transportation are called up, one after the other, the program displays not only the totals for

the selected mode but also the previously determined totals, in order to provide an easy means of comparison.

The best way to see what takes place in the different parts of the program is to examine it line by line.

Line 10 is a REMark that this program uses TI EXTENDED BASIC.

To convert the program to run on TI BASIC, see the discussion at lines 320-340.

Line 100 identifies the program.

Lines 110-310 assign various repeated used strings to string variables.

Lines 320-340 use the DISPLAY command in the manner in which it is available in TI EXTENDED BASIC, where the numbers in parentheses after AT determine the row and column number at which the text is to be displayed on the screen, and the ERASE ALL in line 320 is equivalent to the CALL CLEAR command that clears the screen. In ordinary TI BASIC this convention is not available, and DISPLAY would have to be used in the same manner in which the PRINT command is normally used.

Lines 350-370 call up three subroutines. The first causes a dashed line to be displayed. The second asks you to press >ENTER< in order to continue program execution. And the third clears the screen.

Lines 380-470 again use the DISPLAY command as described above.

Line 480 accepts the typed-in choice and assigns it to the variable WHICH.

Line 490 calls up the subroutine that clears the screen.

Line 500 sends the computer to the appropriate section of the program, depending on the previously keyed-in selection.

Lines 510-560 ask you to key in the distance to be traveled in miles, the average speed including stops for food and fuel (40 mph would probably be a good guess), the average cost of gas per gallon, the miles per gallon applicable to the automobile to be used, the annual car expenses (except gas), such as car payments, insurance, maintenance, and so on, and the average number of miles driven per year. In each case the keyed-in data are assigned to numeric variables that are subsequently used in the calculations.

Line 570 calls up the subroutine that displays a dashed line.

Line 580 calls up a subroutine that asks you to key in the costs involved with food and lodging at the destination.

Lines 590-610 calculate the time en route, one way, based on the input average speed and distance to travel. In line 600 the integer of the result, representing the hours, is separated from the decimal portion. In line 610 the decimal portion, representing tenths and hundredths of an hour, is multiplied by .6 and 100 in order to be converted to minutes.

Lines 620 and 630 assign additional numeric variables (TIME5, TIME6) to the results, so that they can be used later when they are needed to display comparisons with other means of travel.

Line 640 causes the result to be displayed, using the previously assigned string variables along with the numeric variables representing hours and minutes.

Line 650 determines the cost of fuel by multiplying the price per gallon by the total mileage and dividing the result by the miles-per-gallon figure.

Line 660 employs a frequently used formula that limits the number of displayed decimals to two, rounding off the final decimal, depending on the value of the subsequent digit.

Line 670 displays the result.

Line 680 determines the portion of the annual car cost (excluding fuel) that should be figured as part of the trip cost by dividing the annual cost by the average number of miles driven each year and then multiplying the result by the distance to be traveled.

Line 690 again limits the number of decimals to two.

Line 700 prints the result.

Lines 710-740 add the two figures and display the result between two dashed lines.

Line 750 asks you to key in the anticipated cost for food and possibly lodging en route, assigning that figure to the numeric variable FOOD.

Lines 760-830 first add that figure to the previous total in order to display the total one-way cost in line 780. In line 790 that figure is doubled and in line 800 the round-trip total is displayed. Line 810 adds the cost of food and lodging, represented by the numeric variable MOTEL, to the previous total, displaying the final result in line 820. And, finally, that figure is assigned to the numeric variable TRIPCAR in line 830 for use later on.

Lines 840-870 are the last lines in the automobile section of the program, placing a dashed line into display, asking you to press

>ENTER< to continue program execution, clearing the screen, and then, in line 870, returning the program to the menu for another selection.

Lines 880-1350 represent the airline portion of the program, first asking all the pertinent questions associated with travel by commercial airlines (lines 880-970), covering such subjects as travel to the departure airport and from the destination airport, whether a rental car will be needed at the destination, and so on. In lines 1000-1170 the hours-plus-minutes figures that were keyed in are first converted to decimal values (by dividing minutes by .6) in order to be used in the computations. Once the results have been obtained, the decimal values are again converted to minutes by multiplying by .6 and 100. Most of the other lines in this section are similar to those in the first section.

Lines 1360-1900 represent the portion of the program that deals with traveling by bus. The various calculations and displays are comparable to those in the airline section.

Lines 1910-2390 comprise the section of the program that deals with travel by private plane. Here, too, most of the calculations and displays are comparable to those discussed above. Line 1920 asks you to key in the average cruising speed in nautical miles, and line 1930 converts that speed to statute miles, which are used in entering the distance to be traveled. Line 1940 asks for the cost of operating the aircraft based on one flying hour. Those costs include fixed as well as variable costs and are normally known to pilots.

Lines 2400-2500 are the subroutines described above, and lines 2510 and 2520 cause the screen to be cleared and the word "End." to be displayed.

As you will have noticed, I used the RESEQUENCE command to renumber all the lines after the program was written and debugged (and then I added still another line-line10). I would like to offer a word of warning with reference to that command: Don't use the command unless and until the program is complete, debugged, and tested because, especially if the program is a long one, you will have great difficulty in finding a given line that may need editing in some way. Usually, during the process of program writing we tend to assign easy-to-remember line numbers to separate sections. For instance, in writing the above program I used 1000 and up for the

automobile, 2000 and up for the airlines, and so on. Thus, while debugging and editing, I had no trouble finding the lines I was looking for. Resequencing changes all that and makes editing more difficult.

NAME AND ADDRESS LIST

Keeping track of the names, addresses, phone numbers, and, where appropriate, birthdays of relatives, friends, and acquaintances usually calls for the use of some sort of address book. The trouble is that, unless we remember at least the months in which we might have to send birthday cards to certain persons, we're likely to forget to do so in time. This program is a very simple record-keeping program designed to store the names, addresses, telephone numbers, and birthdays of any number of persons. The program will search for entries by last name, city, state, zip code, area code, or birth month, always displaying all records that conform to the entry requirements. Thus, if you enter JONES, it will produce all records for persons with the last name JONES. If you enter CHICAGO, it will display everyone living in Chicago. If you key in CA, it will display everyone living in California. If you type 1, it will display all persons who live in an area where the zip code starts with 1. If you enter 213, the display will respond with everyone whose area code is 213. And if the entry is 2, it will display each person born in February. In addition, if called upon to do so, the program will send the information to a line printer to have the selected list of data printed.

The program does not automatically create a DATA file. Instead, the names, addresses, and so on, must be entered in the DATA lines, following a specific format that must be adhered to if the program is to function satisfactorily. Because this is of overriding importance, let's look at that format before going into detail about the rest of the program.

In the Name & Address Program listing, I have used 10 arbitrary names with the accompanying data. The lines used for that purpose are 1000-1090, and the lines available for such DATA entries are

1000-4999, which means that, theoretically at least, you can enter up to 3999 names, obviously more than you're likely to need. The format for DATA entry is as follows (use only uppercase letters):

LAST NAME,FIRST NAME AND MIDDLE NAME OR INITIAL,STREET
OR P.O. BOX NUMBER,CITY,STATE (2-LETTER ABBREVIATION,ZIP
CODE,AREA CODE,PHONE NUMBER,BIRTH MONTH,BIRTH DAY

It is important that no commas be used anywhere except where indicated. (It is possible to use commas for such names as ABC COMPANY, INC., but then the entire string must be enclosed in quotation marks: "ABC COMPANY, INC.") The placement of commas is so important because the computer READs everything between commas as *one DATA* item, and each record (line) must contain the same number of DATA items. Thus, if some information is missing (area code, birth date, or some such), commas must be used with nothing in between (DOE,JOHN,,,,,,,,, would be needed for a name for which no additional information is available). In other words, each record consists of 10 DATA items separated by nine commas, with the first item, the last name, ahead of the first comma, and the last item, birth day, to the right of the ninth comma. Now let's take a closer look at the program.

THE NAME AND ADDRESS PROGRAM

A program that can be used to store names, addresses, phone numbers, and birth dates.

```

100 REM RANDOM ACCESS
110 REM DATA BASE
120 REM TI99/4A
150 BX$="Birth date: "
200 GOSUB 11000
210 PRINT "This is a simple": :
220 PRINT "random-access data-base": :
230 PRINT "program."
240 GOSUB 11500
250 PRINT "Program lines available": :
260 PRINT "for data entry are ": :
270 PRINT "lines 1000 through 4999": :
280 GOSUB 11500

```

(continued)

```
290 PRINT "Do you want the data sent": :
300 INPUT "to the line printer? Y/N ":YY$
310 GOSUB 11000
320 PRINT "The data stored are ": :
330 PRINT "name, address, phone number": :
340 PRINT "You may retrieve data by": :
350 PRINT 1;" Last name"
360 PRINT 2;" City"
370 PRINT 3;" State (2 Letters)"
380 PRINT 4;" Zip (1st digit)"
390 PRINT 5;" Area code"
400 PRINT 6;" Birth month"
410 GOSUB 11500
420 INPUT "Pick one ":P
430 ON P GOTO 5000,6000,7000,8000,9000,10000
500 READ LN$,FN$,AD$,CT$,ST$,ZC,AC$,PN$,BM$,BD$
510 RETURN
1000 DATA JONES,PHIL,P.O.BOX 1246,SANTA FE,NM,87501,505,
    555 2374,2,11
1010 DATA DOE,JOHN,321 MAIN STREET,MIAMI,FL,31011,305,555
    3578,12,21
1020 DATA DOE,JANE,65 BETA WAY,LOS ANGELES,CA,90027,213,
    555 9611,7,24
1030 DATA JONES,PETER,55 PARK AVE.,LOS ANGELES,CA,90027,
    213,555 8871,1,22
1040 DATA SMITH,HARRY,77 OREGON DR.,DALLAS,TX,75010,214,
    555 6609,10,15
1050 DATA GORDON,FRANK,882 CENTRAL AV.,NEW YORK,NY,10015,
    212,555 5518,1,7
1060 DATA CARTER,JIMMY,45 2ND STREET,BOSTON,MA,03021,617,
    555 6492,3,16
1070 DATA SMITH,MARY,P.O.BOX 129,CHICAGO,IL,60506,312,555
    8755,9,14
1080 DATA CARTER,ELLEN,666 MARKET ST.,SAN FRANCISCO,CA,
    94132,415,555 9876,7,4
1090 DATA GORDON,DR.JOHN,45 BEVERLY DR.,BEVERLY HILLS,CA,
    90210,213,555 4519,4,23
5000 GOSUB 11000
5005 RESTORE
5010 INPUT "Last name? ":NL$
5020 FOR Z=1 TO 1000
5030 GOSUB 500
5040 IF NL$=LN$ THEN 5100
5050 NEXT Z
5100 GOSUB 11500
5200 PRINT FN$;" ";LN$
5210 PRINT AD$
5220 PRINT CT$;" ";ST$;" ";ZC
5230 PRINT AC$;"-";PN$
5240 PRINT BX$;BM$;" ";BD$
5250 IF YY$="Y" THEN 5300
5260 PRINT : :
```

(continued)

```

5290 ON P GOTO 5030,6030,7020,8030,9030,10030
5300 OPEN #1:"RS232"
5310 PRINT #1:FN$;" ";LN$
5320 PRINT #1:AD$
5330 PRINT #1:CT$;" ";ST$;" ";ZC
5340 PRINT #1:AC$;"-";FN$
5350 PRINT #1:BX$;BM$;" ";BD$
5360 PRINT #1:"-----": ;
5370 CLOSE #1
5380 ON P GOTO 5030,6030,7020,8030,9030,10030
6000 GOSUB 11000
6005 RESTORE
6010 INPUT "City? ":TC$
6020 FOR Z=1 TO 1000
6030 GOSUB 500
6070 IF TC$=CT$ THEN 6100
6080 NEXT Z
6100 GOSUB 11500
6200 GOTO 5200
7000 GOSUB 11000
7005 RESTORE
7010 INPUT "State? (2 letters) ":TS$
7015 FOR Z=1 TO 1000
7020 GOSUB 500
7070 IF TS$=ST$ THEN 7100
7080 NEXT Z
7100 GOSUB 11500
7200 GOTO 5200
8000 GOSUB 11000
8005 RESTORE
8010 INPUT "Zip? (1st digit) ":CZ
8020 FOR Z=1 TO 1000
8030 GOSUB 500
8080 ZZ=ZC
8090 ZZ=ZZ/10000
8100 ZZ=INT(ZZ)
8110 IF CZ=ZZ THEN 8200
8120 NEXT Z
8200 GOSUB 11500
8250 GOTO 5200
9000 GOSUB 11000
9005 RESTORE
9010 INPUT "Area code? ":CA$
9020 FOR Z=1 TO 1000
9030 GOSUB 500
9090 IF AC$=CA$ THEN 9200
9100 NEXT Z
9200 GOSUB 11500
9240 GOTO 5200
10000 GOSUB 11000
10005 RESTORE
10010 INPUT "Month of birth? ":MB$

```

(continued)

```
10020 FOR Z=1 TO 1000
10030 GOSUB 500
10110 IF BM$=MB$ THEN 10200
10120 NEXT Z
10200 GOSUB 11500
10220 GOTO 5200
11000 CALL CLEAR
11010 RETURN
11500 PRINT "-----"
11510 RETURN
12000 GOSUB 11000
12010 PRINT TAB(12);"End."
12020 END
```

When the program execution is started, it first displays some basic information:

This is a simple
random-access data-base
program.

Program lines available
for data entry are
lines 1000 through 4999.

Do you want the data sent
to the line printer? (Y/N)

The data stored are
name, address, phone number.
You may retrieve data by:

- 1 Last name
- 2 City
- 3 State (2 letters)
- 4 Zip (1st digit)
- 5 Area code
- 6 Birth month

Pick one

After that the program will ask for the appropriate input (use only uppercase letters). When that has been done it will display, and if called upon print, all the complete records of persons who match the input data. Some sample printouts are shown in Figs. 9-1A thru 9-1F.

```
PHIL JONES
P.O.BOX 1246
SANTA FE,NM 87501
505-555 2374
Birth date: 2 11
-----

PETER JONES
55 PARK AVE.
LOS ANGELES,CA 90027
213-555 8871
Birth date: 1 22
-----
```

Figure 9-1A. The printout that results when data are selected by last name.

```
JANE DOE
65 BETA WAY
LOS ANGELES,CA 90027
213-555 9611
Birth date: 7 24
-----

PETER JONES
55 PARK AVE.
LOS ANGELES,CA 90027
213-555 8871
Birth date: 1 22
-----
```

Figure 9-1B. Here data are selected by the city.

JANE DOE
65 BETA WAY
LOS ANGELES, CA 90027
213-555 9611
Birth date: 7 24

PETER JONES
55 PARK AVE.
LOS ANGELES, CA 90027
213-555 8871
Birth date: 1 22

ELLEN CARTER
666 MARKET ST.
SAN FRANCISCO, CA 94132
415-555 9876
Birth date: 7 4

DR. JOHN GORDON
45 BEVERLY DR.
BEVERLY HILLS, CA 90210
213-555 4519
Birth date: 4 23

Figure 9-1C. Data selection based on the two-letter state identification.

JOHN DOE
321 MAIN STREET
MIAMI, FL 31011
305-555 3578
Birth date: 12 21

Figure 9-1D. Only one name is associated with a zip code starting with 3.

JIMMY CARTER
45 2ND STREET
BOSTON, MA 3021
617-555 6492
Birth date: 3 16

Figure 9-1E. Here, too, only one name is selected based on the area code 617.

```
PETER JONES
55 PARK AVE.
LOS ANGELES,CA 90027
213-555 8871
Birth date: 1 22
-----
```

```
FRANK GORDON
882 CENTRAL AV.
NEW YORK,NY 10015
212-555 5518
Birth date: 1 7
-----
```

Figure 9-1F. Selection based on the month of birth.

Looking at the program line by line:

Lines 100-120 identify the program.

Line 150 assigns a string to the string variable BX\$.

Lines 200-420 display the menu.

Line 430 sends the computer to a line number determined by your selection.

Lines 500 and 510 are a subroutine that READs 10 DATA items (one complete record), assigning each item to a string variable with the exception of the zip code, which is assigned to a numeric variable.

Lines 1000-4999 are the lines available for record data entry, with each record consisting of 10 items or "fields."

Line 5000 calls up a subroutine to clear the screen.

Line 5005 uses the RESTORE command to make sure that the DATA lines are READ from the beginning.

Line 5010 asks you to input the last name of the persons you want the computer to look for.

Lines 5020-5050 are a loop that causes the computer to READ the DATA items (using the subroutine in lines 500 and 510) until it finds a name that matches the one you have keyed in. That's why it is important to use only uppercase letters, because if the name in the DATA line were written in uppercase, and the one you typed in is written in lowercase, the computer would not recognize them as being identical. If a match is found (IF NL\$=LN\$), the computer is sent to line 5100.

Lines 5100-5240 first place a dashed line into display and then display the entire record in five lines.

Line 5250 checks whether you want the information sent to the line printer, in which case the computer is sent to lines 5300-5370.

Line 5260 places some blank lines into display.

Line 5290 is used if the records are *not* to be printed. It sends the computer to one of six lines, depending on the value of P, which represents your original selection.

Line 5300 accesses the line printer.

Lines 5310-5360 print the record data on paper.

Line 5370 disengages the line printer.

Line 5380 is the same as line 5290.

Lines 6000-10220 repeat the above, based on the second through sixth choices from the menu. The only real difference occurs in lines 8080-8100, where the zip code is divided by 10000 in order to be able to separate the first digit from the rest of the number using the INT(ZZ) statement.

Lines 11000-12020 are the purely cosmetic subroutines and the END line.

As is obvious, changes, such as changes of address or phone number, can easily be made by simply retyping the DATA line in question. Furthermore, the same program can be used to store any other type of data, as long as care is taken that the number of items (fields) in each DATA line is always the same. Thus it could be used to record inventory of all manner of products, where the products could be called up by category, or it could be used to record research material or whatever.

ELECTRONIC COOKBOOK PROGRAM

Let's look now at a similar type of random-access program, which we'll use to store favorite recipes from a variety of cookbooks. If you type in the recipes using a fixed formula, you will be able to retrieve them by categories, such as eggs, poultry, beef, pork, and so on, or by the name of the individual recipe. In addition, the program includes a subprogram that can be used to convert weights and measures to the number of persons for whom the recipe is to be prepared.

Before looking at the program in detail, let's see how recipes must be entered. Each recipe is entered in the form of six items in the DATA lines (lines 5000-20000). The first entry must be the category, such as FISH, EGGS, BEEF, and so on, followed by a comma. Next comes the name of the recipe, such as FILLETS AU GRATIN. Then use the next DATA line to enter all ingredients, using *no* commas unless you enclose the whole list in quotation marks, in which case commas are acceptable. The maximum number of characters that can be used for this item is 130 (including blank spaces), or five lines on the screen including line number and the DATA statement. Then use the next DATA lines to enter the instructions. In order to be able to accommodate lengthy instructions, the program provides for a total of three instruction items of up to 130 characters each. Remember that regardless of whether or not all of this space is needed, each recipe must include three instruction fields, even if one or more are blanks. Thus, it should look like this:

```
5000 DATA CATEGORY,NAME
5010 DATA INGREDIENTS
5020 DATA INSTRUCTIONS,INSTRUCTIONS,INSTRUCTIONS
```

where the instructions can be on up to three separate DATA lines. An example of what this looks like is included in the Electronic Cookbook Program in lines 5000-5030.

ELECTRONIC COOKBOOK PROGRAM

An electronic cookbook program.

```
100 REM ELECTRONIC COOKBOOK
110 REM TI99/4A
200 GOSUB 22200
205 GOSUB 22100
210 PRINT "This program stores recipes" ::
220 PRINT "by category, and it includes" ::
225 PRINT "a subprogram to convert" ::
230 PRINT "weights and measures"
231 GOSUB 22100
232 GOSUB 22400
```

(continued)

```
233 GOSUB 22200
350 PRINT "Menu:"
355 GOSUB 22100
360 PRINT 1;"Enter new recipe"
370 PRINT 2;"Find recipe by category"
380 PRINT 3;"Find specific recipe"
390 GOSUB 22100
395 PRINT 4;"Conversion program"
400 GOSUB 22100
405 PRINT 5;"Exit program"
410 GOSUB 22100
415 RESTORE
420 INPUT "          Which? ":WHICH
430 GOSUB 22200
440 ON WHICH GOTO 1000,2000,3000,4000,25000
1000 PRINT "With 'READY' in display," ::
1010 PRINT "type: LIST 5000-20000" ::
1020 GOSUB 22100
1030 PRINT "Then enter recipes in the" ::
1040 PRINT "DATA lines using the format:"
1050 GOSUB 22100
1060 PRINT "DATA CATEGORY,NAME,INGREDI- ENTS,INSTRUCTIONS,
      INSTRUCT- IONS,INSTRUCTIONS"
1070 GOSUB 22100
1080 PRINT "Limit instructions to three" ::
1090 PRINT "DATA items." ::
1100 PRINT ::
1110 PRINT "Use as many DATA lines as" ::
1120 PRINT "are needed"
1130 GOSUB 22100
1140 GOTO 25040
2000 INPUT "Category? ":CATEG$
2010 RESTORE
2020 FOR PASS=1 TO 25
2025 ON ERROR 233
2030 READ R$
2040 IF R$=CATEG$ THEN 2500
2045 READ N$
2046 READ I$
2047 READ D1$,D2$,D3$
2050 NEXT PASS
2500 READ N$
2510 READ I$,D1$,D2$,D3$
2511 PRINT ::
2512 PRINT ::
2520 PRINT N$
2525 GOSUB 22100
2530 PRINT I$
2540 GOSUB 22100
2541 PRINT D1$;D2$;D3$
2542 GOSUB 22100
2543 PRINT ::
```

(continued)

```

2550 INPUT "Printout? (Y/N) ":PR$
2560 IF PR$="Y" THEN GOSUB 2700
2570 IF WHICH=2 THEN 2030
2580 IF WHICH=3 THEN 3030
2700 OPEN #1:"RS232"
2710 PRINT #1:N$
2720 PRINT #1:"-----"
2730 PRINT #1:I$
2740 PRINT #1:"-----"
2745 PRINT #1:D1$;D2$;D3$
2746 PRINT #1:"-----"
2750 CLOSE #1
2760 RETURN
3000 INPUT "Recipe name? ":RN$
3010 RESTORE
3020 FOR PASS=1 TO 1000
3025 ON ERROR 233
3030 READ R$
3040 READ N$
3050 IF N$=RN$ THEN 3500
3060 READ I$,D1$,D2$,D3$
3070 NEXT PASS
3500 PRINT ::
3510 PRINT R$
3520 GOSUB 22100
3530 PRINT N$
3540 GOSUB 22100
3550 READ I$
3560 PRINT I$
3570 GOSUB 22100
3575 PRINT D1$;D2$;D3$
3576 GOSUB 22100
3580 PRINT ::
3590 GOTO 2550
4000 INPUT "No.servings in recipe? ":SERV
4010 INPUT "No.servings desired? ":SERV1
4020 SERV2=SERV/SERV1
4030 PRINT "Use decimals for fractions:" ::
4040 PRINT "1/2=.5;1/3=.33;2/3=.67" ::
4045 PRINT "1/4=.25;3/4=.75" ::
4050 INPUT "Quantity to convert? ":QUANT
4052 GOSUB 22100
4060 QUANT1=QUANT/SERV2
4065 QUANT1=INT(QUANT1*100+.5)/100
4070 PRINT QUANT;" for ";SERV1;" servings=";QUANT1
4080 GOSUB 22100
4090 INPUT "Another conversion? (Y/N) ":YN$
4100 IF YN$="Y" THEN 4110 ELSE 4150
4110 GOSUB 22200
4120 GOTO 4000
4150 GOSUB 22200
4160 GOTO 350

```

(continued)

```
5000 DATA FISH,FILLETS AU GRATIN
5010 DATA 1 LB FISH FILLETS.2/3 CUP CANNED CONDENSED SOUP.
      2 TBS MILK.1 CUP COARSE BREAD CRUMBS.2 TBS MELTED
      BUTTER
5020 DATA ARRANGE FILLETS IN BUTTERED BAKING DISH.COMBINE
      SOUP & MILK & HEAT.POUR OVER FISH.SPRINKLE WITH BREAD
      CRUMBS.DRIZZLE WITH MELTED
5030 DATA BUTTER.BAKE UNCOVERED AT 375 DEG.F.FOR 10 TO 15
      MINS.2 TO 3 SERVINGS.,
5040 DATA FISH,NAME1
5050 DATA INGREDIENTS
5060 DATA INSTRUCTIONS,INSTRUCTIONS,INSTRUCTIONS
5070 DATA EGGS,NAME2
5080 DATA INGREDIENTS
5090 DATA INSTRUCTIONS,INSTRUCTIONS,INSTRUCTIONS
5100 DATA EGGS,NAME3
5110 DATA INGREDIENTS
5120 DATA INSTRUCTIONS,INSTRUCTIONS,INSTRUCTIONS
5130 DATA BEEF,NAME4
5140 DATA INGREDIENTS
5150 DATA INSTRUCTIONS,INSTRUCTIONS,INSTRUCTIONS
22100 PRINT "-----"
22110 RETURN
22200 CALL CLEAR
22210 PRINT ::
22220 RETURN
22400 PRINT ::
22410 INPUT "Press >ENTER<  ":Y$
22420 RETURN
25000 GOSUB 22200
25010 GOSUB 22100
25020 PRINT "          End."
25030 GOSUB 22100
25040 END
```

When we run the program, it first displays its purpose and then presents the menu:

```
1) Enter new recipe
2) Find recipe by category
3) Find specific recipe
-----
4) Conversion program
-----
5) Exit program
   Which?
```

Next, depending on your selection, it responds with the appropriate information or requests for input. If the first choice was selected, it displays a reminder of how data must be entered:

With 'READY' in display,
type LIST 5000-20000

Then enter recipes in the
DATA lines using the format:

DATA CATEGORY, NAME, INGREDI-
ENTS, INSTRUCTIONS, INSTRUCT-
IONS, INSTRUCTIONS

Limit instructions to three
DATA items.

Use as many DATA lines as
are needed.

While this is being displayed, the 'READY' statement shows at the bottom of the screen, telling you that the computer is ready to accept input in accordance with the instructions.

If the second choice is selected, the display responds with:

Category?

asking you to type in the category desired. When that has been done, it displays the name of the first recipe in that category, the ingredients, and all instructions, followed by:

Printout (Y/N)

asking if you want the recipe printed by the line printer. If you answer in the negative (N), the display changes to the second recipe in that category, and so on until all recipes in the selected category have been displayed. If you answer in the affirmative (Y), the printer will print the recipe as shown in Figure 9-2 before the display moves on to the next category. Thus, it is possible to have selected recipes printed while others are ignored. When all recipes in the category have been displayed, the computer returns to the menu.

FILLETS AU GRATIN

1 LB FISH FILLETS. 2/3 CUP CANNED CONDENSED SOUP. 2
TBS MILK. 1 CUP COARSE BREAD CR
UMBS. 2 TBS MELTED BUTTER

ARRANGE FILLETS IN BUTTERED BAKING DISH. COMBINE SO
UP & MILK & HEAT. POUR OVER FIS
H. SPRINKLE WITH BREAD CRUMBS. DRIZZLE WITH MELTED
BUTTER. BAKE UNCOVERED AT 375 DEG. F. FOR 10 TO 15 MI
NS. 2 TO 3 SERVINGS.

Figure 9-2. When the printout option is selected, the recipe is printed.

If you select the third choice, the first display asks:

Recipe name?

and when it has been typed, the program searches the DATA lines for a recipe by that name and, when found, places it into display, again asking if you want it printed by the line printer. Either way, the program returns to the menu when printing is completed or when N has been pressed. Be sure to use the exact name (using uppercase letters), because the computer will return to the menu without displaying anything when the typed-in name does not match the stored name exactly.

If the fourth option is selected, the display shows:

**No.servings in recipe?
No.servings desired?
Use decimals for fractions:
1/2=.5;1/3=.33;2/3=.67
1/4=.25;3/4=.75
Quantity to convert?**

and when the three questions in the above have been answered, the display responds with:

X for Y servings= Z

where X is the quantity to be converted, Y is the number of servings desired, and Z is the converted quantity. This is followed by:

Another conversion? (Y/N)

which either repeats the above or returns the computer to the menu.

Line by line, the program functions as follows:

Lines 100 and 110 identify the program.

Lines 200-430 display the purpose of the program and the menu, using several subroutines to clear the screen, display dashed lines, and to ask you to press >ENTER< to continue program execution.

Line 440 sends the computer to the appropriate line number, depending on your choice.

Lines 1000-1140 are used with the first option, displaying the instructions for entering new recipes in the DATA lines. In line 1140 the computer is sent to line 25040, the END line, which causes the "READY" statement to be displayed by actually exiting the program.

Lines 2000-2760 are used with the second option; line 2000 asks you to key in the category. Line 2010 uses the RESTORE command to make sure the DATA items are READ from the beginning. Lines 2020-2050 represent a loop that causes the computer to READ the groups of six DATA items that make up each recipe up to 25 times. That 25 in line 2020 is arbitrary. The number used should be equal to or greater than the number of categories. Line 2025 uses the ON ERROR command to send the computer back to the menu when all DATA have been READ, resulting in a DATA ERROR condition. In line 2030 the first DATA item is READ, and in line 2040 that item is compared to the category you typed in. If they match, the computer is sent to line 2500. If not, it goes on to lines 2045-2050 to READ the remaining five items and then return to line 2020 to go on to the next recipe. Lines 2500 and 2510 READ the next five DATA items associated with the recipe, and lines 2511-2543 cause the name, ingredients, and instructions to be displayed. Line 2550 asks if you want the displayed information sent to the line printer, sending the computer in line 2560 to line 2700 if yes, and to the next line if no. Lines 2570 and 2580 are needed because this section is used with both the second and third options. Line 2700 accesses the line printer and lines 2710-2760 are the subroutine that causes the data to be printed.

Lines 3000-3590 come into play if the third option was selected. Here the number (1000) used for the loop in line 3020 is again an arbitrary number. Any number equal to or greater than the total number of recipes in the DATA lines can be used. The rest is pretty much a duplication of the previous section, except that the computer is asked to READ the first two DATA items and the second one is then compared to the name of the recipe you've typed in. In line 3590 the computer is sent to line 2550 to use the question once more with reference to the line printer and the subroutine that causes the data to be printed.

Lines 4000-4160 are used if the fourth option is selected. Line 4000 asks the number of servings for which the recipe was written.

Line 4010 asks the number of servings to which you want the weights and measures to be converted. Line 4020 divides one by the other. Lines 4030-4050 are a reminder that decimals must be used instead of fractions, displaying the decimal equivalents of frequently used fractions, and then asking you to type in the weight or measure to be converted. Line 4060 performs the conversion calculation. Line 4065 limits the displayed decimals to two. Line 4070 displays the result, and line 4090 asks if you want to perform additional conversions, causing the computer, in lines 4120 and 4160, to be sent to the appropriate line numbers.

Lines 5000-20000 are reserved for the DATA lines. In lines 5000-5020 I have used an actual recipe as an example (see Figure 9-2); the remaining DATA lines are there simply to permit testing the program. Note the comma at the end of line 5020. It must be there because I have used only two instruction items (fields), and the computer is told, via the comma, that the third is blank.

Lines 22100-25040 are subroutines used primarily for cosmetic reasons, and the line that places the word "End." into display, with line 25040 telling the computer that it has reached the end of the program.

SCHEDULE C TAX PROGRAM

Next we'll write a program designed to simplify the task of filling out the Schedule C federal tax form used to report income from a business or profession. Aside from being a useful program, it is an interesting exercise in program writing. See Figure 9-4 on page 204 for a *printout* of the program.

SCHEDULE C TAX PROGRAM

This program simplifies the task of filling out the Schedule C federal tax form.

```
100 REM SCHEDULE C TAX PROGRAM
110 REM TI EXTENDED BASIC
150 D=1
200 GOSUB 22200
210 GOSUB 22100
220 DISPLAY AT(5,1):"This program simplifies"
```

(continued)

```

230 DISPLAY AT(7,1):"the task of filling out"
240 DISPLAY AT(9,1):"the Schedule C tax form"
250 DISPLAY AT(10,1):"-----"
260 PRINT
265 INPUT "Press >ENTER<  ":E$
270 GOSUB 22200
300 DISPLAY AT(10,1):1;" Enter receipts"
310 DISPLAY AT(12,1):2;" Enter expenses"
320 DISPLAY AT(13,1):"-----"
330 INPUT "Which?  ":WHICH
340 GOSUB 22200
350 ON WHICH GOTO 500,600
500 INPUT "Enter amount  $":A
510 AA=AA+A
520 GOSUB 22100
530 PRINT TAB(16);"$";AA
540 GOSUB 22100
550 GOSUB 22400
560 GOSUB 22200
570 GOTO 300
600 GOSUB 22200
605 INPUT "Enter line number  ":L$
610 LLLL$=SEG$(L$,1,2)
615 L=VAL(LLLL$)
620 IF L<28 OR L=29 THEN 700 ELSE 630
630 LL$=SEG$(L$,1,2)
640 LLL$=SEG$(L$,3,1)
650 LL=VAL(LL$)
660 INPUT "Deductible amount  $":D
670 IF LL=28 THEN 2000 ELSE 3000
680 GOTO 710
700 INPUT "Deductible amount  $":D
710 ON L-5 GOTO 1060,1070,1080,1090,1100,1110,1120,1130,
    1140,1150,1160,1170,1180,1190,1200,1210,1220,1230,
    1240,1250,1260,1270,1280,1290
1060 D6=D6+D
1061 PRINT TAB(18);"$";D6
1062 GOSUB 22400
1063 GOTO 600
1070 D7=D7+D
1071 PRINT TAB(18);"$";D7
1072 GOTO 1062
1080 D8=D8+D
1081 PRINT TAB(18);"$";D8
1082 GOTO 1062
1090 D9=D9+D
1091 PRINT TAB(18);"$";D9
1092 GOTO 1062
1100 D10=D10+D
1101 PRINT TAB(18);"$";D10
1102 GOTO 1062

```

(continued)

```
1110 D11=D11+D
1111 PRINT TAB(18);"$";D11
1112 GOTO 1062
1120 D12=D12+D
1121 PRINT TAB(18);"$";D12
1122 GOTO 1062
1130 D13=D13+D
1131 PRINT TAB(18);"$";D13
1132 GOTO 1062
1140 D14=D14+D
1141 PRINT TAB(18);"$";D14
1142 GOTO 1062
1150 D15=D15+D
1151 PRINT TAB(18);"$";D15
1152 GOTO 1062
1160 D16=D16+D
1161 PRINT TAB(18);"$";D16
1162 GOTO 1062
1170 D17=D17+D
1171 PRINT TAB(18);"$";D17
1172 GOTO 1062
1180 D18=D18+D
1181 PRINT TAB(18);"$";D18
1182 GOTO 1062
1190 D19=D19+D
1191 PRINT TAB(18);"$";D19
1192 GOTO 1062
1200 D20=D20+D
1201 PRINT TAB(18);"$";D20
1202 GOTO 1062
1210 D21=D21+D
1211 PRINT TAB(18);"$";D21
1212 GOTO 1062
1220 D22=D22+D
1221 PRINT TAB(18);"$";D22
1222 GOTO 1062
1230 D23=D23+D
1231 PRINT TAB(18);"$";D23
1232 GOTO 1062
1240 D24=D24+D
1241 PRINT TAB(18);"$";D24
1242 GOTO 1062
1250 D25=D25+D
1251 PRINT TAB(18);"$";D25
1252 GOTO 1062
1260 D26=D26+D
1261 PRINT TAB(18);"$";D26
1262 GOTO 1062
1270 D27=D27+D
1271 PRINT TAB(18);"$";D27
1272 GOTO 1062
1290 D29=D29+D
```

(continued)

```

1291 PRINT TAB(18);"$";D29
1292 GOTO 1062
1300 D28A=D28A+D
1301 PRINT TAB(18);"$";D28A
1302 GOTO 1062
1310 D28B=D28B+D
1311 PRINT TAB(18);"$";D28B
1312 GOTO 1062
1320 D28C=D28C+D
1321 PRINT TAB(18);"$";D28C
1322 GOTO 1062
1330 D30A=D30A+D
1331 PRINT TAB(18);"$";D30A
1332 GOTO 1062
1340 D30B=D30B+D
1341 PRINT TAB(18);"$";D30B
1342 GOTO 1062
1350 D30C=D30C+D
1351 PRINT TAB(18);"$";D30C
1352 GOTO 1062
1360 D30D=D30D+D
1361 PRINT TAB(18);"$";D30D
1362 GOTO 1062
1370 D30E=D30E+D
1371 PRINT TAB(18);"$";D30E
1372 GOTO 1062
1380 D30F=D30F+D
1381 PRINT TAB(18);"$";D30F
1382 GOTO 1062
1390 D30G=D30G+D
1391 PRINT TAB(18);"$";D30G
1392 GOTO 1062
1400 D30H=D30H+D
1401 PRINT TAB(18);"$";D20H
1402 GOTO 1062
1410 D30I=D30I+D
1411 PRINT TAB(18);"$";D30I
1412 GOTO 1062
1420 D30J=D30J+D
1421 PRINT TAB(18);"$";D30J
1422 GOTO 1062
1430 D30K=D30K+D
1431 PRINT TAB(18);"$";D30K
1432 GOTO 1062
1440 D30L=D30L+D
1441 PRINT TAB(18);"$";D30L
1442 GOTO 1062
1450 D30M=D30M+D
1451 PRINT TAB(18);"$";D30M
1452 GOTO 1062
2000 IF LLL$="A" THEN 1300
2010 IF LLL$="B" THEN 1310

```

(continued)

```
2020 IF LLL$="C" THEN 1320
3000 IF LLL$="A" THEN 1330
3010 IF LLL$="B" THEN 1340
3020 IF LLL$="C" THEN 1350
3030 IF LLL$="D" THEN 1360
3040 IF LLL$="E" THEN 1370
3050 IF LLL$="F" THEN 1380
3060 IF LLL$="G" THEN 1390
3070 IF LLL$="H" THEN 1400
3080 IF LLL$="I" THEN 1410
3090 IF LLL$="J" THEN 1420
3100 IF LLL$="K" THEN 1430
3110 IF LLL$="L" THEN 1440
3120 IF LLL$="M" THEN 1450
4000 DT=D6+D7+D8+D9+D10+D11+D12+D13+D14+D15+D16+D17+D18+
    D19+D20+D21+D22+D23+D24+D2325+D26+D27+D28A+D28B+D28C+
    D29+D30A+D30B+D30C+D30D+D30E
4005 DT=DT+D30F+D30G+D30H+D30I+D30J+D30K+D30L+D30M
4010 GOSUB 22200
4020 PRINT "1a Gross receipts & sales.....$";AA
4030 INPUT " b Returns & allowances?.....$":RA
4040 BA=AA-RA
4050 PRINT " c Balance.....$";BA
4055 INPUT "2 Cost of goods sold?.....$":CG
4060 GP=BA-CG
4070 PRINT "3 Gross profit.....$";GP
4080 INPUT "4a Windfall prof.tax credit?....$":WP
4090 INPUT " b Other income?.....$":OI
4095 TI=OI+WP+GP
5000 PRINT "5 Total income.....$";TI
5010 GOSUB 22100
5020 PRINT "31 Total deductions.....$";DT
5030 NP=TI-DT
5040 PRINT "32 Net profit or loss.....$";NP
5050 GOSUB 22100
5060 INPUT "Printout? (Y/N) ":YN$
5070 IF YN$<>"Y" THEN 25000
6000 Q$="Line number..."
6001 W$="Total.....$"
6002 V$="...."
6005 OPEN #1:"RS232"
6010 PRINT #1:Q$;6;V$;W$;D6
6020 PRINT #1:Q$;7;V$;W$;D7
6030 PRINT #1:Q$;8;V$;W$;D8
6050 PRINT #1:Q$;9;V$;W$;D9
6060 PRINT #1:Q$;10;V$;W$;D10
6070 PRINT #1:Q$;11;V$;W$;D11
6080 PRINT #1:Q$;12;V$;W$;D12
6090 PRINT #1:Q$;13;V$;W$;D13
6100 PRINT #1:Q$;14;V$;W$;D14
6110 PRINT #1:Q$;15;V$;W$;D15
6120 PRINT #1:Q$;16;V$;W$;D16
```

(continued)

```

6130 PRINT #1:Q$,17;V$;W$;D17
6140 PRINT #1:Q$,18;V$;W$;D18
6150 PRINT #1:Q$,19;V$;W$;D19
6160 PRINT #1:Q$,20;V$;W$;D20
6170 PRINT #1:Q$,21;V$;W$;D21
6180 PRINT #1:Q$,22;V$;W$;D22
6190 PRINT #1:Q$,23;V$;W$;D23
6200 PRINT #1:Q$,24;V$;W$;D24
6210 PRINT #1:Q$,25;V$;W$;D25
6220 PRINT #1:Q$,26;V$;W$;D26
6230 PRINT #1:Q$,27;V$;W$;D27
6240 PRINT #1:Q$;"28A";V$;W$;D28A
6250 PRINT #1:Q$;"28B";V$;W$;D28B
6260 PRINT #1:Q$;"28C";V$;W$;D28C
6270 PRINT #1:Q$,29;V$;W$;D29
6280 PRINT #1:Q$;"30A";V$;W$;D30A
6290 PRINT #1:Q$;"30B";V$;W$;D30B
6300 PRINT #1:Q$;"30C";V$;W$;D30C
6310 PRINT #1:Q$;"30D";V$;W$;D30D
6320 PRINT #1:Q$;"30E";V$;W$;D30E
6330 PRINT #1:Q$;"30F";V$;W$;D30F
6340 PRINT #1:Q$;"30G";V$;W$;D30G
6350 PRINT #1:Q$;"30H";V$;W$;D30H
6360 PRINT #1:Q$;"30I";V$;W$;D30I
6370 PRINT #1:Q$;"30J";V$;W$;D30J
6380 PRINT #1:Q$;"30K";V$;W$;D30K
6390 PRINT #1:Q$;"30L";V$;W$;D30L
6400 PRINT #1:Q$;"30M";V$;W$;D30M
6410 PRINT #1:"-----"
      "
6420 PRINT #1:Q$;"1a Gross receipts or sales.....$";
      AA
6430 PRINT #1:Q$;" b Returns & allowances.....$";
      RA
6440 PRINT #1:Q$;" c Balance.....$";
      BA
6450 PRINT #1:Q$;"2 Cost of goods sold.....$";
      CG
6460 PRINT #1:Q$;"3 Gross profit.....$";
      GP
6470 PRINT #1:Q$;"4a Windfall prof.tax credit.....$";
      WP
6480 PRINT #1:Q$;" b Other income.....$";
      OI
6490 PRINT #1:Q$;"5 Total income.....$";
      TI
6500 PRINT #1:"-----"
      "
6510 PRINT #1:Q$;"31 Total deductions.....$";
      DT
6520 PRINT #1:Q$;"32 Net profit or loss.....$";
      NP

```

(continued)

```

6525 PRINT #1:"-----"
      "-----"
6530 CLOSE #1
6540 GOTO 25000
22100 PRINT "-----"
22110 RETURN
22200 CALL CLEAR
22210 RETURN
22400 PRINT
22410 PRINT "After last entry,enter 0."
22420 PRINT
22430 PRINT "To exit program, press X. "
22440 PRINT
22450 INPUT "To continue, press >ENTER< ":Y$
22460 IF Y$="X" THEN 25000
22470 IF D=0 THEN 4000 ELSE RETURN
25000 GOSUB 22100
25010 PRINT TAB(12);"End."
25020 GOSUB 22100
25030 END

```

Initially the program displays its purpose, followed by the choice:

```

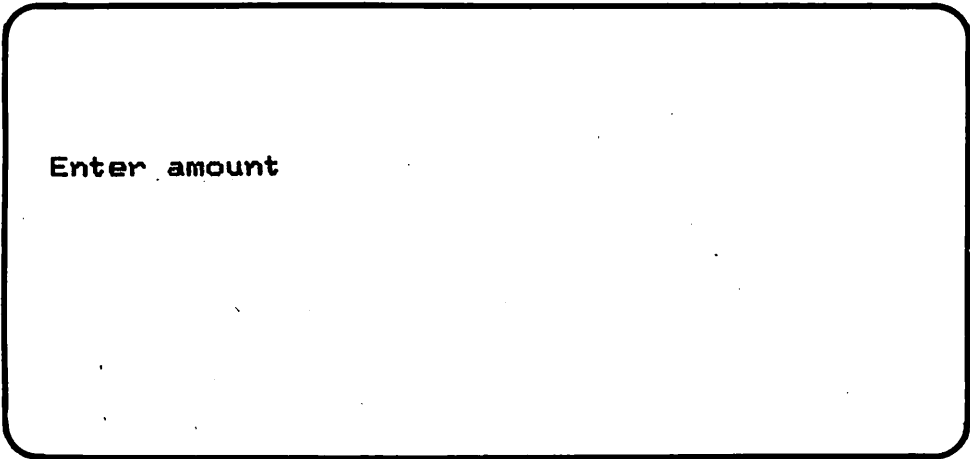
1  Enter receipts
2  Enter expenses
-----

```

Which?

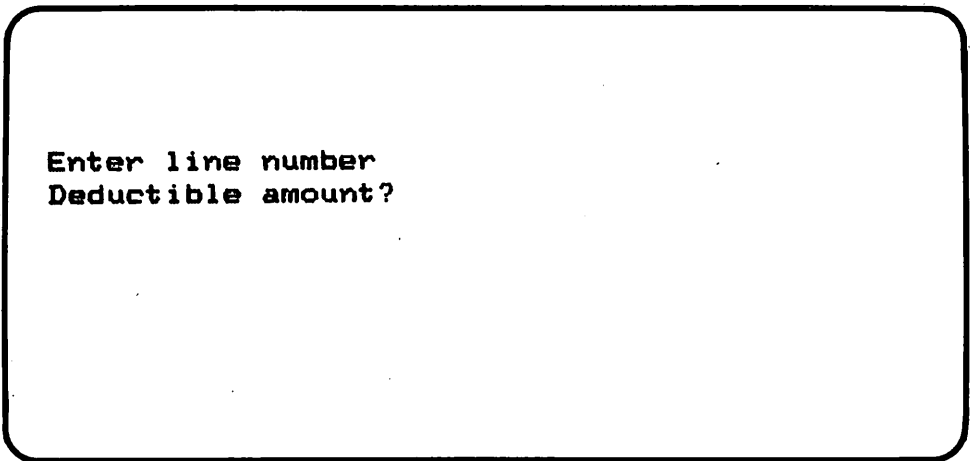
It is important to enter all receipts first, because once expenses are selected, the program, as written, will not return to the above menu. That can be changed by changing the GOTO 600 in line 1063 to GOTO 270. The drawback would be the fact that it adds additional

key strokes during program execution, because the choice between receipts and expenses would have to be made prior to each data entry. When "receipts" is selected, the display asks:



Enter amount

and when "expenses" is selected, the request is:



Enter line number
Deductible amount?

The program internally adds up the receipts and the amounts typed in for each line number. In order to enter the expenses for the correct line numbers, have a blank Schedule C form (as shown in Figure 9-3

SCHEDULE C
(Form 1040)
Department of the Treasury
Internal Revenue Service (O)

Profit or (Loss) From Business or Profession
(Sole Proprietorship)
Partnerships, Joint Ventures, etc., Must File Form 1065.
▶ Attach to Form 1040 or Form 1041. ▶ See Instructions for Schedule C (Form 1040).

OMB. No. 1545-0074
1982
08

Name of proprietor

Social security number of proprietor

A Main business activity (see Instructions) ▶ ; product ▶

B Business name ▶

C Employer identification number

D Business address (number and street) ▶
City, State and ZIP Code ▶

E Accounting method: (1) ☐ Cash (2) ☐ Accrual (3) ☐ Other (specify) ▶

F Method(s) used to value closing inventory:
(1) ☐ Cost (2) ☐ Lower of cost or market (3) ☐ Other (if other, attach explanation)

G Was there any major change in determining quantities, costs, or valuations between opening and closing inventory? . . .
If "Yes," attach explanation.

H Did you deduct expenses for an office in your home?

I Did you operate this business at the end of 1982?

J How many months in 1982 did you actively operate this business? ▶

Part I Income

1 a Gross receipts or sales	1a				
b Returns and allowances	1b				
c Balance (subtract line 1b from line 1a)	1c				
2 Cost of goods sold and/or operations (Schedule C-1, line 8)	2				
3 Gross profit (subtract line 2 from line 1c)	3				
4 a Windfall Profit Tax Credit or Refund received in 1982 (see Instructions)	4a				
b Other income	4b				
5 Total income (add lines 3, 4a, and 4b)	5				

Part II Deductions.

6 Advertising		25 Taxes (Do not include Windfall Profit Tax here. See line 29.) . .	
7 Bad debts from sales or services (Cash method taxpayers, see Instructions)		26 Travel and entertainment . .	
8 Bank service charges		27 Utilities and telephone . .	
9 Car and truck expenses		28 a Wages . .	
10 Commissions		b Jobs credit	
11 Depletion		c Subtract line 28b from 28a .	
12 Depreciation, including Section 179 expense deduction (from Form 4562)		29 Windfall Profit Tax withheld in 1982 . .	
13 Dues and publications		30 Other expenses (specify):	
14 Employee benefit programs		a	
15 Freight (not included on Schedule C-1)		b	
16 Insurance		c	
17 Interest on business indebtedness		d	
18 Laundry and cleaning		e	
19 Legal and professional services		f	
20 Office supplies and postage		g	
21 Pension and profit-sharing plans		h	
22 Rent on business property		i	
23 Repairs		j	
24 Supplies (not included on Schedule C-1)		k	
		l	
		m	
31 Total deductions (add amounts in columns for lines 6 through 30m)	31		
32 Net profit or (loss) (subtract line 31 from line 5). If a profit, enter on Form 1040, line 12, and on Schedule SE, Part I, line 2 (or Form 1041, line 6). If a loss, go on to line 33.	32		
33 If you have a loss, do you have amounts for which you are not "at risk" in this business (see Instructions)? . . <input type="checkbox"/> Yes <input type="checkbox"/> No If you checked "No," enter the loss on Form 1040, line 12, and on Schedule SE, Part I, line 2 (or Form 1041, line 6).			

For Paperwork Reduction Act Notice, see Form 1040 Instructions.

Figure 9-3. Federal Form 1040 Schedule C. Our program uses actual tax form line numbers, and deductible expenses must be entered by line number. Each year you would have to check, and possibly change, some line numbers if the form is revised.

on page 202) available and use it to determine the appropriate line number. Each time an entry is made, the program displays the total for receipts of any specific line number up to that point. After the last entry has been made, enter 0 to let the computer know that no further entries will be made. At that point the display responds with:

```

Line number      1a Gross receipts or sales . . . $xxxx.xx
Line number      b Returns & allowances? . . . . . $
Line number      c Balance . . . . . $xxxx.xx
Line number      2 Cost of goods sold? . . . . . $
Line number      3 Gross profit . . . . . $xxxx.xx
Line number      4a Windfall prof.tax credit? . . . $
Line number      b Other income? . . . . . $
Line number      5 Total income . . . . . $xxxx.xx
-----
Line number      31 Total deductions . . . . . $xxxx.xx
Line number      32 Net profit or loss. . . . . $xxxx.xx
-----
Printout? (Y/N)

```

In most instances the answer should be in the affirmative, producing the printout of all totals (see Figure 9-4 on page 204), and all that is left to do is to enter the data in the printout into the actual tax form.

If the program appears rather long, it is because of all the line numbers, which must be dealt with one at a time in order to obtain the totals for each.

Lines 100 and 110 identify the program, and line 150 assigns the value of 1 to the numeric variable D, because whenever the computer finds that the value of D is zero, it assumes that no further entries are to be made.

Lines 200-265 display the purpose of the program. The DISPLAY AT command is available only in TI EXTENDED BASIC and if that version is not available, it must be replaced by PRINT commands.

Lines 300-340 present the choice, again using the DISPLAY AT command.

Line 350 sends the computer to either of two lines, depending on the selected option.

Lines 500-570 are used to enter receipts. Line 510 adds entered data to previously entered data, assigning the resulting sums to the numeric variable AA.

```

Line number... 6 ....Total.....$ 2387
Line number... 7 ....Total.....$ 0
Line number... 8 ....Total.....$ 0
Line number... 9 ....Total.....$ 3116
Line number... 10 ....Total.....$ 4570
Line number... 11 ....Total.....$ 0
Line number... 12 ....Total.....$ 0
Line number... 13 ....Total.....$ 428
Line number... 14 ....Total.....$ 0
Line number... 15 ....Total.....$ 0
Line number... 16 ....Total.....$ 362
Line number... 17 ....Total.....$ 0
Line number... 18 ....Total.....$ 0
Line number... 19 ....Total.....$ 1000
Line number... 20 ....Total.....$ 1374
Line number... 21 ....Total.....$ 0
Line number... 22 ....Total.....$ 6000
Line number... 23 ....Total.....$ 0
Line number... 24 ....Total.....$ 0
Line number... 25 ....Total.....$ 0
Line number... 26 ....Total.....$ 3786
Line number... 27 ....Total.....$ 1863
Line number...28A....Total.....$ 0
Line number...28B....Total.....$ 0
Line number...28C....Total.....$ 0
Line number... 29 ....Total.....$ 0
Line number...30A....Total.....$ 450
Line number...30B....Total.....$ 763
Line number...30C....Total.....$ 0
Line number...30D....Total.....$ 0
Line number...30E....Total.....$ 0
Line number...30F....Total.....$ 0
Line number...30G....Total.....$ 0
Line number...30H....Total.....$ 0
Line number...30I....Total.....$ 0
Line number...30J....Total.....$ 0
Line number...30K....Total.....$ 0
Line number...30L....Total.....$ 0
Line number...30M....Total.....$ 0
-----
Line number...1a Gross receipts or sales...$ 45700
Line number... b Returns & allowances.....$ 0
Line number... c Balance.....$ 45700
Line number...2 Cost of goods sold.....$ 5611
Line number...3 Gross profit.....$ 40089
Line number...4a Windfall prof.tax credit...$ 0
Line number... b Other income.....$ 0
Line number...5 Total income.....$ 40089
-----
Line number...31 Total deductions.....$ 26099
Line number...32 Net profit or loss.....$ 13990
-----

```

Figure 9-4. A sample printout of the tax program.

Line 570 returns the computer to the menu for another selection.

Lines 600-680 are used if you want to key in deductible expenses.

When the line number is keyed in, it is assigned to the string variable L\$ because several of the numbers use letters (A, B, C, and so on) which cannot be assigned to numeric variables. In line 610 the first two characters of the line number (the digits) are assigned to the string variable LLLL\$. Line 615 assigns the numeric value of LLLL\$ to the numeric variable L. Line 620 checks whether the value of L is smaller than 28 or equal to 29, because lines 28 and 30 are the ones that use alpha characters along with the digits, sending the computer to either of two line numbers. Lines 630-650 once more separate the digits from the alpha characters. Line 660 asks you to key in the deductible amount. Line 670 sends the computer to either of two line numbers if the value of LL is either 28 or 30.

Line 700 asks for input if the selected line number is other than either 28 or 30.

Line 710 checks the value of L-5, because the lowest line number to be used is 6. It then sends the computer to the line number that is designed to deal with the selected Schedule C line number.

Lines 1060-1452 add the keyed-in amount to previously entered amounts for each of the line numbers and cause the current total for that number to be displayed. After that, upon pressing >ENTER<, the program returns to line 600.

Lines 2000-3120 use the values assigned to LLL\$ (the alpha characters) to determine the appropriate 28 or 30 line number.

Lines 4000 and 4005 add the totals from all line numbers to arrive at total deductible expenses.

Lines 4010-5050 print various totals and ask you to key in answers to a number of questions.

Lines 5060 and 5070 ask if you want a printout, causing the computer to go to either one of two line numbers.

Lines 6000-6002 assign three strings to string variables that will subsequently be used by the line printer.

Line 6005 accesses the line printer.

Lines 6010-6525 cause all totals to be sent to the line printer.

Line 6530 turns the line printer off.

Line 6540 sends the computer to the END line.

Lines 22100-25030 are several subroutines that are used repeatedly throughout the program. Line 22460 checks whether X was

pressed in order to exit the program, and line 25470 checks whether 0 was pressed to indicate that no further entries will be made.

KITCHEN TIMER PROGRAM

In Chapter 3 we created a stopwatch program as an exercise in program writing. Now let's take that same program and add a few steps in order to convert it into a kitchen timer. In the new version you're asked to key in answers to these questions

Timing speed?
Stop after minutes.seconds?
Length of tone? (1000=1 sec.)
Tone? (110-44733)
Volume (0=loud, 30=soft)
To start press >ENTER<

The timing speed should be about 100 for actual time. The stop command should be entered with a decimal point between minutes and seconds (1.25). The length of tone refers to the number of seconds for which the alarm will sound when the keyed-in time has been reached. If you're likely to be away from the room in which the computer is located, you might want to enter a longer period and a louder volume than you would otherwise. Each 1000 produces an alarm lasting about 1 second. Tone refers to the pitch. You might want to experiment until you find a pitch you like. Volume refers to the loudness, assuming that the volume on your monitor is turned full up. Again, a bit of experimentation might be in order.

KITCHEN TIMER PROGRAM

A program that converts your computer into a kitchen timer.

```

100 MM$="Minute"
110 MMM$="Minutes"
120 SS$="Second"
130 SSS$="Seconds"
140 CALL CLEAR
150 FOR X=1 TO 10
160 PRINT
170 NEXT X
180 PRINT TAB(12);"-----"
190 PRINT TAB(12);"Clock"
200 PRINT TAB(12);"-----"
210 PRINT
220 PRINT
230 INPUT "Timing speed? ":SPEED
240 PRINT
250 PRINT
260 INPUT "Stop after minutes.seconds? ":TIME
270 PRINT
280 PRINT
290 INPUT "Length of tone (1000=1 sec.) ":DUR
300 INPUT "Tone (110-44733) ":TONE
310 INPUT "Volume (0=loud,30=soft) ":VOL
320 INPUT "To start press >ENTER< ":START$
330 FOR X=1 TO 4
340 PRINT
350 NEXT X
360 FOR PAUSE=1 TO SPEED
370 NEXT PAUSE
380 A=A+1
390 IF A=60 THEN 520
400 IF A>1 THEN 430 ELSE 410
410 S$=SS$
420 GOTO 440
430 S$=SSS$
440 IF B=1 THEN 470 ELSE 450
450 M$=MMM$
460 GOTO 480
470 M$=MM$
480 PRINT B;M$;" and ";A;S$
490 HALT=B+(A/100)
500 IF HALT=TIME THEN 650
510 GOTO 330
520 A=0
530 B=B+1
540 FOR X=1 TO 4
550 PRINT
560 NEXT X

```

(continued)

```
570 IF B=1 THEN 580 ELSE 600
580 M$=MM$
590 GOTO 610
600 M$=MMM$
610 PRINT B;M$;" and";A;S$
620 HALT=B+(A/100)
630 IF HALT=TIME THEN 650
640 GOTO 330
650 CALL SOUND(DUR,TONE,VOL)
660 PRINT
670 PRINT
680 PRINT
690 PRINT "To continue, type CON"
700 BREAK
710 CALL CLEAR
720 INPUT "Stop after minutes.seconds? ":TIME
730 GOTO 510
```

After that the minutes and seconds are displayed as they scroll up the screen until the keyed-in stop time has been reached, at which point the display shows:

To continue type CON
Stop after minutes.seconds?
Breakpoint at (line number)

You can now do whatever needs to be done and, when you're ready, type in the next point at which you want the timing process to stop. Then type CON and the program will continue the timing process

where it left off. If you'd prefer to have it start again at zero minutes and seconds, you'll have to add four lines;

```
635 A=0
636 B=0
725 A=0
726 B=0
```

though I believe that cumulative timing is preferable despite the fact that for the second and subsequent timing periods you'll always have to add the minutes and seconds that have already elapsed to the next timing period. There is still another alternative that would overcome that problem:

```
260 INPUT "Stop after minutes.seconds? ":TIMEX
265 TIME=TIME+TIMEX
720 INPUT "Stop after minutes.seconds? ":TIMEX
725 TIME=TIME+TIMEX
```

in which case the computer internally adds the keyed-in time period to any previously keyed-in time periods, making it unnecessary for you to perform that calculation.

The program seems to require no special explanations, with the possible exception of lines 490 and 620, where the minute and second figures that are displayed are converted to the format in which the time limits are keyed in by dividing the seconds by 100 and then adding them to the minutes. In line 700 the command BREAK is used to stop program execution until you use the CONTINUE command then restart the program.

HOUSEHOLD BUDGET PROGRAM

The last program in this group deals with budgeting the expenses associated with a home. It can be used for any size family and for any income level, and its primary purpose is to determine how much, if anything, is left over each week for fun and games after all annual, monthly, and weekly expenses have been taken care of.

HOUSEHOLD BUDGET PROGRAM

A household budget program.

```

100 REM HOUSEHOLD BUDGET
110 REM TI99/4A
120 GOSUB 690
130 GOSUB 670
140 PRINT "This program is designed to "
150 PRINT "determine the weekly amount "
160 PRINT "available for fun and games "
170 GOSUB 670
180 GOSUB 710
190 GOSUB 750
200 GOSUB 690
210 INPUT "Annual net salary or income?          $":AI
220 INPUT "Annual interest income?              $":IV
230 GOSUB 670
240 INPUT "Monthly rent/house payments?         $":MR
250 INPUT "Monthly car payments?                $":CP
260 INPUT "Other monthly installments?          $":OM
270 INPUT "Annual insurance premiums?           $":IP
280 INPUT "Annual medical/dental costs?          $":MD
290 INPUT "Gas,electricity,water/month?         $":UT
300 INPUT "Average phone bill/month?            $":TP
310 INPUT "Average transportation/month?        $":MT
320 INPUT "Union dues per year?                 $":UY
330 INPUT "Other fixed expenses/month?          $":FE
340 MI=(IV/12)+(AI/12)
350 MI=INT(MI*100+.5)/100
360 ME=MR+UT+TP+MT+MU+FE+CP+OM+(IP/12)+(MD/12)
370 ME=INT(ME*100+.5)/100
380 DC=MI-ME
390 GOSUB 690
400 GOSUB 670
410 PRINT "Income per month=                      $":MI
420 PRINT "Fixed expenses per month=             $":ME
430 PRINT "Left over per month=                     $":DC
440 GOSUB 670
450 GOSUB 710
460 GOSUB 750
470 GOSUB 690
480 INPUT "Average food bill per week?          $":WF
490 INPUT "Average cleaning per week?           $":WC
500 INPUT "Average clothing per year?           $":CY
510 INPUT "Average other costs/week?            $":EW
520 WEEK=WF+WC+((CY/365.25)*7)+EW
530 FG=((DC/30.44)*7)-WEEK
540 WEEK=INT(WEEK*100+.5)/100
550 FG=INT(FG*100+.5)/100
560 GOSUB 670
570 GOSUB 750

```

(continued)

```

580 GOSUB 690
590 GOSUB 670
600 PRINT "Regular weekly living costs=      $";WEEK

610 PRINT
620 PRINT "Left over for fun and games=      $";FG
630 GOSUB 670
640 GOSUB 710
650 GOSUB 750
660 GOTO 780
670 PRINT "-----"
680 RETURN
690 CALL CLEAR
700 RETURN
710 FOR X=1 TO 10
720 PRINT
730 NEXT X
740 RETURN
750 PRINT
760 INPUT "Press >ENTER< ":Y$
770 RETURN
780 GOSUB 690
790 GOSUB 670
800 PRINT TAB(12);"End."
810 GOSUB 670
820 GOSUB 710
830 END

```

After displaying its purpose, the Household Budget program requires that you type in the answers to a long list of questions:

Annual net salary or income?
 Annual interest income?

 Monthly rent/house payments?
 Monthly car payments?
 Other monthly installments?
 Annual insurance premiums?
 Annual medical/dental costs?
 Gas,electricity,water/month?
 Average phone bill/month?
 Average transportation/month?
 Union dues per year?
 Other fixed expenses/month?

where annual net income refers to income after taxes. Interest income, if any, is for interest from investments. Transportation cost per month refers to necessary transportation, such as going to and from work, taking kids to school, and going shopping for food or other necessities.

When these questions have been answered, the program displays:

```
Income per month=          $xxxx.xx
Fixed expenses per month=  $xxxx.xx
Left over per month=       $xxxx.xx
```

after which you're asked to enter averages for living expenses:

```
Average food bill per week?
Average cleaning per week?
Average clothing per year?
Average other costs/week?
```

which then produces the final figures:

```
Regular weekly living costs=  $xxx.xx
Left over for fun and games=  $xxx.xx
```

If the final figure turns out to be negative, it indicates that you're spending in excess of your income and you may have to make adjustments in some of the areas where such adjustments are possible.

Lines 120-190 display the purpose of the program.

Lines 200-330 ask the first series of questions.

Line 340 calculates the monthly income figure.

Line 350 limits the displayed decimals to two.

Line 360 calculates the total fixed expenses per month.

Line 370 again limits the number of displayed decimals to two.

Line 380 calculates the amount left over per month after the fixed monthly expenses have been deducted from the monthly income figure.

Lines 390-470 cause the results of these calculations to be displayed.

Lines 480-510 ask the next series of questions.

Lines 520-530 calculate the weekly living costs and the leftover balance.

Lines 540 and 550 limit the displayed decimals.

Lines 560-650 display the final results.

Line 660 sends the computer to line 780, which clears the screen and then places "End." into display.

Lines 670-830 represent frequently used subroutines and the END sequence.

10

Educational Programs

Computers, when equipped with intelligently written programs, are very useful in motivating kids of all ages to learn. Most likely, one of the reasons is the need for constant interaction with the computer. Another is the fact that computers are extremely patient, allowing the student to make all kinds of mistakes and then to correct those mistakes without embarrassing him or her by commenting in one way or another when a mistake has been made.

We have already discussed a few of the typical educational software programs that are available commercially for the TI-99/4A Home Computer. In this chapter we'll write some original programs designed to deal with a number of different educational subjects. Some are very simple, some less so, and some can be adapted to deal with basic or more advanced subjects. Most are written in TI BASIC, so you can use them regardless of whether your computer is equipped to handle the optional TI EXTENDED BASIC.

SIMPLE ARITHMETIC PROGRAM

Let's start with a mathematics program that presents several choices: single- or double-digit figures and addition, subtraction, multiplication, or division. The program presents random numbers and asks the student to type in the result of performing arithmetic operations.

SIMPLE ARITHMETIC PROGRAM

A simple program to practice arithmetic. It uses randomly generated integer numbers.

```
100 REM SIMPLE MATH/TI99/4A
110 CALL CLEAR
120 PRINT "I am TI99/4A. Tell me your"
130 PRINT "name, please."
140 GOSUB 1350
150 INPUT N$
160 CALL CLEAR
170 PRINT "This program is designed"
180 PRINT "to practice arithmetic.": :
190 PRINT "You have a choice of using:"
200 GOSUB 1430
210 PRINT 1;" Single-digit figures"
220 PRINT 2;" Double-digit figures"
230 GOSUB 1430
240 GOSUB 1350
250 INPUT "Which? ":WHICH
260 IF WHICH=1 THEN 290
270 X=100
280 GOTO 300
290 X=10
300 GOSUB 1340
310 PRINT "You can practice:"
320 GOSUB 1430
330 PRINT 1,"Addition"
340 PRINT 2,"Subtraction"
350 PRINT 3,"Multiplication"
360 PRINT 4,"Division"
370 GOSUB 1430
380 GOSUB 1350
390 INPUT "Which would you like? ":WHICH
400 ON WHICH GOTO 410,640,870,1100
410 GOSUB 1340
420 GOSUB 1450
430 NN=N
440 GOSUB 1450
450 NNN=N
460 PRINT ;TAB(5);NN;" + ";NNN;" =?"
470 GOSUB 1430
480 GOSUB 1350
490 INPUT "The sum is .":S
500 SS=NN+NNN
510 IF SS=S THEN 590
520 PRINT
530 PRINT "Sorry, ";N$;", that's wrong,"
540 PRINT "try again."
550 GOSUB 1350
560 GOSUB 1390
```

(continued)


```
570 GOSUB 1340
580 GOTO 460
590 PRINT
600 PRINT "Good, ";N$;" that's right."
610 GOSUB 1350
620 GOSUB 1390
630 GOTO 410
640 GOSUB 1340
650 GOSUB 1450
660 NN=N
670 GOSUB 1450
680 NNN=N
685 IF NNN>NN THEN NNN=NN AND NN=NNN
690 PRINT ;TAB(5);NN;" - ";NNN;" =?"
700 GOSUB 1430
710 GOSUB 1350
720 INPUT "The balance is ":S
730 SS=NN-NNN
740 IF SS=S THEN 820
750 PRINT
760 PRINT "Sorry, ";N$;" that's wrong"
770 PRINT "Try again."
780 GOSUB 1350
790 GOSUB 1390
800 GOSUB 1340
810 GOTO 690
820 PRINT
830 PRINT "Good, ";N$;" that's right."
840 GOSUB 1350
850 GOSUB 1390
860 GOTO 640
870 GOSUB 1340
880 GOSUB 1450
890 NN=N
900 GOSUB 1450
910 NNN=N
920 PRINT ;TAB(5);NN;" * ";NNN;" =?"
930 GOSUB 1430
940 GOSUB 1350
950 INPUT "The result is ":S
960 SS=NN*NNN
970 IF SS=S THEN 1050
980 PRINT
990 PRINT "Sorry, ";N$;" that's wrong."
1000 PRINT "Try again."
1010 GOSUB 1350
```

(continued)

```
1020 GOSUB 1390
1030 GOSUB 1340
1040 GOTO 920
1050 PRINT
1060 PRINT "Good, ";N$;" , that's right."
1070 GOSUB 1350
1080 GOSUB 1390
1090 GOTO 870
1100 GOSUB 1340
1110 GOSUB 1450
1120 NN=N
1130 GOSUB 1450
1140 NNN=N
1150 PRINT ;TAB(5);NN;" / ";NNN;" ="
1160 GOSUB 1430
1170 GOSUB 1350
1180 INPUT "The result is  ":S
1190 PRINT
1200 SS=NN/NNN
1210 SS=INT(SS*10+.5)/10
1220 IF SS=S THEN 1290
1230 PRINT "Sorry, ";N$;" , that's wrong."
1240 PRINT "Try again."
1250 GOSUB 1350
1260 GOSUB 1390
1270 GOSUB 1340
1280 GOTO 1150
1290 PRINT
1300 PRINT "Good, ";N$;" , that's right."
1310 GOSUB 1350
1320 GOSUB 1390
1330 GOTO 1100
1340 CALL CLEAR
1350 FOR XX=1 TO 10
1360 PRINT
1370 NEXT XX
1380 RETURN
1390 PRINT
1400 INPUT "Press >ENTER< or X to exit  ":Y$
1410 IF Y$="X" THEN 1480
1420 RETURN
1430 PRINT "-----"
1440 RETURN
1450 RANDOMIZE
1460 N=INT(X*RND)+1
1470 RETURN
1480 END
```

When the program is executed, it first displays:

I am TI99/4A. Tell me your
name, please.

After the name has been typed in it continues with:

This program is designed
to practice arithmetic.
You have a choice of using:

- 1 Single-digit figures
2 Double-digit figures

Which?

After either 1 or 2 has been typed, it presents the next choice:

You can practice:

- 1 Addition
2 Subtraction
3 Multiplication
4 Division

Which would you like?

When that choice has been made, it presents two figures:

$$5 + 7 =$$

The sum is?

Depending on what has been typed in in reply, it then displays:

Good, (name), that's right.

or:

Sorry, (name), that's wrong.
Try again.

In the first instance the computer presents the next problem, in the second it repeats the previous problem. The results, in the case of division, have been rounded off to one decimal, and the student, in answering those questions, should do the same. Thus, $6 / 7$ produces a correct reply if the answer is .9, although the actual result is .86.

Line by line, here is what happens:

Line 100 represents the program title and is ignored by the computer.

Line 110 clears the screen.

Lines 120-150 perform the introduction and ask the student to type in his or her name. Line 140 sends the computer to a subroutine (lines 1350-1380) that is used over and over in order to cause the copy to be displayed in the center of the screen instead of across the bottom, which I find annoying.

Line 160 clears the screen again.

Lines 170-260 place a description and the first group of choices into display. Lines 200 and 230 call up a subroutine that places a dashed line into the display. Line 240 again uses the other subroutine to move the copy to the center of the screen. Line 250 assigns the typed-in choice to the numeric variable WHICH, and line 260 tells the computer where to go, depending on the value of WHICH.

Lines 270 and 280 assign a value to the numeric variable X and then send the computer to line 300.

Line 290 assigns another value to that numeric variable.

Lines 300-400 place the second group of choices into display, with line 400 telling the computer where to go, based on the typed-in choice.

Lines 420 and 440 send the computer to a subroutine consisting of lines 1450-1470, where the combination of RANDOMIZE and $\text{INT}(X * \text{RND}) + 1$ produce a random number that is assigned to the numeric variable N and that, in lines 430 and 450, is assigned to the numeric variables NN and NNN, which are then used to represent the values to be manipulated.

Line 460 places the addition equation into display, and line 490 asks that the result be typed in, assigning it to the numeric variable S.

Line 500 causes the computer to perform the calculation, assigning the result to the numeric variable SS.

Line 510 compares the two values and tells the computer to go to line 590 if the typed-in answer is correct.

Lines 530 and 540 are used if the answer is wrong, and line 580 tells the computer to go back to line 460 to display the equation once more.

Line 600 tells the student that the answer is correct, and line 630 sends the computer back to line 410 to produce another equation.

Lines 640-860 are identical to the above, representing the subtraction section.

Lines 870-1090 contain the multiplication section, and lines 1100-1330 are used for division. This last section contains line 1210, where the result is rounded off to one decimal place.

Lines 1340-1470 represent the various subroutines. In line 1400 you're asked to either press >ENTER< to go on or X to quit. If X is pressed, the computer, in line 1410, is told to go to line 1480, which tells the computer that the end of the program has been reached.

The addition and multiplication sections of this program are extremely simple, especially when used with single-digit figures. The division portion is the one that is complicated, because in most instances the result will include fractions.

WORDS THAT MAKE OTHER WORDS

Word Game Program

Now let's look at a program that can prove to be a valuable vocabulary builder. The Word Game Program displays a total of 26 key words, one after the other, asking that the student type in as many words as he or she can think of that can be made up of any number of the letters that constitute the key word.

WORD GAME PROGRAM

A program that can prove helpful as a vocabulary builder.

```
100 REM A WORD GAME/TI99/4A
110 REM WORDS WITHIN WORDS
120 GOSUB 1200
130 PRINT "The letters that make up"
140 PRINT "words can often be used"
150 PRINT "to make up many other words."
160 PRINT
170 PRINT "For instance, the word"
180 PRINT
190 PRINT TAB(12);"MOTHER"
200 PRINT
210 PRINT "contains the following"
220 PRINT "words:"
230 PRINT
240 PRINT "moth, other, her, the, hot,"
250 PRINT "term, home, he, hem, them,"
260 PRINT "and others."
270 PRINT
280 PRINT
290 GOSUB 1220
300 GOSUB 1200
310 PRINT "My name is computer."
320 PRINT "Tell me yours, please."
330 GOSUB 1300
340 INPUT NAME$
350 GOSUB 1200
360 PRINT "Let's see if you, ";NAME$;","
370 PRINT "can come up with more words"
380 PRINT "than I, shall we?"
390 GOSUB 1300
400 INPUT "Are you ready? (Y/N) ":Y$
410 IF Y$<>"Y" THEN 1260 ELSE 420
420 W$="The word is: "
430 T$="Type all the words you can "
440 C$="Compare the words. "
450 F$="Type the number of words you "
460 S$="The score is"
470 GOSUB 1200
480 READ WORD$
490 PRINT W$;">";WORD$;"<"
500 GOSUB 1300
510 PRINT T$
520 INPUT "think of. ":NW$
530 GOSUB 1200
540 PRINT NW$
550 PRINT "-----"
560 READ FF$,Z,EE$
570 PRINT FF$,Z;" ";EE$
580 PRINT "-----"
```

(continued)

```
590 PRINT C$
600 PRINT
610 PRINT F$
620 INPUT "found. ":W
630 A=A+Z
640 B=B+W
650 GOSUB 1200
660 PRINT S$;" ":NAME$;" ";B;" Computer ";A
670 PRINT "-----"
680 GOSUB 1300
690 GOSUB 1220
700 IF Z=216 THEN 720
710 GOTO 470
720 GOSUB 1200
730 INPUT "Another play? (Y/N) ":N$
740 IF N$<>"Y" THEN 1260
750 RESTORE
760 GOTO 470
770 DATA ALWAYS
780 DATA WAY YAW SAY LAY SLAY,5,WORDS
790 DATA BEFORE
800 DATA BE FOR FORE BORE ROBE ROB FOB BEER,8,WORDS
810 DATA CHRISTMAS
820 DATA CHRIST MAST CHASM AT MAT MATCH A SMASH IT IS SIT
    SAT MIST,13,WORDS
830 DATA DEMOCRACY
840 DATA DOME MAY DECAY YORE CORE CARE MARE RACY RACE CRY
    CORD CAY A READ,15,WORDS
850 DATA EXISTENCE
860 DATA SIT IS EXIT EXIST SENT TENSE SIX SEX CENT SCENT
    TIME,11,WORDS
870 DATA FOREST
880 DATA FOR FORE SORE SOFT FROST REST ROT TORE OR OF TO
    SO,12,WORDS
890 DATA GHOST
900 DATA GO HOT SOT HOG TOG TO SO HO SHOT,10,WORDS
910 DATA HEAVEN
920 DATA EVEN HEAVE A VAN HEN EVE HE,7,WORDS
930 DATA INTEREST
940 DATA IN REST RENT SENT INSERT SITTER INTER SEER TIER
    SIT IT TINT,12,WORDS
950 DATA JESTER
960 DATA JET REST JEST JEER RESET SET,6,WORDS
970 DATA KIDNEY
980 DATA KID DIN DEN YEN DIKE KIN KEY IN KIND,9,WORDS
990 DATA LIVELY
1000 DATA LIVE YELL LEI VIE VEIL IVY ILL,7,WORDS
1010 DATA MEDIOCRE
1020 DATA RED CORE MORE MIRE DIRE DOME DIME MODE ME CRIME,
    10,WORDS
1030 DATA NATION
1040 DATA AT ON IN ION TIN TAN NO A AN,11,WORDS
1050 DATA OFFSPRING
```

(continued)


```
1060 DATA OFF SPRING OF SO RING SING FIG PRIG PING FOP FOG
      FIG RIG GO SIN,15,WORDS
1070 DATA PASTURE
1080 DATA UNDER WORLD OR DO RUN DOER LORD WORD DUD LOW
      LOWER,11,WORDS
1090 DATA VESTMENT
1100 DATA VEST MET TENT ME SENT EVE MEET SEE VENT EVENT
      TEST,11,WORDS
1110 DATA WEATHER
1120 DATA ETHER WATER WHEAT REATH HEAT WHAT RAT THAW AT WE
      HE A HERE HER TAR TARE HEAR WEE,18,WORDS
1130 DATA XYLOPHONE
1140 DATA PHONE HONE LONE ONE ON HEY LOX POX HEX LOP HEN
      PEN,12,WORDS
1150 DATA YEAR
1160 DATA EAR ARE AY RAY A,5,WORDS
1170 DATA ZEBRA
1180 DATA ARE BEAR RAZE BRAZE BARE BAR ERA A,8,WORDS
1190 GOTO 470
1200 CALL CLEAR
1210 RETURN
1220 PRINT
1230 INPUT "Press >ENTER< or X to exit ":GO$
1240 IF GO$="X" THEN 1260
1250 RETURN
1260 GOSUB 1200
1270 PRINT TAB(12);"End."
1280 GOSUB 1300
1290 END
1300 FOR Z=1 TO 10
1310 PRINT
1320 NEXT Z
1330 RETURN
```

When we activate the program it first displays its purpose and an example:

The letters that make up
words can often be used
to make up many other words.
For instance, the word
MOTHER
contains the following
words:
moth, other, her, the, hot
term, home, he, hem, them,
and others.

Then the computer introduces itself and asks the player to type in his or her name. After that it displays:

Let's see if you, (name),
can come up with more words
than I, shall we?
Are you ready? (Y/N)

followed by:

The word is >ALWAYS<
Type all the words you can
think of.

After that it displays the words the student typed in and a word list that is stored in the computer, giving the student an opportunity to

compare the two. Next the student is asked to type in the number of words he or she found, after which the computer responds with:

The score is: (name) xx, computer xx

The word lists that are stored in the program are intentionally incomplete in order to give the student an opportunity to beat the score of the computer.

This program is fun to play for kids and adults alike, and I have often had two or more adults sit in front of the computer, trying to outdo one another (as well as the computer) in coming up with more and more possible words. It's a great way to pass a rainy afternoon.

Line by line:

Lines 100 and 110 identify the program.

Lines 120-260 explain the program and present an example.

Lines 290 and 300 are duplicates that were left in accidentally. One should be deleted.

Lines 310-340 perform the introduction and ask that the name of the student be typed in.

Lines 360-410 encourage the student to start.

Lines 420-460 assign a number of strings to string variables.

Line 480 causes the computer to READ the key word from the DATA block, assigning it to the string variable WORD\$, and line 490 causes it to be printed along with the string assigned to the string variable W\$.

Lines 510 and 520 ask the student to type in as many words as possible. (Do not use commas between the words unless you want to enclose the entire word list in quotation marks, because the input words are considered one string and are assigned to the string variable NW\$).

Line 560 causes the computer to go to the DATA lines and to READ the word list, assigning it to the string variable FF\$, the number of words, assigning it to the numeric variable Z, and the last word (WORDS), assigning it to the string variable EE\$. All these data are then displayed in line 570.

Lines 590-660 ask the student to compare the words and type in the number he or she was able to come up with, after which, in line 660, the program displays the cumulative score up to that point.

Line 700 checks the value of Z; if it is 216, the total number of words stored in the word lists in the computer, tells the computer to go to line 720 to find out if you want to play another round. If the answer is other than Y, the computer is told to go to lines 1260-1290 for the END statement. Otherwise it is sent back to line 470 for a fresh start without going through all of the introductory explanations.

Line 750 uses RESTORE to make sure that the items in the DATA block are READ from the beginning.

Lines 770-1180 are the DATA block, and the rest are the various subroutines used throughout the program.

THE PRESIDENTS OF THE UNITED STATES

Now we'll look at a program that might be thought of as presenting a history lesson. It deals with the 40 Presidents of the United States and actually consists of six subprograms. When it is started it

THE PRESIDENTS OF THE UNITED STATES

A program that includes six subprograms about the presidents of the United States.

```
100 CALL CLEAR
110 REM PRESIDENTS OF THE U.S.
120 REM TI EXTENDED BASIC
130 GOTO 530
140 CALL CLEAR :: PRINT "If you're ready, ";N$
150 PRINT :: INPUT "pick a year.....":YY
160 GOSUB 1160
170 RESTORE
180 MM=0
```

(continued)

```
190 IF YY=1789 THEN MM=1 :: GOTO 290
200 FOR K=1 TO 40 :: READ K$ :: NEXT K
210 READ L :: MM=MM+1
220 IF YY>=L THEN 240
230 GOTO 210
240 READ LL :: MM=MM+1
250 IF YY<=LL THEN 270
260 GOTO 240
270 RESTORE
280 MM=MM-1
290 FOR M=1 TO MM :: READ P$ :: NEXT M
300 PRINT "In ";YY;" the President was" :: PRINT :: PRINT
    P$
310 GOSUB 1150 :: GOSUB 1160
320 RESTORE :: GOSUB 1170 :: GOTO 560
330 RESTORE
340 FOR OO=1 TO 40 :: READ O$ :: NEXT OO
350 XX=0 :: RANDOMIZE :: XX=INT(RND*40)+1
360 IF XX<1 THEN 350
370 IF XX>20 THEN 350
380 EE=0
390 FOR WW=1 TO XX :: READ Y1 :: EE=EE+1 :: READ Y2 ::
    NEXT WW
400 CALL CLEAR :: PRINT "Who was the President" :: PRINT :
    : PRINT "from ";Y1;" to ";Y2;"?" :: GOSUB 1160
410 INPUT "Type the name ";PP$
420 RESTORE
430 EE=EE+(EE-1)
440 IF EE>40 THEN 420
450 FOR DD=1 TO EE :: READ P$ :: NEXT DD
460 IF P$=PP$ THEN 510
470 PRINT :: PRINT :: GOSUB 1150
480 PRINT "Sorry, ";N$;", it was ";P$ :: GOSUB 1150 ::
    GOSUB 1160
490 RESTORE
500 GOSUB 1170 :: GOTO 560
510 PRINT :: PRINT :: GOSUB 1150
520 PRINT "That's correct, ";N$ :: GOSUB 1150 :: GOSUB
    1160 :: GOTO 500
530 PRINT "My name is Computer," :: PRINT :: INPUT "type
    yours, please ";N$
540 CALL CLEAR :: GOSUB 1150
550 PRINT "The Presidents of the U.S." :: GOSUB 1150 ::
    GOSUB 1160 :: GOSUB 1170
560 CALL CLEAR :: PRINT "You have six choices:" :: GOSUB
    1150
570 PRINT 1;" Name them in order (1-40)" :: PRINT
580 PRINT 2;" I'll call for them by No." :: PRINT
590 PRINT 3;" You call for them by No." :: PRINT
600 PRINT 4;" You name them to find No." :: PRINT
610 PRINT 5;" You type in a year" :: PRINT
620 PRINT 6;" I'll display the years" :: GOSUB 1150
```

(continued)

```
630 INPUT "Which choice? ":WHICH
640 ON WHICH GOTO 650,960,1020,1070,140,330
650 Q=0 :: B=0
660 RESTORE
670 B=B+1
680 CALL CLEAR :: PRINT "Name President No. ";B :: GOSUB
    1160 :: INPUT PP$
690 IF Q=5 THEN 1000
700 READ P$ :: IF B=40 THEN 1010
710 IF PP$=P$ THEN 730
720 GOTO 760
730 PRINT :: PRINT :: PRINT "That's right, ";N$;". " ::
    GOSUB 1160 :: GOSUB 1170
740 IF Q=5 THEN 960
750 GOTO 670
760 PRINT :: PRINT :: PRINT "Sorry, ";N$;". the answer is:
    " :: PRINT :: PRINT TAB(5);P$ :: GOSUB 1150 :: GOSUB
    1160 :: GOSUB 1170
770 IF Q=5 THEN 960
780 GOTO 670
790 DATA GEORGE WASHINGTON,JOHN ADAMS,THOMAS JEFFERSON,
    JAMES MADISON
800 DATA JAMES MONROE,JOHN QUINCY ADAMS,ANDREW JACKSON,
    MARTIN VAN BUREN
810 DATA WILLIAM H. HARRISON,JOHN TYLER,JAMES POLK,ZACHARY
    TAYLOR
820 DATA MILLARD FILLMORE,FRANKLIN PIERCE,JAMES BUCHANAN,
    ABRAHAM LINCOLN
830 DATA ANDREW JOHNSON,ULYSSES S. GRANT,RUTHERFORD B.
    HAYES,JAMES GARFIELD
840 DATA CHESTER A. ARTHUR,GROVER CLEVELAND,BENJAMIN
    HARRISON,GROVER CLEVELAND
850 DATA WILLIAM MCKINLEY,THEODORE ROOSEVELT,WILLIAM
    HOWARD TAFT,WOODROW WILSON
860 DATA WARREN HARDING,CALVIN COOLIDGE,HERBERT HOOVER,
    FRANKLIN DELANO ROOSEVELT
870 DATA HARRY S. TRUMAN,DWIGHT D. EISENHOWER,JOHN F.
    KENNEDY,LYNDON B. JOHNSON
880 DATA RICHARD M. NIXON,GERALD FORD,JIMMY CARTER,RONALD
    REAGAN
890 DATA 1789,1797,1801,1809,1817,1825
900 DATA 1829,1837,1841,1841,1845,1849
910 DATA 1850,1853,1857,1861,1865,1869
920 DATA 1877,1881,1881,1885,1889,1893
930 DATA 1897,1901,1909,1913,1921,1923
940 DATA 1929,1933,1945,1953,1961,1963
950 DATA 1969,1974,1977,1981,1989
960 X=0 :: RANDOMIZE :: X=INT(RND*40)+1
970 IF X<1 THEN 960 :: IF X>40 THEN 960
980 RESTORE :: FOR W=1 TO X :: READ P$ :: NEXT W
990 B=X :: Q=5 :: GOTO 680
1000 RESTORE :: GOTO 710
```

(continued)

```
1010 CALL CLEAR :: GOSUB 1150 :: PRINT TAB(12);"End." ::  
      GOSUB 1150 :: GOSUB 1160 :: END  
1020 RESTORE  
1030 CALL CLEAR :: PRINT "Which No. President are you" ::  
      PRINT :: INPUT "interested in? ":N  
1040 FOR Z=1 TO N :: READ W$ :: NEXT Z  
1050 GOSUB 1150 :: PRINT TAB(5);W$ :: GOSUB 1150 :: GOSUB  
      1160 :: GOSUB 1170  
1060 GOTO 560  
1070 RESTORE  
1080 CALL CLEAR :: PRINT "Type the president" :: PRINT ::  
      PRINT "whose No. you want" :: GOSUB 1150 :: GOSUB  
      1160 :: INPUT VV$  
1090 V=0  
1100 V=V+1  
1110 READ W$  
1120 IF W$=VV$ THEN 1130 ELSE 1100  
1130 PRINT :: GOSUB 1150 :: PRINT VV$;" is No.";V :: GOSUB  
      1150 :: GOSUB 1160 :: GOSUB 1170  
1140 GOTO 560  
1150 PRINT "-----" :: RETURN  
1160 FOR Z=1 TO 10 :: PRINT :: NEXT Z :: RETURN  
1170 PRINT :: INPUT "Press >ENTER< or X to exit ":Y$  
1180 IF Y$="X" THEN 1200  
1190 RETURN  
1200 CALL CLEAR :: PRINT TAB(12);"End." :: GOSUB 1150 ::  
      GOSUB 1160 :: END
```

first asks that you type in your name, which is then used repeatedly throughout each of the subprograms. After that it displays the title of the program and follows that with:

You have six choices:

- ```

1 Name them in order (1-40)
2 I'll call for them by No.
3 You call for them by No.
4 You name them to find No.
5 You type in a year
6 I'll display the years

```

Which choice?

---

The first choice asks you to type in the names of the Presidents in consecutive order, starting with George Washington and ending with Ronald Reagan. If a wrong name is typed in, the computer tells you so, and then it displays the correct name before asking for the next name.

The second choice results in the computer displaying a number from 1 to 40 at random, asking you to type in the name that is associated with that number. Again, if the answer is wrong, the computer displays the right name before displaying the next number.

The third choice asks you to type in a number and the computer then responds by displaying the name that is associated with that number.

The fourth choice is the reverse. You type in a name and the computer responds by displaying the number associated with that name.

The fifth choice asks you to type in any year between 1789, the year in which George Washington was inaugurated, and today. The computer then responds by displaying the name of the President who was in office that year.

The sixth choice causes the computer to display two years, which represent the term in office of a given President. You are then asked to type in the name of the President who was in office during those years.

Looking at the program line by line can be a bit confusing, primarily because several of the subprograms were added after the original program had already been written, and then the RESe-quence command was used to rearrange the line numbers. In addition, the program was written in TI EXTENDED BASIC in order to keep the number of lines to an acceptable limit.

**Line 100** clears the screen.

**Lines 110 and 120** are REMark lines.

**Line 130** sends the computer to line 530, which is the actual start of the program.

From here on, let's go through the program in the order in which the lines are used, rather than by line number. I believe this will be less confusing.

---



**Line 530** performs the introduction and asks you to type in your name.

**Line 540** clears the screen and places a dashed line into display, using a subroutine (line 1150).

**Line 550** displays the title of the program between two dashed lines, another subroutine (line 1160) moves it up from the bottom of the screen, and a third subroutine (line 1170) asks you to type >ENTER< to continue or X to exit the program.

**Lines 560-630** display the six choices, asking you to type in any number from 1 to 6.

**Line 640** sends the computer to a line number, depending on the typed-in choice.

We'll now go to those line numbers, starting with choice number 1.

**Line 650** assigns the value of zero to two numeric variables.

**Line 660** uses the RESTORE statement to make sure that DATA are READ from the beginning.

**Line 670** increases the value of B by one each time that line is encountered.

**Line 680** asks you to name the President associated with the displayed number (B), which starts with 1 and goes from there to 40.

**Line 690** checks on the variable of Q, the reason for which will become apparent later.

**Line 700** causes the computer to READ one DATA item and checks the value assigned to the numeric variable B, being sent to line 1010 if B equals 40, meaning that all the Presidents have been used up.

**Line 710** compares your answer, which has been assigned to the string variable PP\$, with the name that was READ and assigned to the string variable P\$, telling the computer to go to line 730 if the two strings match.

**Line 720** sends the computer to line 760 if they don't match.

**Line 730** tells you that your answer is correct.

**Line 740** once more checks the value assigned to Q.

**Line 750** sends the computer back to line 670 in order to repeat the previous routine.

**Line 760** is used if your answer was wrong.

**Lines 770 and 780** are duplicates of lines 740 and 750.

---

We now come to the second choice, which starts with line 960.

**Line 960** assigns zero to the numeric variable X and then produces a random number that is assigned to that numeric variable.

**Line 970** makes sure that the random number is between 1 and 40 inclusive.

**Line 980** causes the computer to READ up to 40 DATA items, starting at the beginning, assigning the name to the string variable W\$.

**Line 990** causes the value of X to be assigned to B. It then assigns the value of 5 to the numeric variable Q, which is used in lines 690, 740, and 770 to tell the computer where to go next. It then tells the computer to go to line 680 to repeat the second of the six routines.

We now come to the third choice, starting with line 1020.

**Line 1020** makes sure that DATA are READ from the beginning.

**Line 1030** asks you to key in the number of a given President, assigning it to the numeric variable N.

**Line 1040** represents a loop that causes the computer to READ DATA items up to the number represented by N, assigning the last READ name to the string variable W\$.

**Line 1050** prints that name.

**Line 1060** sends the computer to line 560 to display the six choices once more.

We now come to the fourth choice, starting with line 1070.

**Line 1070** makes sure that DATA items are READ from the beginning.

**Line 1080** asks you to type in the name of the President for whom you want to find the consecutive number. There is one hitch here. Grover Cleveland served two nonconsecutive terms, but the computer will display only the first term.

**Line 1090** assigns zero to V.

**Line 1100** increases the value of V by 1 each time the computer encounters that line.

**Line 1110** causes the computer to READ one DATA item, assigning it to the string variable W\$.

---

**Line 1120** compares the READ name with the one you typed in and, depending on whether or not they match, sends the computer to one of two line numbers.

**Line 1130** is used if the two *do* match. It displays the name and the associated number.

**Line 1140** sends the computer back to line 560 to display the six choices once more.

Next we come to choice number five, starting with line 140.

**Lines 140 and 150** ask you to type in a year, assigning it to the numeric variable YY.

**Lines 160, 170, and 180** are self-explanatory.

**Line 190** checks whether the typed-in year is 1789, in which case the value of 1 is assigned to the numeric variable MM and the computer is told to go to line 290.

**Line 200** represents a loop that causes the computer to perform the READ command 40 times in order to get it past the 40 DATA items that contain the 40 names.

**Line 210** executes the READ command once more, this time assigning the result to the numeric variable L while, at the same time, increasing the value of MM by 1.

**Line 220** compares the year you have typed in with the one that was READ. If your year is greater than or equal to the other, the computer is told to go to line 240.

**Line 230** tells the computer to go back to line 210.

**Line 240** causes the next year to be READ from the DATA lines.

**Line 250** checks whether your year is smaller than or equal to the other, in which case the computer is told to go to line 270.

**Line 260** tells the computer to go back to line 240.

**Lines 270 and 280** are self-explanatory.

**Line 290** sets up a loop that causes the names to be READ from the beginning up to the number represented by the value of MM.

**Line 300** displays the name of the President who was in office during the year that you selected.

**Lines 310 and 320** perform some housekeeping chores and then send the computer to line 560 to display the six choices once more.

---

The sixth choice starts at line 330.

**Line 330** once more RESTOREs the DATA to the beginning.

**Line 340** sets up a loop that READs the first 40 DATA items to get past the 40 names.

**Line 350** picks a random number and assigns it to the numeric variable XX.

**Lines 360 and 370** return the computer to line 350 if the value of XX is smaller than 1 or greater than 20.

**Line 380** assigns zero to EE.

**Line 390** causes the computer to READ two successive years and assigns them to the numeric variables Y1 and Y2.

**Lines 400 and 410** ask you to type in the name of the President who was in office during the time span represented by the two years.

**Lines 420-470** make various comparisons, READ the name of the associated President, compare it to the one you typed in, and tell the computer where to go, depending on your answer.

**Line 480** is used if your answer was wrong.

**Line 520** is used if your answer was right. In each case the computer is then returned to line 560 in order to redisplay the six choices.

If you would like to use this program without TI EXTENDED BASIC, the only changes that need to be made concern the fact that in TI BASIC you cannot place more than one statement on a single line. Therefore, wherever there is a double colon (::) on a line, the statement to the right of the double colon has to be moved to the next line, and the double colon should be deleted. This will make the program physically rather long, but it will perform in a manner identical to the one described here, though it may be a trifle slower. In doing this, be sure to retain the line numbers used in the program listing that is reproduced here, using in-between numbers (101, 102, etc.) for the additional lines in order to make sure that the various GOTO, GOSUB and IF...THEN statements send the computer to the right line numbers. Once the program has been rewritten and test run to make sure that no typographical errors have crept in, you can use the RESequence command to rearrange the line numbers.

---

## SPEED PROGRAM FOR MATHEMATICS OR GRAMMAR

This next program tests your ability to identify and solve a problem quickly. The program offers a choice of two types of problems, mathematical equations or grammar problems. You can also select the length of time that each problem will be displayed on the screen, anywhere from a fraction of a second to several seconds.

### SPEED PROGRAM FOR MATHEMATICS OR GRAMMAR

This program tests your ability to identify and solve problems quickly.

```

100 REM SPEED TEST/TI99/4A
110 GOSUB 10000
120 PRINT "This program is designed": :
130 PRINT "to test your ability": :
140 PRINT "to solve problems fast.": :
150 GOSUB 11000
160 GOSUB 10000
170 PRINT "Your choice:"
180 PRINT "-----"
190 PRINT 1;" Mathematics": :
200 PRINT 2;" Grammar"
210 PRINT "-----"
220 INPUT "Pick one ":P
225 RESTORE
230 IF P=1 THEN 1000 ELSE 2000
300 DATA 12/2.5=?,8.4 4.8 4.6,4.8
310 DATA 3*(15-4)=?,45 30 33,33
320 DATA 14-(7*2)+1=?,28 0 1,1
330 DATA 2^2=?,6 4 12,4
340 DATA 10^3=?,1000 10000 100,1000
350 DATA 5*3^2=?,30 90 45,45
360 DATA -15+(10-15)=?,20 -20 40,-20
370 DATA -25*-2=?,-50 12.5 50,50
380 DATA 99.9+(10/10)=?,100.9 89.9 101.9,100.9
390 DATA 20*-5+2=?,102 -102 -98,-98
400 DATA 20*(-5+2)=?,-98 102 -60,-60
410 DATA 10^-2=?,100 0.01 1,0.01
420 DATA 100.75 - 75.1=?,25.74 25.65 99.75,25.65
430 DATA ((2*2)*2)*2=? ,16 64 32,32
440 DATA 4^2^2=?,256 16 32,256
450 DATA 50/.5=?,10 -100 100,100
460 DATA (18-3)*3=?,45 18 21,45
470 DATA 18-(3*3)=?,45 12 9,9
480 DATA 77.11+12.99=?,100 90.1 90,90.1

```

(continued)

490 DATA 45\*.3=? ,13.5 135 15,13.5  
500 DATA 1) HARRY WANTS TO TAKE JILL AND I TO THE MOVIES.  
510 DATA 2) HARRY WANTS TO TAKE JILL AND ME TO THE MOVIES.  
520 DATA 3) JILL AND ME ARE GOING TO THE MOVIES WITH  
HARRY.  
530 DATA 2  
540 DATA 1) I'M ESPECTING RAIN.  
550 DATA 2) I ESPECT RAIN.  
560 DATA 3) I'M EXPECTING RAIN.  
570 DATA 3  
580 DATA 1) THERE'S 6 APPLES TO EAT.  
590 DATA 2) THERE ARE 6 APPLES TO EAT.  
600 DATA 3) THERE'S 6 APPLES TO BE EATEN.  
610 DATA 2  
620 DATA 1) I'M LYING IN BED.  
630 DATA 2) I'M LAYING IN BED.  
640 DATA 3) I'M LAYING IN THE BED.  
650 DATA 1  
660 DATA 1) I HOPE THAT WE'LL BE GOING ON A TRIP.  
670 DATA 2) HOPEFULLY WE'LL GO ON A TRIP.  
680 DATA 3) HOPEFULLY WE'LL BE GOING ON A TRIP.  
690 DATA 1  
700 DATA 1) JOE OWES HE AND I \$10.  
705 DATA 2) JOE OWES HIM AND I \$10.  
710 DATA 3) JOE OWES HIM AND ME \$10.  
715 DATA 3  
720 DATA 1) HE EXCAPED BEING INJURED.  
725 DATA 2) HE ESCAPED INJURY.  
730 DATA 3) HE EXCAPED INJURY.  
735 DATA 2  
740 DATA 1) FIRST OFF WE'LL GO OUT.  
745 DATA 2) FIRSTLY WE'LL GO OUT.  
750 DATA 3) FIRST WE'LL GO OUT.  
755 DATA 3  
760 DATA 1) THE MEDIA ARE BEING UNFAIR TO THE  
PRESIDENT.  
765 DATA 2) THE MEDIA IS BEING UNFAIR TO THE  
PRESIDENT.  
770 DATA 3) THE MEDIA IS TREATING THE PRESIDENT  
UNFAIRLY.  
775 DATA 1  
780 DATA 1) THE RESTAURANTEUR SERVED US PERSONALLY.  
785 DATA 2) THE RESTAURATEUR SERVED US IN PERSON.  
790 DATA 3) THE RESTAURANTEUR SERVED US IN PERSON.  
795 DATA 2  
800 DATA 1) IT'S BETWEEN BETTY AND JOE AND ME.  
805 DATA 2) IT'S BETWEEN BETTY AND JOE AND I  
810 DATA 3) IT'S AMONG BETTY AND JOE AND ME.  
815 DATA 3  
820 DATA 1) ROSES ARE DIFFERENT FROM DAISIES.  
825 DATA 2) ROSES ARE DIFFERENT THAN DAISIES.  
830 DATA 3) ROSES ARE DIFFERENT THEN DAISIES.  
835 DATA 1

(continued)

```
840 DATA 1) I CAN RUN FASTER THAN HE.
845 DATA 2) I CAN RUN FASTER THEN HE.
850 DATA 3) I CAN RUN FASTER THAN HIM.
855 DATA 1
860 DATA 1) HE'S PRECEEDING AS PER SCHEDULE.
865 DATA 2) HE'S PROCEEDING AS PER SCHEDULE.
870 DATA 3) HE'S PROCEEDING AS PER SCHEDULE.
875 DATA 2
880 DATA 1) 10 MILES IS FURTHER THAN 1 MILE.
885 DATA 2) 10 MILES IS FARTHER THEN 1 MILE.
890 DATA 3) 10 MILES ARE FARTHER THAN 1 MILE.
895 DATA 3
900 DATA 1) THE CAR WON'T START WHEN ITS COLD OUT.
905 DATA 2) THE CAR WILL NOT START WHEN ITS COLD OUT.
910 DATA 3) THE CAR WON'T START WHEN IT'S COLD OUT.
915 DATA 3
920 DATA 1) I WOULD FEEL BETTER IF I WERE WARMER.
925 DATA 2) I WOULD FEEL BETTER IF I WAS WARMER.
930 DATA 3) I WOULD FEEL BETTER IF I WHERE WARMER.
935 DATA 1
940 DATA 1) HIM AND ME IS FRIENDS.
945 DATA 2) HE AND ME ARE FRIENDS.
950 DATA 3) HE AND I ARE FRIENDS.
955 DATA 3
960 DATA 1) THE FOOD TASTES BAD.
965 DATA 2) THE FOOD TASTES BADLY.
970 DATA 3) THE FOOD DOESN'T TASTE WELL.
975 DATA 1
980 DATA 1) THE TRUCK DRIVES TOO SLOW.
985 DATA 2) THE TRUCK IS DRIVING TOO SLOW.
990 DATA 3) THE TRUCK IS DRIVING TOO SLOWLY.
995 DATA 3
1000 GOSUB 10000
1020 GOSUB 4990
1030 GOSUB 10000
1050 R=R+1
1060 IF R>20 THEN 13100
1110 READ M$
1130 PRINT M$
1140 FOR X=1 TO PAUSE
1150 NEXT X
1160 GOSUB 10000
1170 GOTO 12000
1180 FOR XXX=1 TO PAUSE
1190 NEXT XXX
1200 GOSUB 10000
2000 GOSUB 10000
2010 GOSUB 4990
2020 GOSUB 10000
2030 FOR V=1 TO 60
2040 READ Z$
2050 NEXT V
2055 R=0
```

---

(continued)

```
2060 R=R+1
2070 GOSUB 10000
2120 READ G$
2121 READ GG$
2122 READ GGG$
2140 PRINT G$: :
2141 PRINT GG$: :
2142 PRINT GGG$
2150 FOR X=1 TO PAUSE
2160 NEXT X
2170 GOSUB 10000
2180 GOTO 6000
4990 PRINT
5000 PRINT "You can control the time": :
5010 PRINT "you feel you need in order": :
5020 PRINT "to come up with an answer."
5030 PRINT "-----"
5040 PRINT "Type a number from 1 to 10": :
5050 PRINT "1 is short, 10 is long."
5060 PRINT "-----"
5070 INPUT "Number? ":N
5080 PAUSE=N*250
5090 RETURN
6000 INPUT "Answer? ":AA$
6010 READ GGGG$
6020 PRINT
6030 IF AA$=GGGG$ THEN 12500
6040 PRINT "Sorry, that is wrong.": :
6050 PRINT "The correct answer is ": :
6060 PRINT TAB(8);"No.";GGGG$;" of": :
6070 PRINT G$: :
6071 PRINT GG$: :
6072 PRINT GGG$
6080 PRINT
6090 GOTO 12520
10000 CALL CLEAR
10010 RETURN
11000 PRINT
11010 INPUT "Press >ENTER< ":Y$
11020 RETURN
12000 PRINT "Do you want me to give": :
12010 PRINT "you a multiple choice?": :
12030 INPUT "Y/N ":N$
12040 IF N$="N" THEN 12100
12050 READ MM$
12060 PRINT
12070 PRINT MM$
12100 PRINT
12110 INPUT "Answer? ":A$
12120 IF N$<>"N" THEN 12140
12130 READ MM$
12140 READ MMM$
12150 IF A$=MMM$ THEN 12500 ELSE 12600
```

(continued)



```
12500 PRINT
12510 PRINT "That is correct": :
12520 PRINT "Your choice:": :
12530 PRINT "-----"
12540 PRINT 1;" Go on?": :
12550 PRINT 2;" Quit?"
12560 PRINT "-----"
12570 INPUT "Which? ":W
12580 IF P=1 THEN 12900
12590 IF W=1 THEN 2070 ELSE 13000
12600 PRINT
12610 PRINT "Sorry, that is wrong.": :
12620 PRINT "The correct anser is ": :
12630 PRINT TAB(12);MMM$
12640 PRINT : :
12650 GOTO 12520
12900 IF W=1 THEN 1030 ELSE 13000
13000 GOSUB 10000
13010 PRINT TAB(12);"End."
13020 END
13100 GOSUB 10000
13110 PRINT "Those are all the questions": :
13120 PRINT 1;") Select math"
13130 PRINT 2;") Select grammar"
13140 PRINT 3;") Quit": :
13150 INPUT "Pick one ":PP
13160 IF PP<>1 THEN 13180
13170 R=0
13180 IF PP=3 THEN 13200
13190 RESTORE
13200 ON PP GOTO 1000,2000,13000
```

---

When the program is first activated, it displays its purpose and then offers you a choice of mathematics or grammar problems. Next it asks you to decide on the time allowance for each problem by typing a number from 1 to 10 (1 is short, 10 is long). Depending on your choice of subject matter, the program then displays either three short sentences, only one of which is grammatically correct, or mathematical equations, some of which are quite simple while others are more difficult. With the math problems, as soon as the equation has disappeared from the screen the program asks:

---

Do you want me to give  
you a multiple choice?  
Y/N

If you type Y, the program displays three possible answers, only one of which is correct. The program includes a total of 20 problems in each category.

Line by line:

**Lines 100-220** identify the program and offer the choice of mathematics or grammar.

**Line 225** makes sure the items contained in the DATA block are READ from the beginning.

**Lines 230-490** contain the mathematical equations, the three multiple-choice answers, and the correct answer.

**Lines 500-995** contain the grammar problems and the numbers that represent the correct choices.

**Lines 1020 and 2020** send the computer to the subroutine (lines 4990-5090) that permits you to specify the length of time for which the problem will remain in display. In line 5080 the number you typed in is multiplied by 250 and then assigned to the numeric variable PAUSE, which controls the time factor.

**Lines 1030-1200** are used in conjunction with the math problems. Line 1050 increments the value of R by 1 during each pass, and line 1060 sends the computer to line 13100 if the value of R is greater than 20, indicating that all 20 problems have been done. Line 1110 READs the problem, and line 1130 causes it to be displayed. Lines

---

1140 and 1150 represent the loop that controls the time element. Line 1170 sends the computer to line 12000, where you're asked if you want multiple-choice answers displayed.

**Lines 12000-12040** display that choice and send the computer to line 12100 if the answer is negative. Otherwise it goes to the next line.

**Lines 12050-12070** READ the multiple choices and cause them to be displayed.

**Lines 12100-12150** ask you to type in your answer. Then line 12120 checks the string assigned to N\$ to determine whether or not to skip the next line. Lines 12130 and 12140 READ the next two DATA items, and line 12150 compares your answer, assigned to the string variable A\$, with the correct one, assigned to the string variable MMM\$, sending the computer to one of two line numbers where the responses to right or wrong answers are located.

**Lines 12500-12590** are used if the answer you gave was correct, giving you a choice of going on or quitting. In lines 12580 and 12590 the values of the numeric variables P and W are checked to determine whether you're solving the math or grammar problems and whether you want to go on or quit, sending the computer to the appropriate line numbers.

**Lines 12600-12650** are used if your answer was wrong, causing the correct answer to be displayed and then sending the computer back to line 12520 to offer you the choice of going on or quitting.

**Lines 2000-2180** are used when grammar has been selected. After sending the computer to the subroutine that determines the time factor, lines 2030-2050 cause the computer to READ through the first 60 DATA items, which represent the math section. Then, in lines 2120-2142, three sentences are READ and displayed on three separate lines. Lines 2150 and 2160 control the time element, and line 2180 sends the computer to line 6000.

**Lines 6000-6090** check if your answer is right; if not, it displays the three sentences again, identifying the one that is correct.

**Lines 10000-11020** contain two frequently used subroutines that clear the screen and ask you to press >ENTER< to continue program execution.

**Lines 13100-13200** are used after all 20 problems in either category have been used, offering you a choice of picking the same or the other category for another run, or quitting the program.

---

You can, of course, change the problems in the DATA blocks to others that may either be easier or more difficult or that you might prefer for one reason or another. If you do, note that it is important that the total number of items be the same and that the same format is used. In the math section that means:

**problem, multiple-choice answers, correct answer**

and in the grammar portion it must look like this:

**1) sentence  
2) sentence  
3) sentence  
number of correct sentence**

because a change in those formats will require a considerable number of changes throughout the program. Also, if you either increase or decrease the total number of problems in one or both

---

sections, then the number associated with R in line 1060 must be changed to represent the new total in order to avoid a DATA ERROR message.

The program, as shown, is written in TI BASIC. It could be shortened considerably if it were written in TI EXTENDED BASIC, where multiple statements can be grouped on individual lines.

## ARITHMETIC WITH VOICE

This next program Problem Solving, is a lot of fun if you have the optional speech synthesizer (and TI EXTENDED BASIC), because the computer presents a series of relatively simple arithmetic problems by voice before displaying them on the screen and, depending on your typed-in reply, the voice tells you whether your answer was right or wrong. There are a few awkward phrases because of the rather limited vocabulary that is available with the synthesizer. For instance, the vocabulary does not include the words "plus" and "minus" in its word list. Still, it works pretty well.

---

### ARITHMETIC WITH VOICE

---

This program uses the speech synthesizer to play arithmetic games.

---

```
100 CALL CLEAR
110 PRINT "LET'S PLAY SOME" :: PRINT
120 PRINT "SIMPLE ARITHMETIC"
130 FOR X=1 TO 10 :: PRINT :: NEXT X
140 REM SPOKEN MATH
150 REM NEEDS SPEECH SYNTHESIZER AND EXTENDED BASIC
160 CALL SAY("HELLO, I AM COMPUTER. TYPE YOUR NAME AND I
 WILL SPELL IT.")
170 INPUT "YOUR NAME? ":N$
180 CALL SAY(N$)
190 FOR PAUSE=1 TO 250 :: NEXT PAUSE
200 CALL CLEAR :: GOSUB 330
210 CALL SAY("5 AND 7 IS WHAT")
220 INPUT "5+7=? ":SUM
230 IF SUM=12 THEN GOSUB 300 ELSE GOSUB 310
240 IF SUM=12 THEN 250 ELSE 210
250 CALL CLEAR :: GOSUB 330
260 CALL SAY("TWELVE LESS 8 IS WHAT")
```

(continued)

---

```
270 INPUT "12-8=? ":SUM
280 IF SUM=4 THEN GOSUB 300 ELSE GOSUB 310
290 IF SUM=4 THEN 320 ELSE 260
300 CALL SAY("THAT IS CORRECT"):: RETURN
310 CALL SAY("THAT IS NOT CORRECT, TRY AGAIN"):: RETURN
320 CALL CLEAR :: GOSUB 330
325 GOTO 400
330 PRINT TAB(10);"LISTEN!"
340 FOR X=1 TO 10 :: PRINT :: NEXT X :: RETURN
400 CALL SAY("3 LESS 7 IS WHAT")
410 INPUT "3-7=? ":SUM
420 IF SUM=-4 THEN GOSUB 300 ELSE GOSUB 310
430 IF SUM=-4 THEN 440 ELSE 320
440 CALL CLEAR :: GOSUB 330
450 CALL SAY("SIXTY AND 3 LESS 9 IS WHAT")
460 INPUT "60+3-9=? ":SUM
470 IF SUM=54 THEN GOSUB 300 ELSE GOSUB 310
480 IF SUM=54 THEN 490 ELSE 440
490 CALL CLEAR :: GOSUB 330
500 CALL SAY("FIFTY LESS SEVENTY IS WHAT")
510 INPUT "50-70=? ":SUM
520 IF SUM=-20 THEN GOSUB 300 ELSE GOSUB 310
530 IF SUM=-20 THEN 540 ELSE 490
540 CALL CLEAR :: GOSUB 330
550 CALL SAY("2 AND 7 LESS 5 IS WHAT")
560 INPUT "2+7-5=? ":SUM
570 IF SUM=4 THEN GOSUB 300 ELSE GOSUB 310
580 IF SUM=4 THEN 590 ELSE 540
590 CALL CLEAR :: GOSUB 330
600 CALL SAY("1 HUNDRED LESS 7 IS WHAT")
610 INPUT "100-7=? ":SUM
620 IF SUM=93 THEN GOSUB 300 ELSE GOSUB 310
630 IF SUM=93 THEN 640 ELSE 590
640 CALL CLEAR :: GOSUB 330
650 CALL SAY("6 AND 6 AND 6 LESS TEN IS WHAT")
660 INPUT "6+6+6-10=? ":SUM
670 IF SUM=8 THEN GOSUB 300 ELSE GOSUB 310
680 IF SUM=8 THEN 690 ELSE 640
690 CALL CLEAR :: GOSUB 330
700 CALL SAY("FORTY 7 LESS THIRTY 6 IS WHAT")
710 INPUT "47-36=? ":SUM
720 IF SUM=11 THEN GOSUB 300 ELSE GOSUB 310
730 IF SUM=11 THEN 740 ELSE 690
740 CALL CLEAR :: GOSUB 330
750 CALL SAY("2 HUNDRED AND SEVENTY 5 LESS NINETY 5 IS
 WHAT")
760 INPUT "275-95=? ":SUM
770 IF SUM=180 THEN GOSUB 300 ELSE GOSUB 310
780 IF SUM=180 THEN 790 ELSE 740
790 CALL CLEAR :: GOSUB 330
800 CALL SAY("ELEVEN AND 7 LESS TWENTY IS WHAT")
810 INPUT "11+7-20=? ":SUM
```

(continued)

```
820 IF SUM=-2 THEN GOSUB 300 ELSE GOSUB 310
830 IF SUM=-2 THEN 840 ELSE 790
840 CALL CLEAR :: GOSUB 330
850 CALL SAY("5 HUNDRED FORTY 3 AND 4 HUNDRED FIFTY 7
 IS WHAT")
860 INPUT "543+457=? ":SUM
870 IF SUM=1000 THEN GOSUB 300 ELSE GOSUB 310
880 IF SUM=1000 THEN 890 ELSE 840
890 CALL CLEAR :: GOSUB 330
900 CALL SAY("3.5 AND 7.8 IS WHAT")
910 INPUT "3.5+7.8=? ":SUM
920 IF SUM=11.3 THEN GOSUB 300 ELSE GOSUB 310
930 IF SUM=11.3 THEN 940 ELSE 890
940 CALL CLEAR :: GOSUB 330
950 CALL SAY("TWENTY 4.8 LESS TWENTY 1.3 IS WHAT")
960 INPUT "24.8-21.3=? ":SUM
970 IF SUM=3.5 THEN GOSUB 300 ELSE GOSUB 310
980 IF SUM=3.5 THEN 990 ELSE 940
990 CALL CLEAR :: GOSUB 330
1000 CALL SAY("7 HUNDRED FORTY 4.68 AND FORTY 6.32 IS
 WHAT")
1010 INPUT "744.68+46.32=? ":SUM
1020 IF SUM=791 THEN GOSUB 300 ELSE GOSUB 310
1030 IF SUM=791 THEN 1040 ELSE 990
1040 CALL CLEAR :: GOSUB 330
1050 CALL SAY("-5 AND -8 IS WHAT")
1060 INPUT "-5+(-8)=? ":SUM
1070 IF SUM=-13 THEN GOSUB 300 ELSE GOSUB 310
1080 IF SUM=-13 THEN 1090 ELSE 1040
1090 CALL CLEAR :: GOSUB 330
1100 CALL SAY("-3 LESS -9 IS WHAT")
1110 INPUT "-3-(-9)=? ":SUM
1120 IF SUM=6 THEN GOSUB 300 ELSE GOSUB 310
1130 IF SUM=6 THEN 1140 ELSE 1090
1140 CALL CLEAR :: GOSUB 330
1150 CALL SAY("0.7 LESS 0.3 IS WHAT")
1160 INPUT "0.7-0.3=? ":SUM
1170 IF SUM=0.4 THEN GOSUB 300 ELSE GOSUB 310
1180 IF SUM=0.4 THEN 1190 ELSE 1140
1190 CALL CLEAR :: GOSUB 330
1200 CALL SAY("0.12 AND TWENTY 1.24 IS WHAT")
1210 INPUT "0.12+21.24=? ":SUM
1220 IF SUM=21.36 THEN GOSUB 300 ELSE GOSUB 310
1230 IF SUM=21.36 THEN 1240 ELSE 1190
1240 CALL CLEAR :: GOSUB 330
1250 CALL SAY("THIRTEEN AND EIGHTY 7 LESS 1 HUNDRED IS
 WHAT")
1260 INPUT "13+87-100=? ":SUM
1270 IF SUM=0 THEN GOSUB 300 ELSE GOSUB 310
1280 IF SUM=0 THEN 1290 ELSE 1240
1290 CALL CLEAR :: GOSUB 330
1300 CALL SAY("THAT IS ALL FOR NOW")
1310 END
```

---

Before activating the program, be sure the sound on the monitor is turned up. First the screen displays:



LET'S PLAY SOME  
SIMPLE ARITHMETIC

and after a brief pause the voice can be heard:

"Hello, I am Computer. Type your name and I will spell it."

After you have typed in your name, the computer spells it (except in the unlikely event that your name is one of the words in its word list). The display then shows the word:



LISTEN!

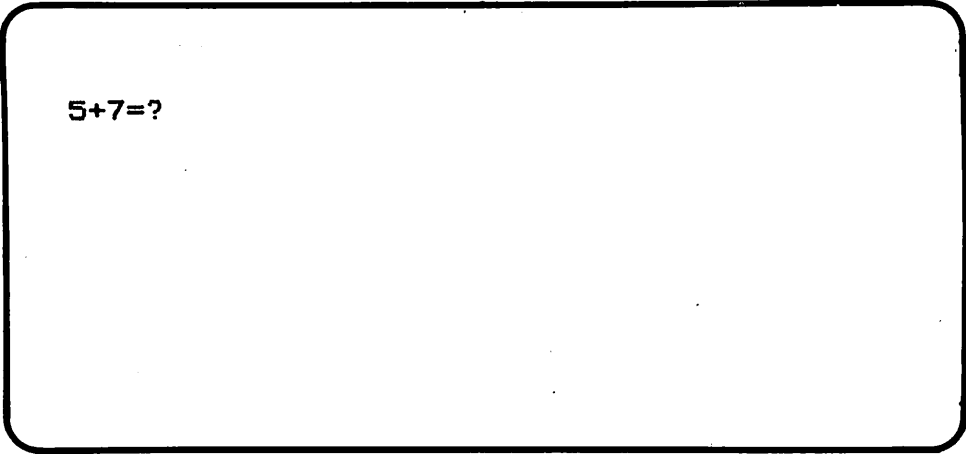
in the center of the screen and after a moment the voice comes on again:

"Five and seven is what?"

---



after which the display changes to:



5+7=?

asking you to type in the result. When you have done so, the voice responds with:

"That is correct."

or

"That is not correct. Try again."

The program then goes on to the next problem or repeats the previous one. The program contains 20 problems in ascending order of difficulty. More can be added without affecting the way it works, as explained at the end of the line-by-line discussion.

Line by line:

**Lines 100-130** place the title of the program into display.

**Lines 140 and 150** are REMarks.

**Line 160** activates the voice synthesizer and tells it what to say, using the CALL SAY statement.

**Line 170** places YOUR NAME? into display, asking you to type it in and assigning it to the string variable N\$.

**Line 180** causes the computer to spell your name.

**Line 190** creates a short pause.

---

**Line 200** clears the screen and uses a subroutine (lines 330 and 340) to place the word LISTEN! into the center of the screen.

**Line 210** tells the computer to speak the first problem.

**Line 220** displays the problem on the screen, asking you to type in the result.

**Line 230** checks whether your answer is right or wrong, sending the computer to one of two subroutines (lines 300 and 310) that cause the voice to tell you whether or not your answer is correct.

**Line 240** once more checks your answer, sending the computer to one of two line numbers to either go to the next problem or to repeat the previous one.

From here on, except for lines 300, 310, 330, and 340, which are subroutines, the program simply repeats the same line sequence for each problem. Notice the way some numbers must be written in the CALL SAY lines in order for the computer to understand what the voice is supposed to say (575 has to be written as 5 HUNDRED SEVENTY 5).

**Line 1300** tells the computer to tell you that you're at the end of the program.

If you want to add more problems, give lines 1300 and 1310 higher line numbers and use the routine represented by the series of five line numbers for each problem (1190-1230 and 1240-1280 are two such sequences) to enter as many additional problems as your heart desires. But be sure to use the words that are listed in the TI EXTENDED BASIC word list.

## SCRAMBLED WORDS

Let's try another program that deals with words. This one displays a bunch of letters that make no sense. It is your job to rearrange these letters so as to make a word. There is a total of 50 words for you to determine. See displays on following pages.

---

It goes like this:

I am your Home Computer.  
Tell me your name, please.

and after you have typed it in:

This program displays  
groups of letters that must  
be rearranged to make words.

and then:

---

The letters are EWRTA

-----  
What is the word?

If the word you now type in is the right one, the computer tells you so and goes on to the next one. If you've typed an incorrect word, it displays the right one before going on.

### SCRAMBLED WORDS PROGRAM

A program that asks you to unscramble some scrambled words.

```
100 CALL CLEAR
110 PRINT "I am your Home Computer." :: PRINT "Tell me your
 name, please"
120 GOSUB 960
130 INPUT N$
140 CALL CLEAR
150 RESTORE
160 REM SCRAMBLED WORDS
170 REM TI EXTENDED BASIC
180 GOSUB 960
190 PRINT "This program displays"
200 PRINT "groups of letters that must"
210 PRINT "be rearranged to make words"
220 GOSUB 960 :: GOSUB 1020
230 L$="The letters are: "
240 W$="The word is "
250 R$="That's right, "
```

(continued)

```

260 F$="No, that's wrong, "
270 GOSUB 970
280 GOSUB 960
290 Q=Q+1
300 IF Q>50 THEN 1040
310 READ A$
320 PRINT L$;A$
330 GOSUB 1020
340 PRINT "What is the word?" :: GOSUB 960 :: INPUT WW$
350 READ AA$
360 IF WW$<>AA$ THEN 410
370 PRINT
380 PRINT R$;N$
390 GOSUB 960
400 GOTO 270
410 PRINT
420 PRINT F$;N$: :
430 PRINT W$;AA$
440 GOSUB 960
450 GOTO 270
460 DATA EWRTA,WATER
470 DATA EBACH,BEACH
480 DATA LEEXI,EXILE
490 DATA RUMSEM,SUMMER
500 DATA RHOSE,HORSE
510 DATA POTUCMRE,COMPUTER
520 DATA HOLOSC,SCHOOL
530 DATA REDOBOM,BEDROOM
540 DATA WLEROF,FLOWER
550 DATA TISHR,SHIRT
560 DATA HENEPOTLE,TELEPHONE
570 DATA DOIRA,RADIO
580 DATA PLAM,LAMP
590 DATA FOCEFE,COFFEE
600 DATA PREPA,PAPER
610 DATA LEAPT,PLATE
620 DATA PELES,SLEEP
630 DATA LIBMAOTUED,AUTOMOBILE
640 DATA CEENF,FENCE
650 DATA AAPRELIN,AIRPLANE
660 DATA THOMER,MOTHER
670 DATA REACHET,TEACHER
680 DATA RUMBEN,NUMBER
690 DATA ICEBYLC,BICYCLE
700 DATA AIRYCIDONT,DICTIONARY
710 DATA MEWDOA,MEADOW
720 DATA ESTOV,STOVE
730 DATA UPIRTCE,PICTURE

```

(continued)

---

```
740 DATA STAAL,ATLAS
750 DATA COALGAT,CATALOG
760 DATA FINKE,KNIFE
770 DATA TREACP,CARPET
780 DATA LICEING,CEILING
790 DATA SEDERST,DESSERT
800 DATA CONEA,OCEAN
810 DATA AACHUETPR,PARACHUTE
820 DATA SEDERT,DESERT
830 DATA BRAYLIR,LIBRARY
840 DATA HOODANBK,HANDBOOK
850 DATA BEALT,TABLE
860 DATA PILCNE,PENCIL
870 DATA UCCSTA,CACTUS
880 DATA KRAMTE,MARKET
890 DATA NOWDIW,WINDOW
900 DATA PEATER,REPEAT
910 DATA RIGEREST,REGISTER
920 DATA APPLENICA,APPLIANCE
930 DATA PRINTERSEE,ENTERPRISE
940 DATA CAYLAFL,FALLACY
950 DATA DOERCORR,RECORDER
960 FOR X=1 TO 10 :: PRINT :: NEXT X :: RETURN
970 GOSUB 1020
980 PRINT "To exit program press X"
990 INPUT "To continue, press >ENTER< ":Y$
1000 IF Y$="X" THEN 1040
1010 CALL CLEAR :: RETURN
1020 PRINT "-----"
1030 RETURN
1040 GOSUB 960
1050 GOSUB 1020
1060 PRINT ;TAB(10);"End."
1070 GOSUB 1020
1080 END
```

---

Let's look at the Word Scrambler Program line by line:

**Line 110** performs the introduction.

**Lines 120** uses a subroutine to place the two lines of copy into the center of the screen.

**Line 130** assigns your name to the string variable N\$.

**Line 150** makes sure that DATA are READ from the top.

**Lines 160 and 170** are REMarks.

---

**Lines 190–210** display the purpose of the program.

**Lines 230–260** assign several strings to string variables.

**Line 290** increases the value of Q each time the line is encountered.

**Line 300** checks the value of Q and, if all 50 words have been used, sends the computer to line 1040.

**Line 310** READs the group of scrambled letters and assigns it to the string variable A\$.

**Line 320** displays the strings represented by two string variables.

**Line 340** asks you to type in the word, assigning it to the string variable WW\$.

**Line 350** READs the correct word as stored in the program, assigning it to the string variable AA\$.

**Line 360** checks if your word matches the correct one, sending the computer to line 410 if your entry was wrong.

**Line 380** is used if your entry was right.

**Line 400** sends the computer back to line 270 to look for the next group of scrambled letters.

**Lines 420 and 430** tell you that your entry was wrong and then display the right word.

**Line 450** is the same as line 400.

**Lines 460–950** represent the DATA block that contains all 50 words and the groups of scrambled letters.

**Lines 960–1080** are the frequently used subroutines and the line that puts the word "End." into display after all words have been used or if you typed X instead of >ENTER< in order to exit the program.

I have tried not to include letter groups that can be used to make more than one word, but it is not impossible that you may find a legitimate word that uses all the letters but is not the one stored in the program. The computer will then insist that you're wrong even though your word may be perfectly good.

## **HISTORY LESSON**

Next we take up a history lesson. The American History Program, works in either of two ways. It may display any one of 25 important dates in American history, asking you to type in the event

---

associated with that date. It then displays the event that is stored in the program, asking you to compare your entry with it before going on to the next date. Alternatively, the program may display an event, asking you to type in the year in which that event took place. It then compares your entry with the date stored in the program to determine whether you were right or wrong. It then continues by displaying another event. The dates and events are picked at random and, because of the peculiarity of random-number generation, there may occasionally be duplications. The dates and events are grouped in the DATA block and can be changed if you like.

### HISTORY LESSON PROGRAM

A computerized history lesson.

```

100 GOTO 140
110 PRINT "-----" :: RETURN
120 FOR X=1 TO 10 :: PRINT :: NEXT X :: RETURN
130 INPUT "Press >ENTER< ":E$:: CALL CLEAR :: RETURN
140 REM AMERICAN HISTORY
150 REM TI EXTENDED BASIC
160 CALL CLEAR
170 GOSUB 110
180 PRINT TAB(5);"American History"
190 GOSUB 110 :: GOSUB 120 :: GOSUB 130
200 PRINT "You have two choices:" :: GOSUB 110
210 PRINT 1;" I display a date" :: PRINT
220 PRINT 2;" I display an event" :: GOSUB 110
230 INPUT "Which? ":WHICH
240 RESTORE :: GOSUB 250 :: GOTO 280
250 RANDOMIZE :: Q=INT(RND*25)+1
260 IF Z=Q THEN 250 ELSE Z=Q
270 RETURN
280 ON WHICH GOTO 370,290
290 QQ=Q/2 :: QQQ=INT(QQ):: QQ=(QQ-QQQ)*100 :: IF QQ=0
 THEN Q=Q-1
300 FOR XX=1 TO Q
310 GOSUB 110
320 READ DATE :: READ EVENT$:: NEXT XX
330 PRINT EVENT$:: GOSUB 110 :: GOSUB 120
340 INPUT "What year? ":YEAR
350 GOSUB 120 :: PRINT TAB(12);DATE :: GOSUB 110
360 GOSUB 130 :: RESTORE :: GOSUB 250 :: GOTO 290
370 QQ=Q/2 :: QQQ=INT(QQ):: QQ=(QQ-QQQ)*100 :: IF QQ=0
 THEN Q=Q-1

```

(continued)



```
380 FOR XXX=1 TO Q
390 GOSUB 110
400 READ DATE :: READ EVENT$:: NEXT XXX
410 PRINT TAB(12);DATE :: GOSUB 110:GOSUB 120
420 INPUT "What took place? ":WHAT$
430 GOSUB 120 :: PRINT EVENT$:: GOSUB 110
440 GOSUB 130 :: RESTORE :: GOSUB 250 :: GOTO 370
450 DATA 1492,CHRISTOPHER COLUMBUS SIGHTS LAND ON OCT.12
460 DATA 1636,HARVARD COLLEGE IS FOUNDED
470 DATA 1683,WILLIAM PENN SIGNS TREATY WITH INDIANS FOR
 PENNSYLVANIA
480 DATA 1752,BENJAMIN FRANKLIN USES KITE TO PROVE
 LIGHTNING IS ELECTRICITY AND INVENTS LIGHTNING ROD
490 DATA 1754,FRENCH AND INDIAN WARS
500 DATA 1776,DECLARATION OF INDEPENDENCE IS SIGNED ON
 JULY 4
510 DATA 1789,INAUGURATION OF GEORGE WASHINGTON AS
 PRESIDENT
520 DATA 1793,ELI WHITNEY INVENTS COTTON GIN
530 DATA 1812,WAR IS DECLARED ON BRITAIN
540 DATA 1848,GOLD WAS DISCOVERED IN CALIFORNIA
550 DATA 1863,LINCOLN ISSUES EMANCIPATION PROCLAMATION
560 DATA 1871,CHICAGO FIRE
570 DATA 1894,THOMAS EDISON FIRST SHOWS MOTION PICTURE
580 DATA 1898,U.S.BATTLESHIP 'MAINE' BLOWN UP AT HAVANA
590 DATA 1906,SAN FRANCISCO EARTHQUAKE
600 DATA 1909,ADMIRAL PEARY REACHES NORTH POLE
610 DATA 1963,JOHN F. KENNEDY ASSASSINATED
620 DATA 1941,JAPAN BOMBS PEARL HARBOR
630 DATA 1918,END OF WORLD WAR I. NOV.11
640 DATA 1929,STOCK MARKET CRASH. OCT.29
650 DATA 1917,U.S. DECLARED WAR ON GERMANY. APR.6
660 DATA 1945,FIRST ATOMIC BOMB EXPLODED IN NEW MEXICO ON
 JULY 16 AND THEN DROPPED ON JAPAN ON AUG.6 AND AUG.9
670 DATA 1950,FIRST U.S. ADVISORS TO SOUTH VIETNAM
680 DATA 1974,RICHARD M. NIXON RESIGNS AS PRESIDENT. AUG.9
690 DATA 1979,THREE MILE ISLAND ACCIDENT AT ATOMIC POWER
 PLANT
700 DATA 1980,MT. ST. HELENS ERUPTS. MAY 18
```

---

Because any given year may be representative of more than one important event, the second choice is probably more useful. As you have probably noticed, each DATA line consists of the year followed by a comma, followed by the event (with no commas). If you want to change these items, be sure to follow the same convention; if you do

---

want to use commas in the description of the event, don't forget that the entire item must be enclosed in quotation marks. Also, if you would like to add more dates and events, you can add as many DATA lines as you like, but the number associated with RND in line 250 must then be changed to reflect the total number of such lines.

Line by line:

**Line 100** is used to send the computer to line 140, skipping over the subroutines, which, in this program, are placed in lines 110-130.

**Lines 140 and 150** are REMarks.

**Lines 170-230** place the title of the program and then the two choices into display, assigning your choice (1 or 2) to the numeric variable WHICH.

**Line 240** first makes sure that the DATA items are READ from the top. It then sends the computer to the subroutine (lines 250-270) where a random number between 1 and 25 is generated in line 250. Line 260 makes sure that the new number is not the same as the one that was generated during the previous pass.

**Line 280** sends the computer to one of two line numbers, depending on your choice.

**Line 290** makes sure that the value that is finally assigned to the numeric variable Q is an odd number, which is important because the computer must subsequently READ the year before the event, and all years are odd-numbered DATA items.

**Lines 300-320** create a loop that causes as many pairs of DATA items to be READ as are represented by the value assigned to Q.

**Line 330** causes the event, assigned to the string variable EVENT\$, to be displayed.

**Line 340** asks you to type in the year that is associated with that event.

**Line 350** causes the year, assigned to the numeric variable DATE, to be displayed, affording you a chance to check whether your entry was right or wrong.

**Line 360** uses RESTORE and then sends the computer to the subroutine that produces a new random number. After that it sends the computer back to line 290 to display the next event.

**Line 370** is identical to line 290.

---

**Lines 380–400** represent another loop that is identical to the first one (lines 300–320).

**Line 410** causes a year, assigned to the numeric variable DATE, to be displayed.

**Line 420** asks you to type in the important event of that year.

**Line 430** prints the event, assigned to the string variable EVENT\$, giving you a chance to compare your entry with the event stored in the program. Note that though your answer may differ, it is not necessarily wrong.

**Line 440** is identical to line 360, except that it eventually returns the computer to line 370 to display the next year.

**Lines 450–700** represent the DATA block, which can be expanded or changed to fit your needs or desires.

## METRIC CONVERSIONS

Nearly everyone in the world today measures things in the metric system. Even though the United States has long held out against it, more and more products appearing in this country are weighed or measured in the metric system. Road signs often show distances in kilometers, gasoline is sometimes sold by the liter, even recipes are beginning to use metric weights and measures. The Metric Conversions Program allows you to convert quantities easily to and from the metric system. When the program is executed it identifies itself and then offers you an extensive choice of conversions.

---

### METRIC CONVERSION PROGRAM

---

This program converts weights and measures to and from the metric system.

---

```
100 REM METRIC CONVERSIONS
110 REM TI99/4A
200 GOSUB 10000
210 PRINT "This program converts": :
220 PRINT "measures and weights": :
230 PRINT "to and from metric.": :
240 GOSUB 11000
```

(continued)

---

```
300 GOSUB 10000
310 PRINT "Your choice:"
320 GOSUB 10500
330 PRINT 1;" Inch/centimeter"
340 PRINT 2;" Foot/meter"
350 PRINT 3;" Mile/kilometer"
360 PRINT 4;" Yard/meter"
370 PRINT 5;" Sq.inch/sq.centimeter"
380 PRINT 6;" Sq.foot/sq.meter"
390 PRINT 7;" Sq.mile/sq.kilometer"
400 PRINT 8;" Sq.yard/sq.meter"
410 PRINT 9;" Cu.inch/cu.centimeter"
420 PRINT 10;" Cu.foot/cu.meter"
430 PRINT 11;" Grain/gram"
440 PRINT 12;" Ounce/gram"
450 PRINT 13;" U.S.pound/kilogram"
460 PRINT 14;" U.S.pound/metric pound"
470 PRINT 15;" Fl.ounce/cu.centimeter"
480 PRINT 16;" Quart/liter"
490 PRINT 17;" U.S.gallon/liter"
500 PRINT 18;" Acre/sq.feet"
510 PRINT 19;" Degrees F./degrees C."
515 PRINT 20;" Exit program"
520 GOSUB 10500
530 INPUT "Which one? ":N
540 ON N GOTO 600,700,800,900,1000,1100,1200,1300,
 1400,1500,1600,1700,1800,1900,2000,2100,2200,
 2300,2400,12000
600 GOSUB 10000
610 PRINT 1;" Inch to centimeter?"
620 PRINT 2;" Centimeter to inch?"
630 GOSUB 10500
640 INPUT "Which? ":W
641 ON W GOTO 645,675
645 GOSUB 10000
646 INPUT "Number of inches? ":I
650 GOSUB 10500
655 C=I*2.54
656 C=INT(C*100+.5)/100
660 PRINT C;" centimeters"
670 GOTO 240
675 GOSUB 10000
680 INPUT "Number of centimeters? ":C
685 GOSUB 10500
690 I=C/2.54
691 I=INT(I*100+.5)/100
692 PRINT I;" inches"
695 GOTO 240
700 GOSUB 10000
705 PRINT 1;" Feet to meters? "
```

(continued)

```

710 PRINT 2;" Meters to feet? "
715 GOSUB 10500
720 INPUT "Which? ":N
725 ON N GOTO 730,760
730 GOSUB 10000
735 INPUT "Number of feet? ":F
740 GOSUB 10500
745 M=F*.3048
750 M=INT(M*100+.5)/100
751 PRINT M;" Meters"
755 GOTO 240
760 GOSUB 10000
765 INPUT "Number of meters? ":M
770 F=M*3.2808
775 F=INT(F*100+.5)/100
777 GOSUB 10500
780 PRINT F;" Feet"
790 GOTO 240
800 GOSUB 10000
805 PRINT 1;" Statute miles to km.?"
806 PRINT 2;" Nautical miles to km.?"
807 PRINT 3;" Km. to statute miles?"
808 PRINT 4;" Km. to nautical miles?"
810 GOSUB 10500
815 INPUT "Which? ":N
816 ON N GOTO 817,830,855,875
817 GOSUB 10000
819 INPUT "Number of statute miles? ":M
820 K=M*1.6089
822 K=INT(K*100+.5)/100
823 GOSUB 10500
825 PRINT K;" Kilometers"
827 GOTO 240
830 GOSUB 10000
832 INPUT "Number of nautical miles? ":M
835 K=M*1.6089/.86839
837 K=INT(K*100+.5)/100
838 GOSUB 10500
840 PRINT K;" Kilometers"
850 GOTO 240
855 GOSUB 10000
857 INPUT "Number of kilometers? ":K
859 M=K*.62137
860 M=INT(M*100+.5)/100
862 GOSUB 10500

```

(continued)

```
865 PRINT M;" Statute miles"
870 GOTO 240
875 GOSUB 10000
877 INPUT "Number of kilometers? ":K
879 M=K*.53956
880 M=INT(M*100+.5)/100
882 GOSUB 10500
885 PRINT M;" Nautical miles"
890 GOTO 240
900 GOSUB 10000
905 PRINT 1;" Yards to meters?"
910 PRINT 2;" Meters to yards?"
915 GOSUB 10500
920 INPUT "Which? ":N
925 ON N GOTO 930,965
930 GOSUB 10000
935 INPUT "Number of yards? ":Y
940 M=Y*.9144
945 M=INT(M*100+.5)/100
946 GOSUB 10500
950 PRINT M;" Meters"
960 GOTO 240
965 GOSUB 10000
970 INPUT "Number of meters? ":M
975 Y=M*1.0936
980 Y=INT(Y*100+.5)/100
982 GOSUB 10500
985 PRINT Y;" Yards"
990 GOTO 240
1000 GOSUB 10000
1005 PRINT 1;" Sq.inches to sq.cm.?"
1010 PRINT 2;" Sq.cm. to sq.inches?"
1015 GOSUB 10500
1020 INPUT "Which? ":N
1022 ON N GOTO 1025,1055
1025 GOSUB 10000
1030 INPUT "Number of sq.inches? ":I
1035 C=I*6.4516
1036 C=INT(C*100+.5)/100
1037 GOSUB 10500
1040 PRINT C;" Sq.centimeters"
1050 GOTO 240
1055 GOSUB 10000
1060 INPUT "Number of sq.centimeters? ":C
1065 I=C*.155
1070 I=INT(I*100+.5)/100
```

(continued)

```
1072 GOSUB 10500
1075 PRINT I;" Sq.inches"
1080 GOTO 240
1100 GOSUB 10000
1105 PRINT 1;" Sq.feet to sq.meters?"
1110 PRINT 2;" Sq.meters to sq.feet?"
1115 GOSUB 10500
1120 INPUT "Which? ":N
1125 ON N GOTO 1130,1160
1130 GOSUB 10000
1135 INPUT "Number of sq.feet? ":F
1140 M=F*.0929
1145 M=INT(M*100+.5)/100
1150 GOSUB 10500
1151 PRINT M;" Sq.meters"
1155 GOTO 240
1160 GOSUB 10000
1165 INPUT "Number of sq.meters? ":M
1170 F=M*10.764
1175 F=INT(F*100+.5)/100
1180 GOSUB 10500
1190 PRINT F;" Sq.feet"
1195 GOTO 240
1200 GOSUB 10000
1205 PRINT 1;" Sq.miles to sq.km.?"
1210 PRINT 2;" Sq.km. to sq.miles?"
1215 GOSUB 10500
1220 INPUT "Which? ":N
1225 ON N GOTO 1230,1265
1230 GOSUB 10000
1235 INPUT "Number of sq.miles? ":M
1240 K=M*2.59
1245 K=INT(K*100+.5)/100
1250 GOSUB 10500
1255 PRINT K;" Sq.kilometers"
1260 GOTO 240
1265 GOSUB 10000
1270 INPUT "Number of sq.kilometers? ":K
1275 M=K*.3861
1280 M=INT(M*100+.5)/100
1285 GOSUB 10500
1290 PRINT M;" Sq.miles"
1295 GOTO 240
1300 GOSUB 10000
1305 PRINT 1;" Sq.yards to sq.meters?"
1310 PRINT 2;" Sq.meters to sq.yards?"
1315 GOSUB 10500
1320 INPUT "Which? ":N
1325 ON N GOTO 1330,1365
1330 GOSUB 10000
1335 INPUT "Number of sq.yards? ":Y
```

(continued)

---

```
1340 M=Y*.83613
1345 M=INT(M*100+.5)/100
1350 GOSUB 10500
1355 PRINT M;" Sq.meters"
1360 GOTO 240
1365 GOSUB 10000
1370 INPUT "Number of sq.meters? ":M
1375 Y=M*1.196
1380 Y=INT(Y*100+.5)/100
1385 GOSUB 10500
1390 PRINT Y;" Sq.yards"
1395 GOTO 240
1400 GOSUB 10000
1405 PRINT 1;" Cu.inches to cu.cm.?"
1410 PRINT 2;" Cu.cm. to cu.inches?"
1415 GOSUB 10500
1420 INPUT "Which? ":N
1425 ON N GOTO 1430,1465
1430 GOSUB 10000
1435 INPUT "Number of cu.inches? ":I
1440 C=I*16.387
1445 C=INT(C*100+.5)/100
1450 GOSUB 10500
1455 PRINT C;" Cu.centimeters"
1460 GOTO 240
1465 GOSUB 10000
1470 INPUT "Number of cu.centimeters? ":C
1475 I=C*.06102
1480 I=INT(I*100+.5)/100
1485 GOSUB 10500
1490 PRINT I;" Cu.inches"
1495 GOTO 240
1500 GOSUB 10000
1505 PRINT 1;" Cu.feet to cu.meters?"
1510 PRINT 2;" Cu.meters to cu.feet?"
1515 GOSUB 10500
1520 INPUT "Which? ":N
1525 ON N GOTO 1530,1565
1530 GOSUB 10000
1535 INPUT "Number of cu.feet? ":F
1540 M=F*.02831
1545 M=INT(M*100+.5)/100
1550 GOSUB 10500
1555 PRINT M;" Cu.meters"
1560 GOTO 240
1565 GOSUB 10000
1570 INPUT "Number of cu.meters? ":M
1575 F=M*35.314
1580 F=INT(F*10+.5)/100
1585 GOSUB 10500
1590 PRINT F;" Cu.feet"
1595 GOTO 240
```

(continued)



```
1600 GOSUB 10000
1605 PRINT 1;" Grains to grams?"
1610 PRINT 2;" Grams to grains?"
1615 GOSUB 10500
1620 INPUT "Which? ":N
1625 ON N GOTO 1630,1665
1630 GOSUB 10000
1635 INPUT "Number of grains? ":G
1640 GG=G/15.432
1645 GG=INT(GG*100+.5)/100
1650 GOSUB 10500
1655 PRINT GG;" Grams"
1660 GOTO 240
1665 GOSUB 10000
1670 INPUT "Number of grams? ":GG
1675 G=GG*15.432
1680 G=INT(G*100+.5)/100
1685 GOSUB 10500
1690 PRINT G;" Grains"
1695 GOTO 240
1700 GOSUB 10000
1705 PRINT 1;" Ounces to grams?"
1710 PRINT 2;" Grams to ounces? "
1715 GOSUB 10500
1720 INPUT "Which? ":N
1725 ON N GOTO 1730,1765
1730 GOSUB 10000
1735 INPUT "Number of ounces? ":O
1740 G=O*28.35
1745 G=INT(G*100+.5)/100
1750 GOSUB 10500
1755 PRINT G;" Grams"
1760 GOTO 240
1765 GOSUB 10000
1770 INPUT "Number of grams? ":G
1775 O=G*.03527
1780 O=INT(O*100+.5)/100
1785 GOSUB 10500
1790 PRINT O;" Ounces"
1795 GOTO 240
1800 GOSUB 10000
1805 PRINT 1;" U.S.pounds to kilograms?"
1810 PRINT 2;" Kilograms to U.S.pounds?"
1815 GOSUB 10500
1820 INPUT "Which? ":N
1825 ON N GOTO 1830,1865
1830 GOSUB 10000
1835 INPUT "Number of U.S.pounds? ":P
1840 K=P*.45359
1845 K=INT(K*100+.5)/100
1850 GOSUB 10500
1855 PRINT K;" Kilograms"
1860 GOTO 240
```

(continued)

---

```
1865 GOSUB 10000
1870 INPUT "Number of kilograms? ":K
1875 P=K*2.2046
1880 P=INT(P*100+.5)/100
1885 GOSUB 10500
1890 PRINT P;" U.S.pounds"
1895 GOTO 240
1900 GOSUB 10000
1905 PRINT 1;" U.S.lbs to metric lbs?"
1910 PRINT 2;" Metric lbs to U.S. lbs?"
1915 GOSUB 10500
1920 INPUT "Which? ":N
1925 ON N GOTO 1930,1965
1930 GOSUB 10000
1935 INPUT "Number of U.S.pounds? ":P
1940 MP=P/1.17137
1945 MP=INT(MP*100+.5)/100
1950 GOSUB 10500
1955 PRINT MP;" Metric pounds"
1960 GOTO 240
1965 GOSUB 10000
1970 INPUT "Number of metric pounds? ":MP
1975 P=MP*1.17137
1980 P=INT(P*100+.5)/100
1985 GOSUB 10500
1990 PRINT P;" U.S.Pounds"
1995 GOTO 240
2000 GOSUB 10000
2005 PRINT 1;" Fluid ounces to cu.cm.?"
2010 PRINT 2;" Cu.cm. to fluid ounces?"
2015 GOSUB 10500
2020 INPUT "Which? ":N
2025 ON N GOTO 2030,2065
2030 GOSUB 10000
2035 INPUT "Number of fluid ounces? ":FO
2040 C=FO*29.57
2045 C=INT(C*100+.5)/100
2050 GOSUB 10500
2055 PRINT C;" Cu.centimeters"
2060 GOTO 240
2065 GOSUB 10000
2070 INPUT "Number of cu.centimeters? ":C
2075 FO=C/29.57
2080 FO=INT(FO*100+.5)/100
2085 GOSUB 10500
2090 PRINT FO;" Fluid ounces"
2095 GOTO 240
2100 GOSUB 10000
2105 PRINT 1;" Quarts to liters?"
2110 PRINT 2;" Liters to quarts?"
2115 GOSUB 10500
2120 INPUT "Which? ":N
2125 ON N GOTO 2130,2165
```

(continued)

```

2130 GOSUB 10000
2135 INPUT "Number of quarts? ":Q
2140 L=((Q*4)*3.7853)/4/4
2145 L=INT(L*100+.5)/100
2150 GOSUB 10500
2155 PRINT L;" Liters"
2160 GOTO 240
2165 GOSUB 10000
2170 INPUT "Number of liters? ":L
2175 Q=4*((L/4)/3.7853)*4
2180 Q=INT(Q*100+.5)/100
2185 GOSUB 10500
2190 PRINT Q;" Quarts"
2195 GOTO 240
2200 GOSUB 10000
2205 PRINT 1;" U.S.gallons to liters?"
2210 PRINT 2;" Liters to U.S.gallons?"
2215 GOSUB 10500
2220 INPUT "Which? ":N
2225 ON N GOTO 2230,2265
2230 GOSUB 10000
2235 INPUT "Number of U.S.gallons? ":G
2240 L=G*3.7853
2245 L=INT(L*100+.5)/100
2250 GOSUB 10500
2255 PRINT L;" Liters"
2260 GOTO 240
2265 GOSUB 10000
2270 INPUT "Number of liters? ":L
2275 G=L/3.7853
2280 G=INT(G*100+.5)/100
2285 GOSUB 10500
2290 PRINT G;" U.S. gallons"
2295 GOTO 240
2300 GOSUB 10000
2305 PRINT 1;" Acres to sq.feet?"
2310 PRINT 2;" Sq.feet to acres?"
2315 GOSUB 10500
2320 INPUT "Which? ":N
2325 ON N GOTO 2330,2365
2330 GOSUB 10000
2335 INPUT "Number of acres? ":A
2340 F=A*43560
2345 GOSUB 10500
2350 PRINT F;" Sq.feet"
2360 GOTO 240
2365 GOSUB 10000
2370 INPUT "Number of sq.feet? ":F
2375 A=F/43560
2380 A=INT(A*100+.5)/100
2385 GOSUB 10500
2390 PRINT A;" Acres"

```

(continued)

---

```
2395 GOTO 240
2400 GOSUB 10000
2405 PRINT 1;" Degrees F. to degrees C."
2410 PRINT 2;" Degrees C. to degrees F."
2415 GOSUB 10500
2420 INPUT "Which? ":N
2425 ON N GOTO 2430,2465
2430 GOSUB 10000
2435 INPUT "Number of degrees F.? ":DF
2440 DC=(DF-32)/1.8
2445 DC=INT(DC*10+.5)/10
2450 GOSUB 10500
2455 PRINT DC;" Degrees celsius"
2460 GOTO 240
2465 GOSUB 10000
2470 INPUT "Number of degrees C.? ":DC
2475 DF=(DC*1.8)+32
2480 DF=INT(DF*10+.5)/10
2485 GOSUB 10500
2490 PRINT DF;" Degrees Fahrenheit"
2495 GOTO 240
10000 CALL CLEAR
10010 RETURN
10500 PRINT "-----"
10510 RETURN
11000 PRINT
11010 INPUT "Press >ENTER< ":Y$
11020 RETURN
12000 GOSUB 10000
12010 PRINT TAB(12);"End"
12020 END
```

---

In most cases the results calculated by the program have been rounded off to two decimals in order to improve legibility. Each time a result is displayed, the program returns to the menu to offer you the choice of running another conversion or exiting the program.

The program is written in TI BASIC, limiting each line to a single statement. If TI EXTENDED BASIC is available, it could be shortened considerably by grouping related statements on a single line.

Line by line:

**Lines 100 and 110** are REMarks.

**Lines 200-530** display the purpose of the program and then the menu with its 20 choices.

**Line 540** sends the computer to one of 20 line numbers, depending on your choice.

---

**Lines 600-695** deal with inch/centimeter conversions, with lines 655 and 690 containing the conversion formulas and lines 656 and 691 rounding the results to two decimals. All the other lines in this section are self-explanatory.

**Lines 700-2495** include all the other conversions and are more or less duplications of the above, needing no special explanation.

**Lines 10000-12020** are the repeatedly used subroutines and the END line.

## AUTHORS AND THEIR WORKS

The next program, Famous Authors, deals with famous authors and their better-known literary works. In structure and operation it is quite similar to the history program we examined earlier. It, too, offers two choices. It will either display the name of the author and ask you to type in his or her most famous work, or it will display the title of a literary work and ask you to identify the author. Again, the second choice is probably more useful than the first, because different readers will think of different works by certain authors as being their most significant literary contribution.

---

### FAMOUS AUTHORS PROGRAM

---

A program dealing with well-known authors and some of their works.

---

```
100 CALL CLEAR
110 GOTO 220
120 REM AUTHORS AND THEIR WORKS
130 REM TI EXTENDED BASIC
140 RANDOMIZE :: Q=INT(RND*30)+1
150 QQ=Q/2 :: QQQ=INT(QQ):: QQ=(QQ/QQQ)*100 :: IF QQ=0
 THEN Q=Q-1
160 IF Q=Z THEN 140 ELSE Z=Q
170 RESTORE :: RETURN
180 FOR X=1 TO 10 :: PRINT :: NEXT X :: RETURN
190 PRINT "-----" :: RETURN
200 PRINT :: INPUT "Press >ENTER< or X to exit ":E$
210 IF E$="X" THEN 870 ELSE RETURN
220 GOSUB 190
230 PRINT "Authors and their works"
240 GOSUB 190 :: GOSUB 180 :: GOSUB 200 :: CALL CLEAR
```

(continued)

---

```
250 PRINT "You have two choices:" :: GOSUB 190
260 PRINT 1;"I display author by name" :: PRINT
270 PRINT 2;"I display the work title" :: GOSUB 190 ::
 GOSUB 180
280 INPUT "Which? ":WHICH
290 CALL CLEAR
300 PRINT "Do want consecutive or" :: PRINT "random
 selection? (C/R) " :: GOSUB 190 :: GOSUB 180
310 INPUT CR$
320 ON WHICH GOTO 330,450
330 CALL CLEAR :: IF CR$="R" THEN GOSUB 140 ELSE GOTO 390
340 FOR XX=1 TO Q :: READ AUTHOR$:: READ WORK$:: NEXT XX
350 PRINT AUTHOR$:: GOSUB 190 :: GOSUB 180
360 INPUT "Name the work: ":W$:: GOSUB 190 :: PRINT ::
 PRINT
370 PRINT WORK$:: GOSUB 190 :: GOSUB 200
380 CALL CLEAR :: GOTO 330
390 RESTORE
400 READ AUTHOR$:: READ WORK$
410 PRINT AUTHOR$:: GOSUB 190 :: GOSUB 180
420 INPUT "Name the work: ":W$:: GOSUB 190 :: PRINT ::
 PRINT
430 PRINT WORK$:: GOSUB 190 :: GOSUB 200
440 CALL CLEAR :: GOTO 400
450 CALL CLEAR :: IF CR$="R" THEN GOSUB 140 ELSE GOTO 510
460 FOR XXX=1 TO Q :: READ AUTHOR$:: READ WORK$:: NEXT
 XXX
470 PRINT WORK$:: GOSUB 190 :: GOSUB 180
480 INPUT "Name the author: ":A$:: GOSUB 190 :: PRINT ::
 PRINT
490 PRINT AUTHOR$:: GOSUB 190 :: GOSUB 200
500 CALL CLEAR :: GOTO 450
510 RESTORE
520 READ AUTHOR$:: READ WORK$
530 PRINT WORK$:: GOSUB 190 :: GOSUB 180
540 INPUT "Name the author: ":A$:: GOSUB 190 :: PRINT ::
 PRINT
550 PRINT AUTHOR$:: GOSUB 190 :: GOSUB 200
560 CALL CLEAR :: GOTO 520
570 DATA THEODORE DREISER,AN AMERICAN TRAGEDY
580 DATA LEON TOLSTOY,ANNA KARENINA
590 DATA SINCLAIR LEWIS,ARROWSMITH
600 DATA LEW WALLACE,BEN HUR
610 DATA ALDOUS HUXLEY,BRAVE NEW WORLD
620 DATA THORNTON WILDER,THE BRIDGE OF SAN LUIS REY
630 DATA THOMAS MANN,BUDDENBROOKS
640 DATA FYODOR DOSTOIEVSKI,THE BROTHERS KARAMAZOV
650 DATA ALEXANDRE DUMAS,CAMILLE
660 DATA RUDYARD KIPLING,CAPTAINS COURAGEOUS
670 DATA LEWIS CARROLL,ALICE IN WONDERLAND
680 DATA CHARLES DICKENS,DAVIF COPPERFIELD
```

(continued)

```
690 DATA KARL MARX,DAS KAPITAL
700 DATA STENDAH,THE CHARTERHOUSE OF PARMA
710 DATA JAMES FENIMORE COOPER,THE LAST OF THE MOHICANS
720 DATA DANTE ALIGHIERI,THE DIVINE COMEDY
730 DATA ROBERT LOUIS STEVENSON,DR. JEKYLL AND MR. HYDE
740 DATA HENRIK IBSEN,A DOLL'S HOUSE
750 DATA ERNEST HEMINGWAY,A FAREWELL TO ARMS
760 DATA JOHN GALSWORTHY,THE FORSYTE SAGA
770 DATA MARY SHELLEY,FRANKENSTEIN
780 DATA JOHN STEINBECK,THE GRAPES OF WRATH
790 DATA F.SCOTT FITZGERALD,THE GREAT GATSBY
800 DATA JONATHAN SWIFT,GULLIVER'S TRAVELS
810 DATA MARK TWAIN,HUCKLEBERRY FINN
820 DATA VICTOR HUGO,THE HUNCHBACK OF NOTRE DAME
830 DATA HOMER,THE ILIAD
840 DATA SIR WALTER SCOTT,IVANHOE
850 DATA ADOLF HITLER,MEIN KAMPF
860 DATA CHARLOTTE BRONTE,JANE EYRE
870 CALL CLEAR :: GOSUB 190 :: PRINT TAB(12);"End." ::
 GOSUB 190 :: GOSUB 180 :: END
```

---

The program is written in TI EXTENDED BASIC in order to keep the number of lines to a minimum. It will work just as well in TI BASIC, but then each statement must be placed on a separate line number. If you do convert the program to TI BASIC, be sure to use the right line numbers in conjunction with the different GOTO, GOSUB, and IF...THEN...ELSE statements.

This program, like the history lesson, can be modified simply by changing the items in the DATA block. Thus, it need not necessarily deal with authors and literature. It could just as well be used to determine states and their capitals, countries and their capitals or the continents on which they are located, composers and their works, or any other two subjects that bear a relationship to one another.

I have added one choice that was not included in the history program, the choice of having the authors or their works displayed in random fashion or in consecutive order (the order in which they are listed in the DATA block).

When the program is run, it displays its title and your choices:

---

Authors and their works.  
-----

You have two choices:  
-----

1 I display author by name  
2 I display the work title  
-----

Which?

Do you want consecutive or  
random selection?

When both questions have been answered, it either displays the name of an author, asking you to type in the title of the work, or vice versa. In each case it then displays the associated title or author before going on to the next. You can exit the program at any time by typing X.

Line by line:

**Line 100** clears the screen.

**Line 110** tells the computer to skip over the group of REMarks and subroutines in lines 120-210 and go to line 220.

**Lines 120 and 130** are REMarks.

**Lines 140-170** generate a random number between 1 and 30 (the number of author/work combinations used) and assign that number to the numeric variable Q. Line 150 makes sure that the value assigned to Q is an odd number. Line 160 is used to prevent displaying the same selection twice in a row. Occasional repetitions are unavoidable when the random mode is selected, but we don't want the same selection to appear several times in succession. Line 170 makes sure that DATA are READ from the beginning and then RETURNS the computer.

**Line 180** is a subroutine that places 10 blank lines into the display in order to position copy at screen center.

**Line 190** is a subroutine that places a dashed line into display.



**Lines 200 and 210** are the subroutine that asks you to press >ENTER< in order to continue program execution or X to exit the program.

**Lines 220-310** place the title and the various choices into display, assigning your choice of authors or their works to the numeric variable WHICH and your choice of consecutive versus random to the string variable CR\$.

**Line 320** sends the computer to one of two line numbers, depending on the choice represented by WHICH.

**Line 330** clears the screen, checks the value assigned to CR\$, and sends the computer either to the subroutine that creates the random number or to line 390.

**Line 340** produces a loop that causes the author's name and the work title to be READ as many times as the number represented by Q, assigning the name to the string variable AUTHOR\$ and the work title to the string variable WORK\$.

**Line 350** places the name of the author into display.

**Line 360** asks you to type in the title of the work.

**Line 370** displays the work title stored in the program, giving you a chance to compare the two.

**Line 380** clears the screen and returns the computer to line 330 for the next random selection.

**Line 390** returns the DATA pointer to the top.

**Line 400** causes the computer to READ the first author name and work title.

**Line 410** displays the author's name.

**Line 420** asks you to type in the associated work title.

**Line 430** displays the work title stored in the program.

**Line 440** clears the screen and returns the computer to line 400 for the next consecutive selection.

**Lines 450-560** are identical to the above, the only difference being that the work title is displayed and you're asked to type in the author's name.

**Lines 570-860** represent the DATA block in which the author's names and the work titles are stored.

**Line 870** is the END line.

There is no particular reason to limit the number of author / work combinations to 30. You can add as many as you like. The only

---

changes that would be required are to change the number associated with the RND statement in line 140 to reflect the total number of DATA pairs. Furthermore, the END line would have to be moved down, and the line number used in line 210 in conjunction with the  $E\$ = "X"$  equation would have to be changed accordingly.

## **OBJECT RECOGNITION**

The last program in this group, the Children's Counting Game is designed for the relatively young. It places a number of shapes on the screen and asks the child to count and key in the number. How long the shapes remain in display is controlled by the user, and can be from as little as approximately a half a second to several seconds. What makes the program challenging is the fact that the shapes move up and down and across the screen at different rates of speed, and it takes a certain amount of concentration to determine the actual number. The program uses the graphics capabilities of the TI-99/4A, which are described in detail in Chapter 11. It is designed to be used with a color monitor and with the voice synthesizer, although it can be used without the latter. It does, however, require TI EXTENDED BASIC.

## **CHILDREN'S COUNTING GAME**

When the Children's Counting Game Program is run it first displays a line of copy that, among other things, asks if you have a voice synthesizer. If none is available, the subsequent information is printed. If one is available, the screen changes to blue with all shapes in display (see Figure 10-1) while the voice announces:

"In this program you must type in the number of  
shapes on the screen. What time period do you want?  
One is short. Ten is long. Type a number  
from 1 to 10."

As soon as a number has been typed in and >ENTER< has been pressed, the display is blanked out and three airplane-like shapes

---

travel across the screen from left to right for a period of time determined by that number. The screen then goes blank and the voice asks:

"What number?"

After the child has entered the number of objects, the voice responds with:

"That is correct"

or:

"That is not correct, try again."

If the answer is correct, the next group of shapes appears. If it is incorrect, the previous group is displayed again for another try.

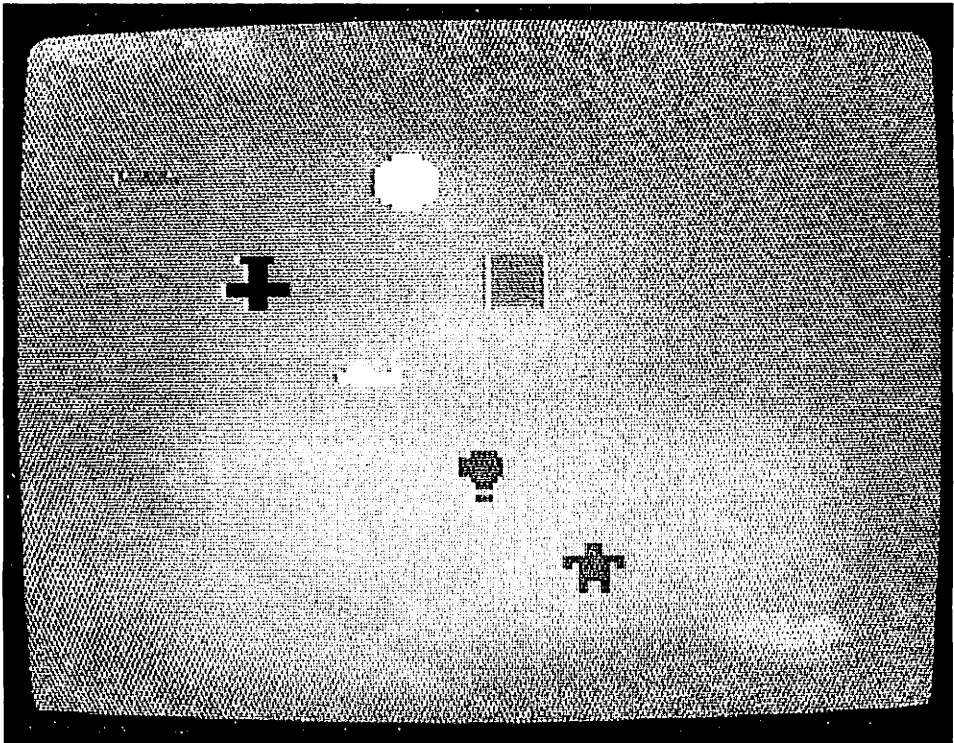


Figure 10-1. The initial display while the synthesizer speaks.

---

There are 10 such groups of objects in all, and recognition becomes increasingly difficult as the objects appear in different colors and move at considerable speed.

### CHILDREN'S COUNTING GAME PROGRAM

This program combines voice, color, and graphics to test your ability to recognize the number of objects on the screen.

```

100 REM COUNTING OBJECTS
110 REM TI EXTENDED BASIC
120 CALL CLEAR
130 GOTO 170
140 PRINT "-----" :: RETURN
150 FOR X=1 TO 8 :: PRINT :: NEXT X :: RETURN
160 INPUT "PRESS >ENTER<":E$:: RETURN
170 A$="00008EFF00000000"
180 B$="3C181818FFFF1818"
190 C$="000071FF00000000"
200 D$="3C7E7E7E3C180018"
210 E$="1818FFBD3C3C2424"
220 F$="3C7EFFFFFFFF7E3C"
230 G$="FFFFFFFFFFFFFFFF"
240 GOSUB 140
250 PRINT "This program is designed to be used with the
 voice syn- thesizer, but it can be run without it." ::
 GOSUB 140 :: GOSUB 150
260 PRINT "Do you have a voice synthe- sizer? (Y/N)" ::
 INPUT Y$
270 CALL CLEAR
280 IF Y$="N" THEN GOSUB 1250
290 CALL CHAR(96,A$)
300 CALL CHAR(97,B$)
310 CALL CHAR(98,C$)
320 CALL CHAR(99,D$)
330 CALL CHAR(100,E$)
340 CALL CHAR(101,F$)
350 CALL CHAR(102,G$)
360 CALL SCREEN(6)
370 CALL SPRITE(#1,96,7,30,30)
380 CALL SPRITE(#2,97,2,60,60)
390 CALL SPRITE(#3,98,16,90,90)
400 CALL SPRITE(#4,99,13,120,120)
410 CALL SPRITE(#5,100,5,150,150)
420 CALL SPRITE(#6,101,12,30,100)
430 CALL SPRITE(#7,102,3,60,130)
440 CALL MAGNIFY(2)
450 IF Y$="XX" THEN 500

```

(continued)

```
460 CALL SAY("IN THIS PROGRAM YOU MUST TYPE IN THE NUMBER
 OF SHAPES ON THE SCREEN")
470 CALL SAY("WHAT TIME PERIOD DO YOU WANT. 1 IS SHORT.
 TEN IS LONG")
480 CALL SAY("TYPE A NUMBER FROM 1 TO TEN")
490 INPUT N
500 N=N*300
510 CALL DELSPRITE(ALL)
520 PASS=1
530 CALL SPRITE(#8,96,8,50,1,0,25)
540 CALL SPRITE(#9,96,8,80,1,0,25)
550 CALL SPRITE(#10,96,8,110,1,0,25)
560 NS=3
570 GOSUB 590
580 GOTO 690
590 FOR PAUSE=1 TO N :: NEXT PAUSE
600 CALL DELSPRITE(ALL)
610 IF Y#="XX" THEN 1290
620 CALL SAY("WHAT NUMBER")
630 CALL CLEAR :: INPUT NN
640 IF NN=NS THEN 670 ELSE 650
650 CALL SAY("THAT IS NOT CORRECT. TRY AGAIN")
660 ON PASS GOTO 530,700,780,820,910,950,1010,1050,1080,
 1150
670 CALL SAY("THAT IS CORRECT")
680 RETURN
690 PASS=2
700 CALL SPRITE(#11,99,7,150,50,-10,0)
710 CALL SPRITE(#12,99,7,150,80,-10,0)
720 CALL SPRITE(#13,99,7,150,110,-10,0)
730 CALL SPRITE(#14,99,7,150,140,-10,0)
740 CALL SPRITE(#15,99,7,100,95,-10,0)
750 CALL SPRITE(#16,99,7,100,125,-10,0)
760 NS=6
770 GOSUB 590 :: GOTO 780
780 PASS=3
790 CALL SPRITE(#17,97,2,1,50,30,0)
800 CALL SPRITE(#18,97,2,1,100,30,0)
810 NS=2 :: GOSUB 590
820 PASS=4
830 CALL SPRITE(#19,102,5,10,10,0,50)
840 CALL SPRITE(#20,102,5,30,30,0,50)
850 CALL SPRITE(#21,102,5,50,50,0,50)
860 CALL SPRITE(#22,102,5,70,70,0,50)
870 CALL SPRITE(#23,102,5,90,90,0,50)
880 CALL SPRITE(#24,102,5,110,110,0,50)
890 CALL SPRITE(#25,102,5,130,130,0,50)
900 NS=7 :: GOSUB 590
910 PASS=5
920 CALL SPRITE(#26,100,7,50,50,20,30)
930 CALL SPRITE(#27,100,5,50,200,-20,-30)
```

(continued)

---

---

```
940 NS=2 :: GOSUB 590
950 PASS=6
960 CALL SPRITE(#28,101,14,40,40,5,60)
970 CALL SPRITE(#1,101,9,70,100,0,-25)
980 CALL SPRITE(#2,101,5,15,30,5,25)
990 CALL SPRITE(#3,101,16,140,100,-60,0)
1000 NS=4 :: GOSUB 590
1010 PASS=7
1020 CALL SPRITE(#4,96,13,50,1,0,80)
1030 CALL SPRITE(#5,98,7,60,250,0,-80)
1040 NS=2 :: GOSUB 590
1050 PASS=8
1060 CALL SPRITE(#6,102,12,1,1,70,90)
1070 NS=1 :: GOSUB 590
1080 PASS=9
1090 CALL SPRITE(#7,100,4,10,240,5,-5)
1100 CALL SPRITE(#8,100,5,30,220,5,5)
1110 CALL SPRITE(#9,100,7,100,30,-5,5)
1120 CALL SPRITE(#10,100,13,70,100,5,-5)
1130 CALL SPRITE(#11,100,12,150,150,-5,-5)
1140 NS=5 :: GOSUB 590
1150 PASS=10
1160 CALL SPRITE(#12,97,16,1,125,50,0)
1170 CALL SPRITE(#13,98,16,60,30,0,-50)
1180 CALL SPRITE(#14,96,16,30,200,0,50)
1190 CALL SPRITE(#15,99,16,130,80,-10,0)
1200 CALL SPRITE(#16,100,16,120,10,0,5)
1210 CALL SPRITE(#17,101,2,1,1,25,35)
1220 CALL SPRITE(#18,102,7,1,250,25,-35)
1230 NS=7 :: GOSUB 590
1240 END
1250 PRINT "In this program you must type in the number
 of shapeson the screen." :: PRINT
1260 PRINT "What time period do you want? 1 is short,
 10 is long. Type a number from 1 to 10" ::
 GOSUB 140 :: GOSUB 150
1270 INPUT N :: CALL CLEAR
1280 Y$="XX" :: RETURN
1290 CALL CLEAR :: PRINT "HOW MANY?" :: INPUT NN
1300 IF NN=NS THEN 1340 ELSE 1310
1310 PRINT "THAT IS WRONG. TRY AGAIN"
1320 GOSUB 160
1330 CALL CLEAR :: GOTO 660
1340 PRINT "THAT IS RIGHT."
1350 GOSUB 160 :: PASS=PASS+1
1360 IF PASS>10 THEN END
1370 CALL CLEAR :: GOTO 660
```

---

---

Line by line:

**Lines 100 and 110** are REMarks.

**Line 120** clears the screen.

**Line 130** tells the computer to skip the next three lines, which contain three frequently used subroutines.

**Lines 170–230** assign alphanumeric strings to string variables. These strings determine the shapes of the objects (see Chapter 11).

**Lines 250 and 260** explain the program and ask if a voice synthesizer is available.

**Line 280** sends the computer to a subroutine if a voice synthesizer is not available. The subroutine (lines 1250–1280) places the words that otherwise would be spoken into display.

**Lines 290–350** assign the previously determined shapes to character numbers (96–102) for subsequent use.

**Line 360** assigns color number 6 (dark blue) to the screen.

**Lines 370–430** place the seven objects into display at specified places on the screen.

**Line 440** doubles the size of the objects.

**Line 450** tells the computer to skip the next four lines if the voice synthesizer is not available.

**Lines 460–480** activate the voice synthesizer and cause it to tell the student to type in a number from 1 to 10 to determine the length of time for which the objects will be displayed.

**Line 490** assigns the input number to the numeric variable N.

**Line 500** multiplies the value of N by 300.

**Line 510** clears the objects from the screen.

**Line 520** identifies the first moving display by assigning the value of 1 to the numeric variable PASS.

**Lines 530–550** cause three shapes to move across the screen. The numbers in parentheses represent the number of the object, the number assigned to the shape, the color (8), the vertical position on the screen, the horizontal starting position on the screen (1), the vertical velocity (0), and the horizontal velocity (25). For details, see Chapter 11.

**Line 560** assigns the number of objects on the screen (3) to the numeric variable NS.

**Line 570** sends the computer to a subroutine (lines 590–680), which is explained below.

---

**Line 580** is used after the subroutine has been executed, sending the computer to line 690.

**Line 590** uses a loop to produce the time period during which the objects remain on the screen.

**Line 600** clears them from the screen.

**Line 610** once more checks if the voice synthesizer is being used.

**Line 620** activates the voice synthesizer.

**Line 630** clears the screen and assigns the child's answer to the numeric variable NN.

**Line 640** checks whether or not N matches NN and then, sends the computer to one of two line numbers, either to repeat the last display or to go on to the next one.

**Line 650** activates the voice synthesizer if the answer is wrong.

**Line 660** sends the computer to one of 10 line numbers, depending on the value of PASS.

**Line 670** activates the voice synthesizer if the answer is right.

**Line 680** returns the computer to line 580, which then sends it to line 690.

**Lines 690-1240** are repeats of the above, placing the other nine groups of shapes into display.

**Lines 1250-1370** contain the information that must be displayed if the voice synthesizer is not being used.

---



---

# 11

---

## ***Sprites and Other Graphics***

---

---

One of the most fascinating aspects of the TI-99/4A Home Computer is its multitude of graphics capabilities. The computer is equipped with a large number of subprograms that permit you to create all kinds of graphic images. The problem is that the methods involved are a bit complicated, and it usually requires a considerable amount of trial and error to become reasonably proficient in their use.

In this chapter we'll write a number of programs that illustrate the use of these subprograms. I suggest that you follow me through, one step at a time, as I explain the various graphics capabilities of the TI-99/4A.

### **CALL HCHAR**

To start, let's draw a simple picture of a tree, using only one of the most basic statements. The statement is

CALL HCHAR

where HCHAR stands for horizontal character. The program that draws a tree is shown on page 281. The picture that results is a ground surface line, a straight tree trunk made up of the letters H, and a triangular crown made up of the letter A. Figure 11-1 shows the final picture. Let's look at each line and see what happens.

---

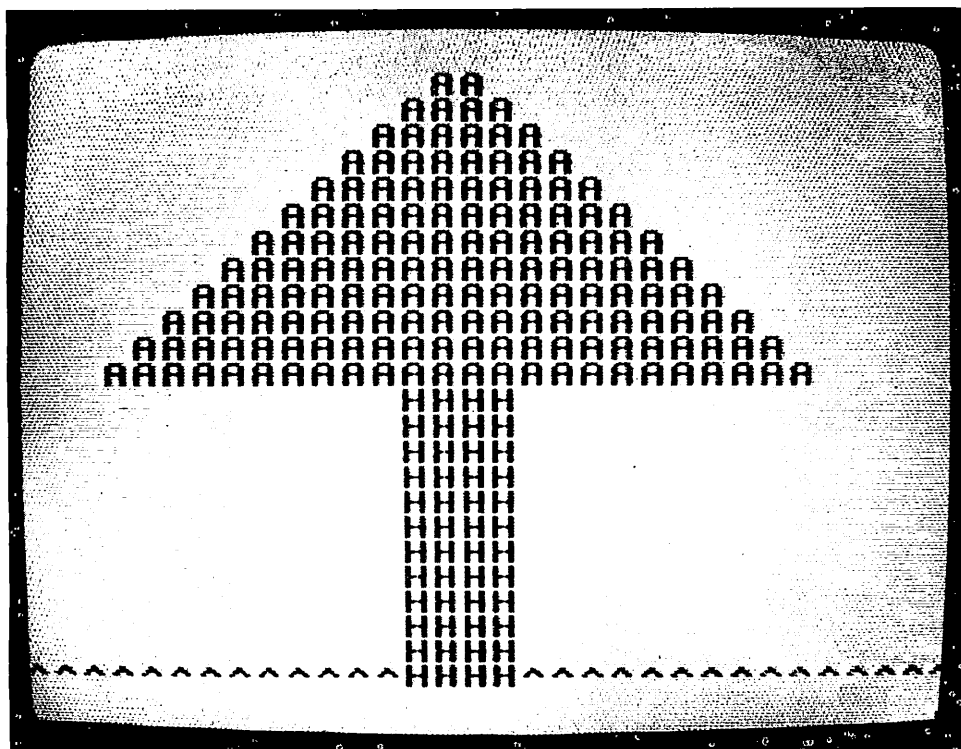


Figure 11-1. The display of the symmetrical tree.

### CALL HCHAR DEMONSTRATION PROGRAM

This program creates a symmetrical tree.

```

100 CALL CLEAR
110 WIDTH=16
120 ROW=12
130 CALL HCHAR(24,1,94,32)
140 FOR X=1 TO 12
150 WIDTH=WIDTH-1
160 REP=REP+2
170 HEIGHT=HEIGHT+1
180 CALL HCHAR(HEIGHT,WIDTH,65,REP)
190 NEXT X
200 FOR X=1 TO 12
210 ROW=ROW+1
220 CALL HCHAR(ROW,14,72,4)
230 NEXT X
240 GOTO 240

```



the ASCII character code for the letter H, and 4 is the number of times H is to be printed on each row.

**Line 230** completes the second loop.

**Line 240** makes sure the picture remains on the screen. To exit the program, press FCTN 4 (CLEAR).

What we have learned is that CALL HCHAR can be used to place a given character anywhere on the screen, repeating it horizontally as often as we like. If the fourth value, the one controlling the number of repetitions, is left out, only a single character will be printed.

## CALL VCHAR

The companion statement to CALL HCHAR is CALL VCHAR, which is identical except that it produces vertical rather than horizontal repetitions. The CALL VCHAR Demonstration Program is another short program, one that uses both statements to produce a picture of a box, as seen in Figure 11-2 on the following page.

### THE CALL VCHAR DEMONSTRATION PROGRAM

This program creates a box.

```

100 CALL CLEAR
110 REM GRAPHIC BOX
120 CALL VCHAR(6,2,91,18)
130 CALL VCHAR(2,6,91,18)
140 CALL VCHAR(2,30,93,18)
150 CALL HCHAR(2,6,95,25)
160 CALL HCHAR(23,2,95,25)
170 CALL HCHAR(20,6,95,25)
180 ROW=23
190 COL=2
200 FOR X=1 TO 3
210 ROW=ROW-1
220 COL=COL+1
230 CALL HCHAR(ROW,COL,47)
240 NEXT X
250 ROW=6
260 COL=2
270 FOR X=1 TO 3
280 ROW=ROW-1

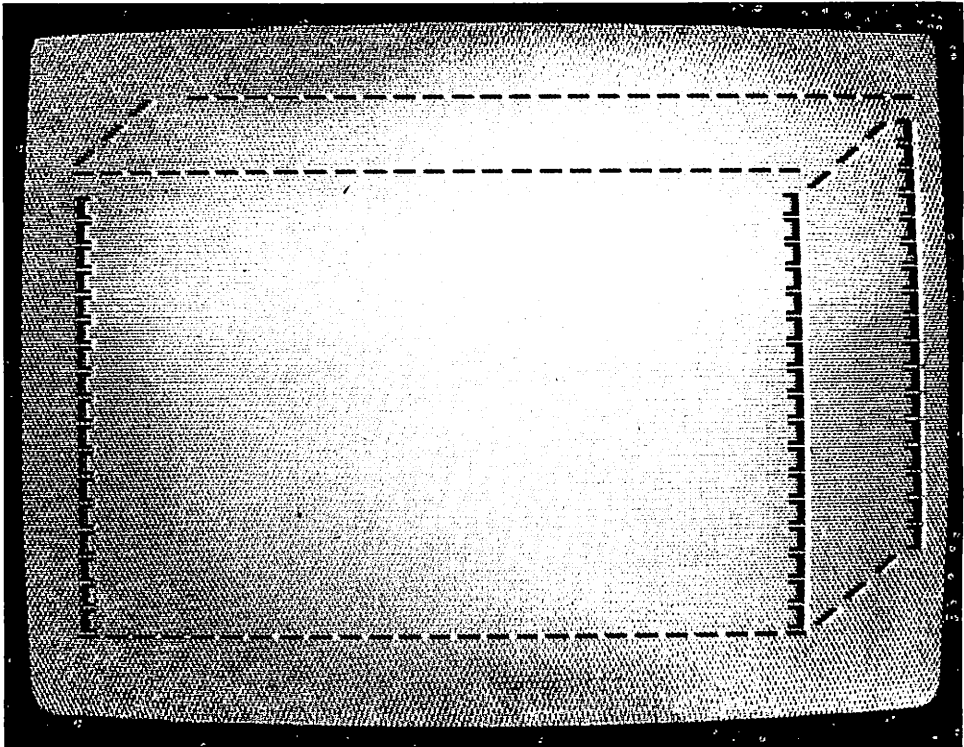
```

(continued)

```
290 COL=COL+1
300 CALL HCHAR(ROW,COL,47)
310 NEXT X
320 ROW=24
330 COL=27
340 FOR X=1 TO 3
350 ROW=ROW-1
360 COL=COL+1
370 CALL HCHAR(ROW,COL,47)
380 NEXT X
390 GOTO 390
```

---

**Lines 120-140** place three vertical lines into display, each consisting of 18 symbols, using left and right brackets (ASCII character codes 91 and 93), with the position of the first of the 18 characters determined by the first (row) and second (column) numbers.



**Figure 11-2.** The display of the box.

---

**Lines 150–170** place three horizontal lines into display, using underlines (ASCII character code 95).

**Lines 180 and 190, 250 and 260, and 320 and 330** assign values to the numeric variables ROW and COL.

**Lines 200–240, 270–310, and 340–380** represent three loops where the values of the numeric variables are increased and decreased in order to cause diagonal lines to be displayed, using slashes (ASCII character code 47) in three different places. Here the number controlling repetitions was left out because only one character is to be printed during each pass through the loop. The finished box is shown in Figure 11-2.

## CALL CHAR

So far we have contented ourselves with using characters that can be identified by the ASCII character codes numbered from 30 through 95 (see Appendix I). More often than not, however, we're likely to want to use characters of our own design. That, too, is possible, and the statement to use is CALL CHAR. But now we're getting into some further complications. Each character is the size of the average capital letter, and each character consists of a total of 64 dots. Figure 6-1 shows what such a box looks like. Now, each little square can be turned on or off. If all are turned on, you get a black (or colored) square. If all are turned off, you get nothing. The method for determining which squares we want to turn on and which are to be left off requires cutting the block in half, producing two vertical rectangles consisting of 32 squares each (see Figure 6-2). The ON/OFF positions for each square are determined by a total of 16 hexadecimal numbers, with each individual number determining the condition of four horizontally arranged squares. Figure 6-3 illustrates which numbers (and letters, since the hexadecimal system uses letters for values from 9 to 16) produce what results.

The CALL CHAR Demonstration Program #1 produces a number of different symbols and will serve as a good way to illustrate how this statement works.

---

**CALL CHAR DEMONSTRATION PROGRAM #1**

This program produces 10 different shapes.

---

```
100 CALL CLEAR
110 REM CHARACTER SHAPES
120 A$="FFFFFFFFFFFFFFFF"
130 B$="3C7EFFFFFFFF7E3C"
140 C$="000000FFFF000000"
150 D$="2B2B2B2B2B2B2B2B"
160 E$="0000001100000000"
170 F$="FFB001B001B001FF"
180 G$="FF000000000000FF"
190 H$="1B1BFFBD3C3C2424"
200 J$="00B0C0E0F0F8FCFE"
210 K$="FCF8F0E0C0B00000"
220 CALL CHAR(96,A$)
230 CALL HCHAR(2,10,96,10)
240 CALL CHAR(97,B$)
250 CALL HCHAR(4,10,97,10)
260 CALL CHAR(98,C$)
270 CALL HCHAR(6,10,98,10)
280 CALL CHAR(99,D$)
290 CALL HCHAR(8,10,99,10)
300 CALL CHAR(100,E$)
310 CALL HCHAR(10,10,100,10)
320 CALL CHAR(101,F$)
330 CALL HCHAR(12,10,101,10)
340 CALL CHAR(102,G$)
350 CALL HCHAR(14,10,102,10)
360 CALL CHAR(103,H$)
370 CALL HCHAR(16,10,103,10)
380 CALL CHAR(104,J$)
390 CALL HCHAR(18,10,104,10)
400 CALL CHAR(105,K$)
410 CALL HCHAR(20,10,105,10)
420 GOTO 420
```

---

**Lines 120 through 210** assign different patterns to string variables.

- A\$ produces a solid square.
  - B\$ produces a solid ball.
  - C\$ produces a narrow horizontal line.
  - D\$ produces vertical bars.
  - E\$ produces short horizontal dashes.
-

F\$ produces an empty square.  
 G\$ produces two narrow horizontal lines.  
 H\$ produces a little man.  
 J\$ produces a right triangle facing up.  
 K\$ produces a right triangle facing down.

**Lines 220, 240, 260, etc.,** assign an identification number to a character (96–105) and use the string variable to determine the shape of that character.

**Lines 230, 250, 270, etc.,** cause the characters to be printed on rows 2, 4, 6, etc., starting at column 10 and repeating each character 10 times.

I wrote the program in the form described in order to give the clearest possible picture. In practice it could be shortened, and you could save a fair amount of typing (and reduce the chance of making typing errors) by using CALL CHAR Demonstration Program #2 which produces the result shown in Figure 11-3 found on the next page.

### CALL CHAR DEMONSTRATION PROGRAM #2

The shape program, simplified.

```

100 CALL CLEAR
110 REM CHARACTER SHAPES
120 A$="FFFFFFFFFFFFFFFF"
130 B$="3C7EFFFFFFFF7E3C"
140 C$="000000FFFF000000"
150 D$="2828282828282828"
160 E$="0000001100000000"
170 F$="FF800180018001FF"
180 G$="FF000000000000FF"
190 H$="1818FFBD3C3C2424"
200 J$="0080C0E0F0F8FCFE"
210 K$="FCFBF0E0C0800000"
220 CALL CHAR(96,A$)
230 CALL CHAR(97,B$)
240 CALL CHAR(98,C$)
250 CALL CHAR(99,D$)
260 CALL CHAR(100,E$)
270 CALL CHAR(101,F$)
280 CALL CHAR(102,G$)
290 CALL CHAR(103,H$)

```

(continued)



```
300 CALL CHAR(104,J$)
310 CALL CHAR(105,K$)
320 ROW=2
330 IDENT=96
340 FOR X=1 TO 9
350 ROW=ROW+2
360 IDENT=IDENT+1
370 CALL HCHAR(ROW,10,IDENT,10)
380 NEXT X
390 GOTO 390
```

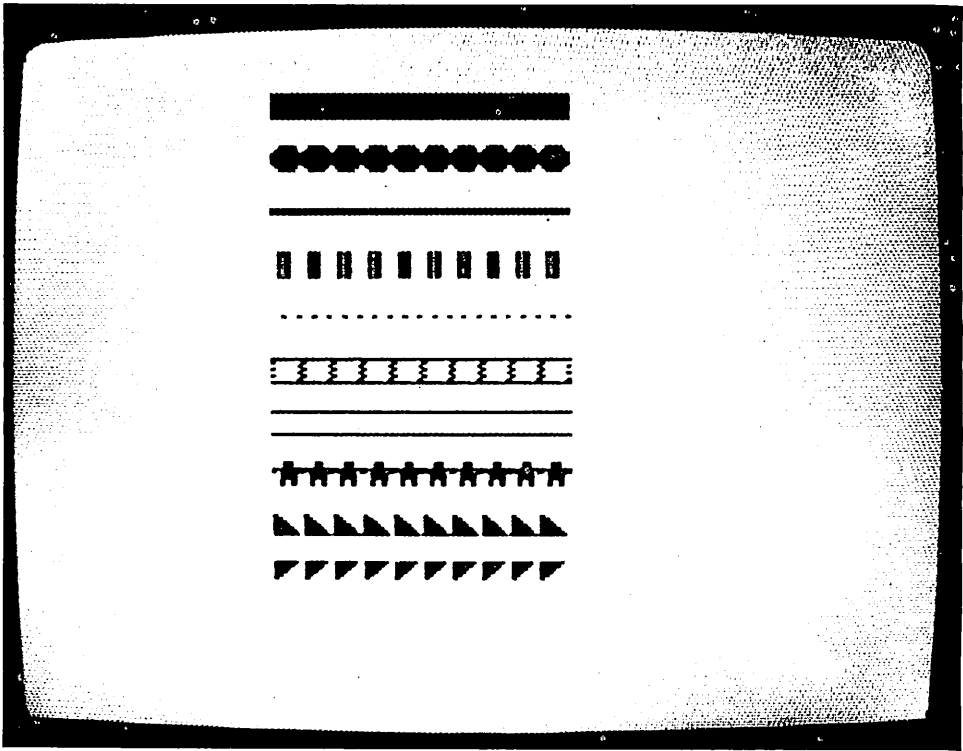


Figure 11-3. The display produced by Figure 11-5 or Figure 11-6.

## **CALL COLOR AND CALL SCREEN**

So far we have dealt with characters and character shapes, but have ignored colors. There are two subprograms that deal with colors, **CALL COLOR** and **CALL SCREEN**, where the first controls the colors of the characters, and the latter the color of the screen as a

---

whole. Different color monitors react differently, and you may have to do a bit of experimenting, but as a general rule, the following numbers produce these colors:

|               |                |                 |
|---------------|----------------|-----------------|
| 1 transparent | 2 black        | 3 medium green  |
| 4 light green | 5 dark blue    | 6 light blue    |
| 7 dark red    | 8 cyan         | 9 medium red    |
| 10 light red  | 11 dark yellow | 12 light yellow |
| 13 dark green | 14 magenta     | 15 gray         |
| 16 white      |                |                 |

Before we go on, there are a few items that need to be understood. You can specify a foreground color and a background color for each character. The background color has nothing to do with the screen color. It is simply the color of the square on which the character is printed. When no color command is given, the character is printed in black and the background color is transparent, causing the screen color to surround the character, as we saw when we ran the previous program. In addition, the CALL COLOR statement requires us to identify the character set that matches the character code we used. The character sets are identified by 15 numbers, all except one representing a group of eight ASCII character codes:

| Set No. | ASCII Code | Set No. | ASCII Code |
|---------|------------|---------|------------|
| 0       | 30-31      | 1       | 32-39      |
| 2       | 40-47      | 3       | 48-55      |
| 4       | 56-63      | 5       | 64-71      |
| 6       | 72-79      | 7       | 80-87      |
| 8       | 88-95      | 9       | 96-103     |
| 10      | 104-111    | 11      | 112-119    |
| 12      | 120-127    | 13      | 128-135    |
| 14      | 136-143    |         |            |

In the previous program we used character codes 96-105, which fall into character sets 9 and 10. The convention is

100 CALL COLOR (9,7,1)

which produces dark red characters on a transparent background.

---

The statement controlling the screen color is

100 CALL SCREEN (number)

where the number is one of the 16 color numbers.

Now, in order to demonstrate, let's add some color instructions to our previous program. The new version called Color CALL CHAR Program displays all possible color combinations, one after another. It first asks you to key in the speed at which you want the program to run, because even at a very fast speed, using a low number such as 2 or 3, the program takes more than 3 minutes to go through all the steps. It first shows all possible colors for the character shapes, using transparent as the background color; then it goes through all screen colors in combination with all character colors; finally, it goes through all background colors in combination with all foreground colors and all screen colors. Occasionally the images briefly disappear from the screen, because the color is either transparent or identical to the screen color.

### COLOR CALL CHAR PROGRAM

Here we have added color instructions to the program.

---

```
100 CALL CLEAR
110 REM CHARACTER SHAPES
120 INPUT "SPEED? ":SPEED
130 BG=1
140 CALL CLEAR
150 A$="FFFFFFFFFFFFFFFF"
160 B$="3C7EFFFFFFFF7E3C"
170 C$="000000FFFF000000"
180 D$="2828282828282828"
190 E$="0000001100000000"
200 F$="FF800180018001FF"
210 G$="FF000000000000FF"
220 H$="1818FFBD3C3C2424"
230 J$="0080C0E0F0F8FCFE"
240 K$="FCF8F0E0C0800000"
250 CALL CHAR(96,A$)
260 CALL CHAR(97,B$)
270 CALL CHAR(98,C$)
280 CALL CHAR(99,D$)
```

(continued)

---

```
290 CALL CHAR(100,E$)
300 CALL CHAR(101,F$)
310 CALL CHAR(102,G$)
320 CALL CHAR(103,H$)
330 CALL CHAR(104,J$)
340 CALL CHAR(105,K$)
350 GOSUB 440
360 ROW=2
370 IDENT=96
380 FOR X=1 TO 9
390 ROW=ROW+2
400 IDENT=IDENT+1
410 CALL HCHAR(ROW,10,IDENT,10)
420 NEXT X
430 RETURN
440 FOR FG=1 TO 14
450 CALL COLOR(9,FG,BG)
460 CALL COLOR(10,FG,BG)
470 GOSUB 360
480 FOR PAUSE=1 TO SPEED :: NEXT PAUSE
490 NEXT FG
500 FIELD=FIELD+1
510 IF FIELD>14 THEN 550
520 CALL SCREEN(FIELD)
530 FOR PAUSE=1 TO SPEED :: NEXT PAUSE
540 GOTO 350
550 BG=BG+1
560 IF BG>14 THEN 580
570 GOTO 350
580 END
```

---

**Line 120** asks you to key in the speed at which you want the program to run. Five is a good number if you're in a hurry. If you want time to study the different color combinations, a number between 100 and 500 will be better.

**Line 130** assigns a value of 1 (transparent) to the numeric variable BG, which controls the background color.

**Lines 150-430** are the same as discussed in the previous version of the program.

**Line 440** raises the value assigned to the numeric variable FG by 1 during each pass, controlling the foreground colors.

**Lines 450 and 460** control the colors, where 9 and 10 represent the two character code sets used in conjunction with the CALL CHAR statements. FG controls the foreground colors and BG controls the background colors.

---

**Line 470** sends the computer to the subroutine (lines 360–430) that causes the characters to be displayed on the screen.

**Lines 480 and 530** control the speed at which the program runs.

**Line 490** completes the foreground color loop.

**Line 500** increases the value assigned to the numeric variable FIELD by 1 during each pass, controlling the screen colors.

**Line 510** checks whether all 16 colors have been used, in which case the computer is sent to line 550.

**Line 520** assigns the value of FIELD to the screen, changing its color during each pass.

**Line 540** sends the computer to line 350 to cause all foreground colors to be displayed with each screen color.

**Line 550** raises the value assigned to the numeric variable BG by 1 during each pass, controlling the background colors.

**Line 560** checks whether all background colors have been used, in which case the computer is sent to line 580, the END line.

**Line 570** sends the computer to line 350 to go through all foreground colors and all screen colors in combination with each background color once more.

**Line 580** is the END line.

So far we have created various character shapes and placed them into certain places on the screen. Now let's go one step further and move them around. The Moving Object Program takes a black square and moves it from the upper left-hand corner of the screen diagonally down and toward the right. It then returns, leaving a trail of black squares, until it gets back to where it came from, only to start the process all over again.

### THE MOVING OBJECT PROGRAM

This program moves an object around on the screen.

---

```
100 REM MOVING OBJECT
110 CALL CLEAR
120 A$="FFFFFFFFFFFFFFF"
130 CALL CHAR(96,A$)
140 H=H+1
150 V=V+1
160 IF H=31 OR V=23 THEN 210
170 CALL HCHAR(V,H,96)
```

(continued)

---

---

```

180 FOR PAUSE=1 TO 50 :: NEXT PAUSE
190 CALL CLEAR
200 GOTO 140
210 H=H-1
220 V=V-1
230 CALL HCHAR(V,H,96)
240 IF H=1 OR V=1 THEN 140 ELSE 210

```

---

**Line 120** defines the character shape and assigns it to the string variable A\$.

**Line 130** assigns number 96 to the character shape.

**Lines 140 and 150** increase the values assigned to the numeric variables H and V by 1 during each pass, where H controls the column number and V represents the row number.

**Line 160** is used to limit the distance of travel.

**Line 170** displays character 96 in row V and column H.

**Line 180** controls the speed of travel.

**Line 190** clears the previous position of the character before displaying it in the next position.

**Line 200** causes the action to be repeated by sending the computer back to line 140.

**Lines 210 and 220** reduce the values assigned to the two numeric variables by 1 during each pass.

**Line 230** is identical to line 170.

**Line 240** limits the travel to the starting position.

As this little program demonstrates, by controlling the values assigned to the variables that represent the row and column numbers, we can create a degree of motion.

### THE RANDOM MOVING OBJECT PROGRAM

This version of the program gradually fills the screen with squares of constantly changing colors.

---

```

100 REM MOVING OBJECT
110 CALL CLEAR
120 A$="FFFFFFFFFFFFFFFF"
130 CALL CHAR(96,A$)
140 RANDOMIZE :: H=INT(RND*32)+1
150 RANDOMIZE :: V=INT(RND*24)+1
160 RANDOMIZE :: SC=INT(RND*16)+1

```

(continued)

---

```
170 RANDOMIZE :: FG=INT(RND*16)+.
180 CALL SCREEN(SC)
190 CALL COLOR(9,FG,1)
200 CALL HCHAR(V,H,96)
210 FOR PAUSE=1 TO 50 :: NEXT PAUSE
220 GOTO 140
```

---

The Random Moving Object Program is another example of motion. Here the row and column numbers as well as the screen and character colors are created at random, causing the screen to fill gradually with squares of constantly changing colors while the screen color also changes all the time.

**Lines 100-130** are the same as before.

**Lines 140-170** produces the random numbers for columns (H), for rows (V), for the screen color (SC), and for the character foreground color (FG).

**Lines 180 and 190** use subroutines to produce the desired screen and character colors.

**Line 200** displays the character in constantly changing positions and colors.

**Line 210** controls the speed with which characters are displayed.

**Line 220** returns the computer to line 140 to produce a new set of random numbers and start all over again. To stop the program, press FCTN 4 (CLEAR).

Sometimes we want to create images that are larger than a single character. Such images must consist of several characters placed next to one another in order to produce the larger image. For instance, let's assume that we want to create a large ball, say, four times the size of the average character. What we have to do is create four characters, where each represents a quarter of the ball, and then place them next to one another. The Shape Demonstration Program does just that.

---

---

## THE SHAPE DEMONSTRATION PROGRAM

---

This program creates a ball made up of four separate shapes.

---

```
100 CALL CLEAR
110 BALL1$="030F1F3F7F7FFFFF"
120 BALL2$="C0F0F8FCFEFEFFFF"
130 BALL3$="FFFF7F7F3F1FCF03"
140 BALL4$="FFFFFFEFEFCF8FOC0"
150 CALL VCHAR(12,16,96)
160 CALL VCHAR(12,17,97)
170 CALL VCHAR(13,16,98)
180 CALL VCHAR(13,17,99)
190 CALL CHAR(96,BALL1$)
200 CALL CHAR(97,BALL2$)
210 CALL CHAR(98,BALL3$)
220 CALL CHAR(99,BALL4$)
230 GOTO 230
```

---

**Lines 110–140** assign the character shapes for the four quarters of the ball to the four string variables.

**Lines 150–180** determine the positions of the four portions on adjacent column and row numbers.

**Lines 190–220** assign four character codes (96,97,98,99) to the four portions and use the string variables to define the character shapes.

**Line 230** keeps the ball in display until you press FCTN 4 (CLEAR).

With this method you can produce images of any size and shape, though the process tends to be tedious. Still, by combining this capability with varying colors and possibly motion, you can create all manner of interesting imagery.

## CALL SPRITE

Up to now we have concerned ourselves with what is described as *low-resolution* graphics. This type of graphic capability is available with TI BASIC as well as TI EXTENDED BASIC. In addition, if TI EXTENDED BASIC is available, your computer is capable of *high-resolution* graphics. The subprogram used for this purpose

---



is CALL SPRITE, and associated with it are a number of additional subprograms that perform a considerable variety of interesting functions.

A sprite is simply a fancy name for a character shape. A sprite can be any one of the ASCII characters, or it can be any shape that you have created, using the previously discussed CALL CHAR statement. Let's start with a simple program that illustrates creating a sprite representing one of the ASCII characters (SPRITE Demonstration Program #1). As you can see, the program actually consists of only one line (line 120), which creates a sprite representing the ASCII character 38, the ampersand. The numbers in parentheses (#1,38,2,96,128) represent the following: #1 is the sprite number. You can have a great many sprites on the screen at one time, but each one must be given its own number. The second number (38) is the ASCII character code. The third number (2) is the color number, black in this example. The fourth number (96) is referred to as the *dot-row* number. Previously we dealt with only 24 row numbers and 32 column numbers. But when we're using sprites we are dealing with 192 rows, each of which represents one dot width, and 256 columns of single-dot width. Thus, the program places an ampersand on row 96 and column 128, which is more or less screen center. The last line simply keeps it in display until you press FCTN 4 (CLEAR).

---

#### SPRITE DEMONSTRATION PROGRAM #1

---

This program creates a sprite in the shape of an ampersand.

---

```
100 REM SPRITE 1
110 CALL CLEAR
120 CALL SPRITE(#1,38,2,96,128)
130 GOTO 130
```

---

If you want to create your own character shapes, you have to use character code numbers between 32 to 143, but you can redefine the codes that stand for ASCII characters. Let's now write another miniprogram that defines the shape of the sprite to our own specifications (SPRITE Demonstration Program #2.)

---

---

**SPRITE DEMONSTRATION PROGRAM #2**

---

Here we create a sprite in the shape of a little square man.

---

```
100 REM SPRITE 2
110 CALL CLEAR
120 H$="1818FFBD3C3C2424"
130 CALL CHAR(96,H$)
140 CALL SPRITE(#1,96,7,96,124)
150 GOTO 150
```

---

**Line 120** creates the shape of the little man that we used in one of the earlier programs, assigning the code to the string variable H\$.

**Line 130** assigns that shape to a character number (96).

**Line 140** produces sprite #1, shape 96, color 7, at dot row 96 and dot column 124.

**Line 150** keeps the sprite in display.

## **CALL MAGNIFY**

Now that we have figured out how to create a sprite, let's go one step further. So far we have dealt with characters that are the size of an ordinary capital letter or, in cases where we designed our own characters, possibly even smaller. When we're dealing with sprites, there is another subprogram available to us to increase the size of those characters by doubling, tripling, or quadrupling their dimensions. The statement used for that purpose is **CALL MAGNIFY (number)**, where the number can be 1, 2, 3, or 4. Figure 11-14 **SPRITE Demonstration Program #3** represents an addition to our short program that illustrates how that works.

---

**SPRITE DEMONSTRATION PROGRAM #3**

---

Now we use **MAGNIFY** to double the size of the sprite.

---

```
100 REM SPRITE 3
110 CALL CLEAR
120 H$="1818FFBD3C3C2424"
130 CALL CHAR(96,H$)
140 CALL SPRITE(#1,96,7,96,124)
150 FOR PAUSE=1 TO 50 :: NEXT PAUSE
160 CALL MAGNIFY(2)
170 GOTO 170
```

---

**Lines 100–140** are the same as before.

**Line 150** creates a short pause while the sprite is displayed at its original size.

**Line 160** uses the magnification statement to cause the sprite to double in size.

**Line 170** holds it in display.

## CALL MOTION

Before, when we wanted to move our character on the screen, we had to achieve motion by specifying ascending or descending row and column numbers, creating some rather jerky movement. With sprites we have a special subprogram at our disposal, **CALL MOTION**, which creates very smooth movement in any desired direction. The statement is used in conjunction with three values in parentheses:

100 CALL MOTION (#1,A,B)

where #1 is the identification of the sprite that is to be moved. A and B are numeric variables, representing the speed at which the sprite is to move in a given direction. The first of the two (A) is the *row velocity*, the speed at which the sprite moves either up or down. The available values for these variables are from  $-128$  to  $+127$ . Negative numbers cause the sprite to move up, whereas positive values move it down. Zero produces no movement. The greater the number on either side of zero, the greater the speed of movement. The second of the two variables (B) controls *column velocity*, the speed at which the sprite moves from one side to the other. The available numbers are the same; and negative numbers move it to the left, whereas positive numbers move it to the right.

**SPRITE Demonstration Program #4** is the same program we discussed before with some added lines that cause the sprite to jump across the screen in huge leaps. After a while the leaps get shorter and shorter. The sprite then reverses direction and continues with increasing vigor.

---

### SPRITE DEMONSTRATION PROGRAM #4

Here we use MOTION to make our sprite move.

```

100 REM SPRITE4
110 CALL CLEAR
120 H$="1818FFBD3C3C2424"
130 CALL CHAR(96,H$)
140 CALL SPRITE(#1,96,7,96,124)
150 FOR PAUSE=1 TO 50 :: NEXT PAUSE
160 CALL MAGNIFY(2)
170 FOR X=-50 TO 50
180 FOR Y=-20 TO 20
190 CALL MOTION(#1,Y,X)
200 NEXT Y
210 NEXT X
220 GOTO 140

```

**Lines 100–160** are unchanged from the previous program.

**Lines 170 and 180** set up two loops that assign ascending numbers to the numeric variables X and Y, where X stands for horizontal motion and Y for vertical motion.

**Line 190** tells sprite #1 to move in accordance with the values of the two variables.

**Lines 200 and 210** close the loops.

**Line 220** returns the computer to line 140 for another run through the program.

As you must have gathered by now, a combination of row and column velocities produces movement at any desired angle and at any speed you like. To simplify experimentation, SPRITE Demonstration Program #5 lists a program you can use to practice the CALL MOTION statement.

### SPRITE DEMONSTRATION PROGRAM #5

This program can be used to practice the use of MOTION.

```

100 REM SPRITE 5
110 CALL CLEAR
120 INPUT "ROW VELOCITY? ":Y
130 INPUT "COL VELOCITY? ":X
140 CALL CLEAR

```

(continued)

```
150 H$="1818FFBD3C3C2424"
160 CALL CHAR(96,H$)
170 CALL SPRITE(#1,96,7,96,124)
180 FOR PAUSE=1 TO 50 :: NEXT PAUSE
190 CALL MAGNIFY(2)
200 CALL MOTION(#1,Y,X)
210 GOTO 120
```

---

**Lines 120 and 130** ask you to key in the two velocity values for row and column velocities.

**Lines 140-190** are the same as in the previous program.

**Line 200** causes the desired motion.

**Line 210** returns you to line 120 to permit you to enter different velocity values. While the two INPUT lines appear on the screen, the sprite continues to move in the established direction until you key in new values and press >ENTER<, at which point the sprite changes to the new direction and speed.

If you're designing some kind of motion program in which the direction and speed of one or several sprites is important, you can use this program to determine the criteria that will produce the effect you want.

## **CALL DELSPRITE**

A sprite remains on the screen, either still or in motion, until the program is terminated, or until we use yet another subprogram, CALL DELSPRITE, which erases the sprite from the screen. It is used in conjunction with the sprite number(s) in parentheses as shown in SPRITE Demonstration Program #6. This program works like the previous one except that the sprite disappears intermittently for about a half a second as a result of lines 220-240.

### **SPRITE DEMONSTRATION PROGRAM #6**

DELSPRITE deletes the sprite from the display.

---

```
100 REM SPRITE 6
110 CALL CLEAR
120 H$="1818FFBD3C3C2424"
130 CALL CHAR(96,H$)
```

(continued)

---

```
140 CALL SPRITE(#1,96,7,96,124)
150 FOR PAUSE=1 TO 50 :: NEXT PAUSE
160 CALL MAGNIFY(2)
170 FOR X=-6 TO 6
180 FOR Y=-10 TO 10
190 CALL MOTION(#1,Y,X)
200 NEXT Y
210 NEXT X
220 CALL DELSPRITE(#1)
230 FOR PAUSE=1 TO 300
240 NEXT PAUSE
250 GOTO 140
```

---

## CALL PATTERN

You can change the shape of a sprite by using the subprogram CALL PATTERN in conjunction with the number of the sprite to be changed and a new character code number.

The SPRITE Demonstration Program #7 illustrates. Line 170 changes the shape of the sprite from our little man to an ampersand.

### SPRITE DEMONSTRATION PROGRAM #7

PATTERN changes the shape of our sprite.

---

```
100 REM SPRITE 7
110 CALL CLEAR
120 H$="1818FFBD3C3C2424"
130 CALL CHAR(96,H$)
140 CALL SPRITE(#1,96,7,96,124)
150 CALL MAGNIFY(2)
160 FOR PAUSE=1 TO 500 :: NEXT PAUSE
170 CALL PATTERN(#1,38)
180 FOR PAUSE=1 TO 500 :: NEXT PAUSE
190 GOTO 120
```

---

When you run the program, the shape of the sprite changes at about 2-second intervals until you stop the program.

---

## CALL LOCATE

If you want to move a sprite to another location, use the CALL LOCATE subprogram in conjunction with the sprite number and the dot row and dot column numbers to which you want to move it, as illustrated in SPRITE Demonstration Program #8. Here the only line that was changed from the previous program is line 170, which moves the sprite to a new location. When you run the program, the sprite jumps back and forth at roughly 2-second intervals.

---

### SPRITE DEMONSTRATION PROGRAM #8

---

LOCATE relocates our sprite.

---

```
100 REM SPRITE 8
110 CALL CLEAR
120 H$="1818FFBD3C3C2424"
130 CALL CHAR(96,H$)
140 CALL SPRITE(#1,96,7,96,124)
150 CALL MAGNIFY(2)
160 FOR PAUSE=1 TO 500 :: NEXT PAUSE
170 CALL LOCATE(#1,50,50)
180 FOR PAUSE=1 TO 500 :: NEXT PAUSE
190 GOTO 120
```

---

## CALL COLOR and SPRITES

When dealing with sprites, you can change the color of any of the sprites by using the CALL COLOR statement in conjunction with the sprite identification number and just one number for the desired change in color (sprites have no background color), as demonstrated in SPRITE Demonstration Program #9. Here line 170 has been changed to increase the value of the numeric variable C progressively to 16 and then return it to 1, and in line 180 the color of the sprite is changed in accordance with the value assigned to that numeric variable.

---

**SPRITE DEMONSTRATION PROGRAM #9**

COLOR changes the color of our sprite.

---

```
100 REM SPRITE 9
110 CALL CLEAR
120 H$="1818FFBD3C3C2424"
130 CALL CHAR(96,H$)
140 CALL SPRITE(#1,96,7,96,124)
150 CALL MAGNIFY(2)
160 FOR PAUSE=1 TO 500 :: NEXT PAUSE
170 C=C+1 :: IF C>16 THEN C=1
180 CALL COLOR(#1,C)
190 FOR PAUSE=1 TO 500 :: NEXT PAUSE
200 GOTO 120
```

---

## **CALL DISTANCE**

You can determine the distance of a sprite from another sprite, or from a given row/column position, by using the CALL DISTANCE subprogram. In the first case you use the statement in conjunction with two sprite numbers and a numeric variable in parentheses. In the second case you use it with one sprite number, a dot row and dot column position coordinate, and a numeric variable, also in parentheses. The distance is then assigned to the numeric variable. The figure that results is actually the square of the true distance. Thus, before causing it to be displayed, the square root should be calculated, using the SQR statement. SPRITE Demonstration Program #10 demonstrates this function.

**Line 150** creates a second sprite and places it in a different position (dot row 50, dot column 50).

**Line 170** uses the statement that produces the distance squared.

**Line 180** takes the square root of that figure and then causes the result to be displayed. In this case the resulting figure is 87.13208364, representing 87 dot width. Since the decimals are of no practical use, we might want to add a line that reduces the figure to its integer.

---



**SPRITE DEMONSTRATION PROGRAM #10**

This program displays the distance between two sprites.

---

```
100 REM SPRITE 10
110 CALL CLEAR
120 H$="1818FFBD3C3C2424"
130 CALL CHAR(96,H$)
140 CALL SPRITE(#1,96,7,96,124)
150 CALL SPRITE(#2,96,7,50,50)
160 CALL MAGNIFY(2)
170 CALL DISTANCE(#1,#2,D)
180 D=SQR(D):: PRINT D
190 GOTO 190
```

---

If you want a certain action to take place when a sprite has reached a given distance position, you can then use:

100 IF D=X THEN action or line number (ELSE. . .)

instead of using the PRINT statement to have the number printed.

## **CALL COINC**

When we design programs that use several sprites in motion, we may want to achieve a certain action if and when they collide. The subprogram for that purpose is CALL COINC (for coincide or coincident). The program detects the collision of two sprites or the arrival of a sprite at a given dot row/dot column position. The statement is used in conjunction with any of three subscripts in parentheses:

```
(sprite #, sprite #, tolerance, numeric variable)
(sprite #, dot-row, dot column, tolerance, numeric
variable)
(ALL, numeric variable)
```

---

where the first determines the coincidence between two sprites. The tolerance value must be given and refers to the distance between the (nearly) colliding sprites in terms of dot width. The second performs the identical function with reference to a given sprite and a dot/column position on the screen. The third is used when you want the statement to apply to all sprites. In that case no tolerance figure is used. SPRITE Demonstration Program #11 demonstrates this function.

#### SPRITE DEMONSTRATION PROGRAM #11

COINC detects an impending collision.

```
100 REM SPRITE 11
110 CALL CLEAR
120 H$="1818FFBD3C3C2424"
130 CALL CHAR(96,H$)
140 CALL SPRITE(#1,96,7,96,124)
150 CALL SPRITE(#2,96,7,50,75)
160 CALL MAGNIFY(2)
170 CALL MOTION(#1,-1,-1)
180 CALL COINC(#1,#2,20,A)
190 PRINT A
200 CALL COINC(ALL,B)
210 PRINT B
220 GOTO 180
```

**Line 170** commands sprite #1 to move slowly toward sprite #2.

**Line 180** calls for a coincidence to be detected when the upper left-hand corner of sprite #1 is within 20 dot widths of the upper left-hand corner of sprite #2, assigning zero to the numeric variable A when they are far enough apart, and assigning -1 to that variable when the two are within the tolerance.

**Line 190** continuously prints the value of A (0 or -1).

**Line 200** uses the statement in the third variation, requiring no tolerance, assigning the results to the numeric variable B.

**Line 210** prints the results.

**Line 220** sends the computer back to line 180 to make sure that the CALL COINC statement is executed for each move of the sprite.

As before, if you want a certain action to take place when two sprites collide or are about to collide, such as have one disappear or change color or direction, then, instead of having the values produced by the numeric variables printed, you can use the IF. THEN. . ELSE statement to trigger such action.

Playing with sprites can be a very enjoyable pastime. You can devise your own computer games, combining the graphics functions with music and voice. In Chapter 12 I have included an example of the use of such multiple functions.

---

---

# 12

---

## ***Programs Strictly for Fun***

---

---

In this chapter we'll write a number of programs designed with no more serious purpose than to offer a bit of fun or to be used in the pursuit of a hobby. Some of the game programs can be played by more than one person, and others are intended as solo entertainment.

### ***A DICE GAME***

We'll start with a dice game. I have included two versions of the game, one requiring the voice synthesizer, the other a silent version.

Before we start, here are the rules of the game: Two dice are used. On the first throw, if you throw a 2 or a 12, you lose your bet but retain the dice. If you throw a 7 or 11, you win. If you throw any other number, that number becomes the "point" that you'll try to match during subsequent throws of the dice. If you make your point, you win. If you throw a 7, you lose your bet and the dice go to the next player.

Let's look first at the program called The Talking Dice Game. When it is activated it displays its title, and after a brief pause the voice says:

"To start, press enter"

---

## TALKING DICE GAME

This program uses the speech synthesizer to simulate a dice game.

```
100 GOTO 130
110 PRINT "-----" :: RETURN
120 FOR X=1 TO 10 :: PRINT :: NEXT X :: RETURN
130 REM A DICE GAME WITH VOICE
140 REM TI EXTENDED BASIC
150 CALL CLEAR :: GOSUB 110
160 PRINT "This program simulates a" :: PRINT
170 PRINT "dice game played with" :: PRINT
180 PRINT "two dice." :: GOSUB 110 :: GOSUB 120
190 CALL SAY("TO START PRESS ENTER")
200 INPUT ENTER$
210 RANDOMIZE :: PLAY=INT(RND*12)+1 :: IF PLAY=1 THEN 210
220 IF PLAY=2 THEN PLAY$="TWO"
230 IF PLAY=3 THEN PLAY$="THREE"
240 IF PLAY=4 THEN PLAY$="FOUR"
250 IF PLAY=5 THEN PLAY$="FIVE"
260 IF PLAY=6 THEN PLAY$="SIX"
270 IF PLAY=7 THEN PLAY$="SEVEN"
280 IF PLAY=8 THEN PLAY$="EIGHT"
290 IF PLAY=9 THEN PLAY$="NINE"
300 IF PLAY=10 THEN PLAY$="TEN"
310 IF PLAY=11 THEN PLAY$="ELEVEN"
320 IF PLAY=12 THEN PLAY$="TWELVE"
330 IF XX$="ROUND" THEN 520
340 POINT=PLAY :: POINT$=PLAY$
350 CALL SAY("YOU MADE"):: CALL SAY(PLAY$)
360 IF XX$="ROUND" THEN 390
370 IF PLAY=2 OR PLAY=12 THEN 400
380 IF PLAY=7 OR PLAY=11 THEN 430
390 IF PLAY>2 AND PLAY<7 OR PLAY>7 AND PLAY<11 THEN 460
400 CALL SAY("NO GOOD,,TRY AGAIN,,PRESS ENTER")
410 GOSUB 640
420 INPUT ENTER$:: IF ENTER$="Q" THEN 630 ELSE 210
430 CALL SAY("YOU WON,,TO GO ON PRESS ENTER")
440 GOSUB 640
450 INPUT ENTER$:: IF ENTER$="Q" THEN 630 ELSE 210
460 CALL SAY("THE POINT IS"):: CALL SAY(PLAY$)
470 CALL SAY("PRESS ENTER")
480 PRINT "The point is ";PLAY :: GOSUB 110 :: GOSUB 120
490 XX$="ROUND"
500 GOSUB 640
510 INPUT ENTER$:: IF ENTER$="Q" THEN 630 ELSE 210
520 CALL CLEAR
530 CALL SAY("THE POINT IS"):: CALL SAY(POINT$)
540 PRINT "The point is ";POINT :: GOSUB 110 :: GOSUB 120
550 CALL SAY("YOU MADE"):: CALL SAY(PLAY$)
560 IF PLAY=POINT THEN 620
```

(continued)

```

570 IF PLAY<>7 THEN 580 ELSE 610
580 CALL SAY("TO GO ON,,PRESS ENTER")
590 GOSUB 640
600 INPUT ENTER$:: IF ENTER$="Q" THEN 630 ELSE 210
610 XX$="START" :: GOSUB 110 :: PRINT TAB(5);"New shooter!"
 " :: GOSUB 110 :: GOSUB 120 :: GOTO 400
620 XX$="START" :: GOTO 430
630 CALL CLEAR :: GOSUB 110 :: PRINT TAB(12);"End." ::
 GOSUB 110 :: GOSUB 120 :: END
640 PRINT "To quit press Q" :: PRINT :: PRINT
650 RETURN

```

---

Pressing >ENTER< produces a random number between 2 and 12, which, at first, is not displayed. Instead the voice responds with one of three announcements, depending on that number:

"You made (number), no good, try again, press enter"

or

"You made (number), no good, try again, press enter"

or

"You made (number), you won. To go on, press enter"

In the last instance the display shows:

The point is (number)

---

This display will be repeated until you make your point or throw a 7. Also displayed, whenever the voice prompts you to press >ENTER<, is:

A screenshot of a TI-99/4A computer screen. The screen is rectangular with rounded corners and a black border. The text "To quit press Q" is displayed in a monospaced font, centered horizontally and positioned in the upper half of the screen.

To quit press Q

in order to give you the option to quit the game at any time. If you lose your bet because you've thrown a 7 before making your point, the display reminds you with:

A screenshot of a TI-99/4A computer screen. The screen is rectangular with rounded corners and a black border. The text "New shooter!" is displayed in a monospaced font, centered horizontally and positioned in the upper half of the screen.

New shooter!

---

in addition to the audio announcement of your loss. It goes without saying that you can play this game alone, or with others, but it's more fun when several persons are involved and when bets are placed.

Line by line:

**Line 100** tells the computer to skip over two lines that contain two subroutines.

**Line 110** is the subroutine that places a dashed line into display.

**Line 120** is the subroutine that centers copy on the screen.

**Lines 130 and 140** are REMarks.

**Lines 150–180** place the title of the program into display.

**Line 190** tells the voice synthesizer what to say.

**Line 200** places a question mark into the lower left-hand corner of the screen and causes the speaker in the monitor to beep to remind you to press >ENTER<.

**Line 210** produces a random number and assigns it to the numeric variable PLAY. It then checks that number and causes the computer to try again if the number is 1.

**Lines 220–320** assign strings representing 11 numbers to the string variable PLAY\$, depending on the random number.

**Line 330** examines the string assigned to the string variable XX\$ to determine whether or not this is the first throw. If it is not the first throw, the computer is told to go to line 520.

**Line 340** assigns the value of PLAY to the numeric variable POINT and the string assigned to PLAY\$ to the string variable POINT\$.

**Line 350** tells the voice synthesizer what to say.

**Line 360** once more examines the string assigned to XX\$, sending the computer to line 390 if this is not the first throw.

**Lines 370 and 380** are used if this is the first throw, examining the value of PLAY and sending the computer to one of two line numbers if you threw a 2, 12, 7, or 11, resulting in an immediate win or loss. If any other number came up, the computer goes to the next line.

**Line 390** tells the computer to go to line 460 if a number larger than 2 and smaller than 7 or larger than 7 and smaller than 11 was thrown.

**Line 400** tells the computer what to say if you have lost.

**Line 410** uses the subroutine (lines 640 and 650) to display the choice that permits you to quit.

---



**Line 420** checks whether you pressed >ENTER< or Q and tells the computer where to go next.

**Line 430** tells the computer what to say if you've won.

**Lines 440 and 450** are the same as lines 410 and 420.

**Lines 460 and 470** tell the computer what to say if a point has been made.

**Line 480** causes the point to be displayed.

**Line 490** assigns the string "ROUND" to the string variable XX\$ to indicate that subsequent throws of the dice are not the first throw.

**Lines 500 and 510** are the same as lines 410 and 420.

**Line 520** clears the screen.

**Lines 530 and 540** tell the computer what to say and again display the point that was made with the first throw.

**Line 550** cause the computer to announce what your throw was.

**Line 560** compares the point with your throw and sends the computer to line 620 if the two match.

**Line 570** checks if your throw was a 7, sending the computer to one of two line numbers.

**Line 580** once more tells the computer what to say.

**Lines 590 and 600** are the same as lines 410 and 420.

**Line 610** assigns the string "START" to the string variable XX\$ to indicate that the next throw will be the first throw of a new player, and it displays the reminder that the dice are to go to a new shooter.

**Line 620** also assigns "START" to XX\$ and then sends the computer to line 430 to tell you that you won.

**Line 630** is the END line, which is used if you press Q instead of >ENTER<.

**Lines 640 and 650** are the subroutine that permits you to quit the game.

This version of the dice game was written in TI EXTENDED BASIC. The Display Dice Game eliminates the voice feature and is written in TI BASIC. Otherwise it performs in exactly the same manner, displaying the announcements on the screen.

---

# DISPLAY DICE GAME PROGRAM

The dice game program without the speech synthesizer.

---

```

100 REM A DICE GAME/TI99/4A
110 REM TI BASIC (NO VOICE)
120 CALL CLEAR
130 GOSUB 830
140 PRINT "This program simulates a": :
150 PRINT "dice game played with": :
160 PRINT "two dice.": :
170 GOSUB 830
180 GOSUB 850
190 GOSUB 730
200 POINT$="The point is "
210 LOST$="You lost. New shooter!"
220 MADE$="You made "
230 CRAPS$="Craps, you lost."
240 WON$="You won!"
250 CALL CLEAR
260 XX=XX+1
270 RANDOMIZE
280 PLAY=INT(RND*12)+1
290 IF PLAY=1 THEN 270
300 IF XX=1 THEN 310 ELSE 420
310 IF PLAY=7 THEN 530
320 IF PLAY=11 THEN 530
330 IF PLAY=2 THEN 660
340 IF PLAY=12 THEN 660
350 GOSUB 830
360 PRINT POINT$;PLAY
370 GOSUB 830
380 GOSUB 850
390 POINT=PLAY
400 GOSUB 730
410 GOTO 250
420 GOSUB 830
430 PRINT POINT$;POINT
440 PRINT
450 PRINT MADE$;PLAY
460 GOSUB 830
470 GOSUB 850
480 IF POINT=PLAY THEN 530
490 IF PLAY=7 THEN 600
500 IF XX=1 THEN 580 ELSE 510
510 GOSUB 730
520 GOTO 250
530 GOSUB 830

```

(continued)

---

```
540 PRINT PLAY;" ";WON$
550 XX=0
560 GOSUB 830
570 GOSUB 850
580 GOSUB 730
590 GOTO 250
600 GOSUB 830
610 PRINT LOST$: :
620 XX=0
630 GOSUB 830
640 GOSUB 730
650 GOTO 250
660 GOSUB 830
670 PRINT PLAY;" ";CRAPS$
680 XX=0
690 GOSUB 830
700 GOSUB 850
710 GOSUB 730
720 GOTO 250
730 PRINT
740 INPUT "Press >ENTER< or Q to quit ":E$
750 IF E$="Q" THEN 770 ELSE 760
760 RETURN
770 CALL CLEAR
780 GOSUB 830
790 PRINT TAB(12);"End."
800 GOSUB 830
810 GOSUB 850
820 END
830 PRINT "-----"
840 RETURN
850 FOR X=1 TO 10
860 PRINT
870 NEXT X
880 RETURN
```

---

Line by line:

**Lines 100 and 110** are REMarks.

**Line 120** clears the screen.

**Lines 130–190** place the title of the program into display, using a number of subroutines to make the display look reasonably attractive.

**Lines 200–240** assign a number of strings to string variables.

**Lines 250–280** clear the screen, cause the value of XX to be raised by 1 each time the line is encountered, and then produce a random number between 1 and 12.

---

**Line 290** tells the computer to try again if the random number is 1.

**Line 300** checks the value assigned to XX to determine if this is the first throw.

**Lines 310–340** determine whether you threw one of the winning or losing numbers on the first throw.

**Line 360** prints the string assigned to POINT\$ and the value assigned to PLAY.

**Line 390** assigns the value of PLAY to the variable POINT.

**Lines 430 and 450** display the strings assigned to POINT\$ and MADE and the values of POINT and PLAY.

**Lines 480 and 490** check if you made your point or threw a 7, sending the computer to one of two line numbers if the answer to either question is affirmative.

**Line 500** checks whether or not this is the first throw.

**Lines 540 and 550** are used if you won, reassigning a value of zero to XX.

**Lines 610 and 620** are used if you lost.

**Lines 670 and 680** are used if you threw a 2 or a 12 on the first throw.

**Lines 740–760** represent the frequently used subroutine telling you to press >ENTER< or Q.

**Lines 770–820** are the END lines.

**Lines 830 and 840** are the dashed-line subroutine.

**Lines 850–880** are the subroutine that moves copy to the center of the screen.

## **MODEL RAILROADING PROGRAM**

The next program is of an entirely different nature. It is designed for those readers whose hobby is model railroading. Model railroading, especially for those hobbyists who take pride in faithfully reproducing real scenes to a specific scale, involves a great number of calculations, some of which are rather difficult and time-consuming to perform with pencil and paper. This Model Railroad Program consists of a number of subprograms that take care of these calculations, and one that transforms your computer into a clock that keeps scale time. It can be used with any of the seven popular gauges.

---

**MODEL RAILROAD PROGRAM**

This program is designed to assist model railroaders in their hobby.

---

```
100 REM MODEL RAILROADING
110 REM WRITTEN IN TI BASIC
120 GOSUB 2800
130 GOSUB 2780
140 PRINT "Programs for model railroads"
150 GOSUB 2780
160 GOSUB 2820
170 GOSUB 2860
180 GOSUB 2800
190 PRINT "This is a collection of"
200 PRINT "programs that is designed"
210 PRINT "to be used by model rail-"
220 PRINT "roaders using any of the"
230 PRINT "seven popular gauges."
240 GOSUB 2780
250 GOSUB 2820
260 GOSUB 2860
270 GOSUB 2800
280 PRINT "Select your gauge:"
290 GOSUB 2780
300 PRINT 1,"O"
310 PRINT 2,"S"
320 PRINT 3,"OO"
330 PRINT 4,"HO"
340 PRINT 5,"TT"
350 PRINT 6,"N"
360 PRINT 7,"Z"
370 GOSUB 2780
380 INPUT "Which? ":WHICH
390 ON WHICH GOSUB 410,430,450,470,490,510,530
400 GOTO 550
410 G=48
420 RETURN
430 G=64
440 RETURN
450 G=76.2
460 RETURN
470 G=87.1
480 RETURN
490 G=120
500 RETURN
510 G=160
520 RETURN
530 G=220
540 RETURN
550 GOSUB 2800
```

(continued)

---

```

560 PRINT "Menu:"
570 GOSUB 2780
580 PRINT 1;" Convert linear measures"
590 PRINT 2;" Convert MPH to FPM"
600 PRINT 3;" Convert to scale time"
610 PRINT 4;" Scale-time clock"
620 PRINT 5;" Prototype speed=distance"
630 PRINT 6;" Distance=prototype speed"
640 PRINT 7;" Determine grade data"
650 PRINT 8;" Determine curve data"
660 PRINT 9;" Ohm's Law"
670 PRINT 10;" Check wire resistance"
680 GOSUB 2780
690 PRINT 11;" Exit the program"
700 GOSUB 2780
710 INPUT "Select one ":SELECT
720 GOSUB 2800
730 ON SELECT GOTO 740,960,1020,1330,1790,1910,200,2340,
 2450,2720,2890
740 PRINT "Do you want to convert:"
750 GOSUB 2780
760 PRINT 1;" Prototype to scale?"
770 PRINT " or"
780 PRINT 2;" Scale to prototype?"
790 GOSUB 2780
800 INPUT "Type 1 or 2 ":TYPE
810 GOSUB 2800
820 ON TYPE GOTO 830,910
830 INPUT "Prototype measure? ":FEET
840 FEET=FEET/G
850 FEET=INT(FEET*10+.5)/10
860 GOSUB 2780
870 PRINT "Scale measure= ";FEET
880 GOSUB 2780
890 GOSUB 2860
900 GOTO 550
910 INPUT "Scale measure? ":SCALE
920 SCALE=SCALE*G
930 GOSUB 2780
940 PRINT "Prototype measure= ":SCALE
950 GOTO 880
960 INPUT "Prototype MPH? ":MPH
970 GOSUB 2780
980 FPM=MPH*5280/G/60
990 FPM=INT(FPM*10+.5)/10
1000 PRINT "Feet per minute= ";FPM
1010 GOTO 880
1020 PRINT "Scale time by:"
1030 GOSUB 2780
1040 PRINT 1;" Scale or"
1050 PRINT 2;" Percentage?"

```

(continued)

---

```
1060 GOSUB 2780
1070 INPUT "Type 1 or 2 ":TYPE
1080 GOSUB 2800
1090 ON TYPE GOTO 1100,1220
1100 INPUT "Hours.minutes to convert? ":TIME
1110 TIME1=INT(TIME)
1120 TIME2=TIME-TIME1
1130 TIME3=TIME2/.6
1140 TIME=TIME1+TIME3
1150 TIME=TIME/G
1160 HOURS=INT(TIME)
1170 MINS=(TIME-HOURS)*.6*100
1180 GOSUB 2780
1190 MINS=INT(MINS*10+.5)/10
1200 PRINT "Scale time= ";HOURS;" hours and ";MINS;"
 minutes"
1210 GOTO 880
1220 INPUT "Hours.minutes to convert? ":TIME
1230 INPUT "Conversion percentage? ":PER
1240 TIME1=INT(TIME)
1250 TIME2=TIME-TIME1
1260 TIME3=TIME2/.6
1270 TIME=TIME1+TIME3
1280 TIME=TIME*(PER/100)
1290 HOURS=INT(TIME)
1300 MINS=(TIME-HOURS)*.6*100
1310 GOSUB 2780
1320 GOTO 1200
1330 PRINT "The computer = a scale clock"
1340 GOSUB 2780
1350 INPUT "Timing speed (actual=100) ":SPEED
1360 INPUT "Timing limit (hrs.mins) ":LIMIT
1370 LIMIT1=INT(LIMIT)
1380 LIMIT2=LIMIT-LIMIT1
1390 GOSUB 2780
1400 INPUT "To start, press >ENTER< ":RET$
1410 GOSUB 2800
1420 M$="Minute"
1430 MM$="Minutes"
1440 S$="Second"
1450 SS$="Seconds"
1460 H$="Hour"
1470 HH$="Hours"
1480 FOR FAUSE=1 TO SPEED
1490 NEXT FAUSE
1500 FOR XX=1 TO 5
1510 PRINT
1520 NEXT XX
1530 A=A+1
1540 IF A=60 THEN 1690
1550 IF A=1 THEN 1560 ELSE 1580
```

(continued)

---

```

1560 SSS$=S$
1570 GOTO 1590
1580 SSS$=SS$
1590 IF B=1 THEN 1600 ELSE 1620
1600 MMM$=M$
1610 GOTO 1630
1620 MMM$=MM$
1630 IF C=1 THEN 1640 ELSE 1660
1640 HHH$=H$
1650 GOTO 1670
1660 HHH$=HH$
1670 PRINT C;HHH$;" ";B;MMM$;" ";A;SS$
1680 GOTO 1480
1690 A=0
1700 B=B+1
1710 IF C+(B/100)=LIMIT1+LIMIT2 THEN 1760
1720 IF B=60 THEN 1730 ELSE 1480
1730 B=0
1740 C=C+1
1750 GOTO 1480
1760 PRINT "To continue time,type CON"
1770 BREAK
1780 GOTO 1480
1790 INPUT "Distance covered (feet) " :FEET
1800 INPUT "Time (mins.secs) " :TIME
1810 TIME1=INT(TIME)
1820 TIME2=(TIME-TIME1)/.6
1830 TIME=TIME1+TIME2
1840 FEET=FEET*G/5280
1850 TIME=TIME/60
1860 SPEED=FEET/TIME
1870 SPEED=INT(SPEED*10+.5)/10
1880 GOSUB 2780
1890 PRINT "Prototype speed= ";SPEED;" mph."
1900 GOTO 880
1910 INPUT "Prototype speed? (mph)9":SPEED
1920 INPUT "Time? (mins.secs) ":TIME
1930 TIME1=INT(TIME)
1940 TIME2=(TIME-TIME1)/.6
1950 TIME=TIME1+TIME2
1960 FEET=SPEED*5280/60*TIME/G
1970 FEET=INT(FEET*10+.5)/10
1980 GOSUB 2780
1990 PRINT "Distance covered= ";FEET;" feet"
2000 GOTO 880
2010 PRINT "Do you want to find:"
2020 GOSUB 2780
2030 PRINT 1;" Distance by grade percent"
2040 PRINT " and elevation change,"
2050 PRINT 2;" Grade percent by distance"
2060 PRINT " and elevation change or"

```

(continued)

---



```
2070 PRINT 3;" Elevation change by"
2080 PRINT " percent and distance"
2090 GOSUB 2780
2100 INPUT "Type 1,2 or 3 ":GRADE
2110 GOSUB 2800
2120 ON GRADE GOTO 2130,2200,2270
2130 INPUT "Percent grade desired? ":PER
2140 INPUT "Inches to climb/descend? ":ELEV
2150 GOSUB 2780
2160 DIST=PER/ELEV*100
2170 DIST=INT(DIST*10+.5)/10
2180 PRINT "Distance needed= ";DIST;" inches"
2190 GOTO 880
2200 INPUT "Distance available? (inches) ":DIST
2210 INPUT "Inches to climb/drescend? ":INCH
2220 GOSUB 2780
2230 PER=INCH/DIST*100
2240 PER=INT(PER*10+.5)/10
2250 PRINT "Percent grade= ";PER;"%"
2260 GOTO 880
2270 INPUT "Percent grade desired? ":PER
2280 INPUT "Distance available (inches) ":INCH
2290 GOSUB 2780
2300 ELEV=PER/100*DIST
2310 ELEV=INT(ELEV*10+.5)/10
2320 PRINT "Change in elevation= ";ELEV;" inches."
2330 GOTO 880
2340 INPUT "Curve radius? ":RAD
2350 INPUT "Curve what % of circle? ":CIR
2360 GOSUB 2780
2370 CIR=CIR/100
2380 INCH=RAD*(2*3.14159)*CIR
2390 INCH=INT(INCH*10+.5)/10
2400 PRINT "Length of curve= ";INCH;" inches."
2410 MILES=INCH/12/5280*6
2420 MILES=INT(MILES*10+.5)/10
2430 PRINT "Length in prototype miles= ";MILES;"
 miles."
2440 GOTO 880
2450 PRINT "Ohm's Law:"
2460 GOSUB 2780
2470 PRINT 1;" Find ohms"
2480 PRINT 2;" Find amperes"
2490 PRINT 3;" Find volts"
2500 GOSUB 2780
```

(continued)

```

2510 INPUT "Type 1,2 or 3 ":TYPE
2520 GOSUB 2800
2530 ON TYPE GOTO 2540,2600,2660
2540 INPUT "Number of volts? ":VOLT
2550 INPUT "Number of amperes? ":AMP
2560 GOSUB 2780
2570 OHM=VOLT/AMP
2580 PRINT "Ohms= ";OHM;" ohms"
2590 GOTO 880
2600 INPUT "Number of volts? ":VOLT
2610 INPUT "Number of ohms? ":OHM
2620 GOSUB 2780
2630 AMP=VOLT/OHM
2640 PRINT "Amperes= ";AMP;" amperes."
2650 GOTO 880
2660 INPUT "Number of amperes? ":AMP
2670 INPUT "Number of ohms? ":OHM
2680 GOSUB 2780
2690 VOLT=AMP*OHM
2700 PRINT "Volts= ";VOLT;" volts."
2710 GOTO 880
2720 INPUT "Length of wire? (inches) ":WIRE
2730 INPUT "Diameter of wire? (mils) ":MILS
2740 GOSUB 2780
2750 OHM=10.4*WIRE/MILS
2760 PRINT "Wire resistance= ";OHM;" ohms"
2770 GOTO 880
2780 PRINT "-----"
2790 RETURN
2800 CALL CLEAR
2810 RETURN
2820 FOR X=1 TO 10
2830 PRINT
2840 NEXT X
2850 RETURN
2860 PRINT
2870 INPUT "Press >ENTER< ":Y$
2880 RETURN
2890 GOSUB 2800
2900 GOSUB 2780
2910 PRINT TAB(13);"End."
2920 GOSUB 2780
2930 GOSUB 2820
2940 END

```

---

When it is executed, it first displays:

Programs for model railroads

-----  
This is a collection of  
programs that is designed  
to be used by model rail-  
roaders using any of the  
seven popular gauges.  
-----

Select your gauge:

-----  
1            O  
2            S  
3            OO  
4            HO  
5            TT  
6            N  
7            Z  
-----

Which?

which is followed by these choices:

1 Convert linear measures  
2 Convert MPH to FPM  
3 Convert to scale time  
4 Scale-time clock  
5 Prototype speed=distance  
6 Distance=prototype speed  
7 Determine grade data  
8 Determine curve data  
9 Ohm's Law  
10 Check wire resistance  
-----

11 Exit program  
-----

Select one

---

with some of these subprograms consisting of two or more choices, such as converting to or from a certain measure. If the program seems excessively long, it is because of this multitude of choices and the fact that it is written in TI BASIC, which permits only one statement per line.

The first subprogram permits conversion from prototype to scale or scale to prototype, using any type of measure (feet, inches, meters, etc.).

The second subprogram converts miles per hour to feet per minute and vice versa.

The third subprogram converts hours and minutes to scale hours and minutes, using either the selected scale or a percentage figure selected by the user.

The fourth subprogram turns the computer into a clock that displays hours, minutes, and seconds at any rate of speed that the user selects, simplifying the task of running timed operations between stations on a fixed schedule.

The fifth subprogram uses the distance covered in feet and the time in minutes and seconds to determine the equivalent prototype speed in miles per hour.

The sixth subprogram does the reverse. It uses a given prototype speed in miles per hour plus a time factor in minutes and seconds to determine scale distance.

The seventh subprogram determines the distance required in order to effect a change in elevation at a given percent grade, or the percent grade based on the change in elevation and the available distance, or the change in elevation achieved in a given distance at a given percent grade.

The eighth subprogram determines the linear length of track required for a curve of a given radius representing a specified percent of a full circle.

The ninth subprogram uses Ohm's law to find volts from amperes and ohms, ohms from volts and amperes, and amperes from volts and ohms.

The tenth subprogram determines the resistance of electrical wire in ohms, based on wire length in inches and diameter in mils.

---

Line by line:

**Lines 100 and 110** are REMarks.

**Lines 120–380** place the title of the program and the selection of available gauges into display, repeatedly using a group of subroutines, primarily for cosmetic reasons.

**Lines 390–540** send the computer to any of seven subroutines to assign conversion factors to the numeric variable G.

**Lines 550–720** display the menu with the 10 subprograms.

**Line 730** sends the computer to one of 11 line numbers depending on the selection entered.

**Lines 740–950** are used for the first subprogram, where lines 840 and 920 perform the calculations. All other lines should be self-explanatory.

**Lines 960–1010** represent the second subprogram. Line 980 multiplies miles per hour by 5280, the number of feet in a statute mile, and then divides the result by the scale conversion factor and 60 (the number of minutes in an hour) to arrive at a feet-per-minute figure.

**Lines 1020–1320** represent the third subprogram, where lines 1100 and 1220 ask you to key in hours and minutes to be converted. Lines 1110–1190 convert that time into scale time based on the scale conversion factor. Lines 1240–1300 convert the time that was entered, using a percentage figure keyed in by you in line 1230. In each instance the minutes are first separated from the hour figure and then converted to decimal values in order to be manipulated mathematically. They are then converted back to minutes before the result is displayed.

**Lines 1330–1780** produce the scale-time clock in the same way we produced a kitchen timer in Chapter 9.

**Lines 1790–1900** represent the fifth subprogram, with lines 1810–1870 performing the time conversions and related calculations.

**Lines 1910–2000** represent the sixth subprogram and are the reverse of the above.

**Lines 2010–2330** represent the seventh subprogram, where you have three choices with reference to grade data determination. Lines 2160 and 2170 perform the calculations for the first choice.

---

Lines 2230 and 2240 handle the calculations for the second choice. And lines 2300 and 2310 do the same for the third choice. **Lines 2340-2440** take care of the eighth subprogram, and lines 2380 and 2390 perform the calculation. The figure 3.14159 is the value of pi, which is not automatically available in TI BASIC. **Lines 2450-2710** are the ninth subprogram, dealing with Ohm's law. The calculations are made in lines 2570, 2630, and 2690. **Lines 2720-2770** are the tenth subprogram; the calculation is made in line 2750. **Lines 2780-2920** are the various subroutines and the END lines.

## A NUMBERS GAME

The next program deals with numbers and requires a reasonable degree of proficiency in solving mathematical puzzles. It displays a number together with three groups of four numbers each, only one of which can be manipulated mathematically to produce a result that matches the key number. You are asked to identify the group of numbers that fits the criterion. For instance, given:

39

---

A) 7   6   5   2

B) 30   9   3   7

C) 15   7   16   2

---

the A group (on the following page) is the right one:

---



$7*6-5+2$

with the equations figured one calculation at a time from left to right. (No parentheses were used in the equations displayed by the program, though some would require parentheses if the calculations are to be performed by your computer.) If you pick the right number group, the computer tells you so and displays the equations. If you pick the wrong one, you're told the right number group and the equation is also displayed.

The Number Game, contains 12 key numbers and their associated number groups. You can easily add more by writing additional DATA lines; the only change that would have to be made is in line 210, where the number associated with the numeric variable Z must reflect the total number of key numbers in order to avoid a DATA ERROR message.

### NUMBERS GAME PROGRAM

A game with numbers.

---

```
100 REM A GAME WITH NUMBERS
110 REM TI EXTENDED BASIC
120 CALL CLEAR
130 GOTO 170
140 PRINT "-----" :: RETURN
150 FOR X=1 TO 7 :: PRINT :: NEXT X :: RETURN
160 PRINT :: INPUT "Press >ENTER< ":E$:: RETURN
170 GOSUB 140
180 PRINT "This is a game with numbers requiring a degree
```

(continued)

---

```

190 PRINT "I will display a key number and groups of
 secondary numbers. You must pick the group that
 can be used to"
200 PRINT " produce an equation that results in the key
 number using +, -, * and/or / ." :: GOSUB 140 ::
 PRINT :: GOSUB 160
210 Z=Z+1 :: IF Z>12 THEN 580
220 READ KEY,A$,N$,B$,NN$,C$,NNN$,SUM$
230 READ EQUATION$
240 CALL CLEAR
250 PRINT TAB(12);KEY :: GOSUB 140
260 PRINT A$;" ";N$:: PRINT
270 PRINT TAB(7);B$;" ";NN$:: PRINT
280 PRINT TAB(10);C$;" ";NNN$:: GOSUB 140 :: PRINT
290 INPUT "Which group? ":ABC$
300 GOSUB 150
310 IF ABC$=SUM$ THEN 320 ELSE 330
320 GOSUB 140 :: PRINT TAB(10);"That's right" :: PRINT
 "The equation is ";EQUATION$:: GOSUB 140 :: GOSUB
 160 :: GOTO 210
330 GOSUB 140 :: PRINT TAB(10);"That's wrong" :: PRINT "It
 is group ";SUM$;" (" ;EQUATION$;)" :: GOSUB 140 ::
 GOSUB 160 :: GOTO 210
340 DATA 38,A),7 6 5 1,B),30 8 3 6,C),15 7 16 1,A
350 DATA 7*6-5+1
360 DATA 417,A),212 200 1 8,B),25 16 9 8,C),17 50
 8 2,B
370 DATA 25*16+9+8
380 DATA 1,A),15 3 25 30,B),7 10 4 2,C),18 17 0
 6,A
390 DATA 15/3+25/30
400 DATA 99,A),90 9 14 8,B),9 11 7 3,C),33 30 0
 10,C
410 DATA 33*30/10+0
420 DATA 602,A),27 18 212 3,B),-1 20 17 31,C),102
 50 10 6,B
430 DATA -1+20*31-17
440 DATA 0,A),15 102 5.44 37,B),6 11 55 2,C),19 10
 190 -3,A
450 DATA 37*15/102-5.44
460 DATA 15,A),3 4 5 20,B),10 1 4 7,C),145 30 99
 3,A
470 DATA 3+4*5-20
480 DATA 100,A),17 25 23 36,B),7 13 11 2.2,C),50 2
 1 8,B
490 DATA 7+13*11/2.2
500 DATA 11,A),11 1 4 17,B),10 1 77 8,C),3 22 9
 14,B
510 DATA 10+1+77/8
520 DATA 1000,A),500 2 15 7,B),47 6 22 1.04,C),100
 10 5 50,B
530 DATA 47*22+6/1.04

```

(continued)

---



```
540 DATA -10,A),65 110 5 7,B),78 55 22 6,C),15 45
 -65 10,A
550 DATA 7*5+65-110
560 DATA 3.33,A),10 7 18 4,B),69 3.5 30 1.5,C),99.3
 66 2,B
570 DATA 69*1.5-3.5/30
580 CALL CLEAR :: GOSUB 140 :: PRINT TAB(12);"End." ::
 GOSUB 140 :: GOSUB 150 ::
END
```

---

Line by line:

**Lines 100 and 110** are REMarks.

**Lines 120 and 130** clear the screen and send the computer to line 170, skipping over the subroutines in lines 140-160.

**Lines 170-200** display the purpose of the program.

**Line 210** send the computer to the END line if all DATA have been used.

**Lines 220 and 230** READ all DATA items associated with a given key number.

**Line 240** clears the screen.

**Lines 250-280** cause the items to be displayed on the screen.

**Line 290** asks you to key in your selection.

**Line 310** checks whether your selection matches the one stored in the program.

**Lines 320 and 330** are used if your selection is right (320) or wrong (330).

**Lines 340-570** are the DATA block, where the first item is the key number, the second is the letter identification for the first number group, the third is the number group, and so on, with the last item in the line representing the correct number group, and the single item in the following line the equation.

**Line 580** is the END line. If you add more DATA lines, start with line 590, leaving line 580 in place.

## A GAME WITH AIRPLANES

The last program in this group is a sort of home-made video game in which a bunch of airplanes and a hot-air balloon cavort in the sky. I've called it OSHKOSH because the action reminds me of

---

the annual gathering of the thousands of members of the Experimental Aircraft Association during the first week of August in Oshkosh, Wisconsin, where, for the better part of 10 days, the sky is constantly alive with all manner of home-built and antique aircraft.

The program includes four airplanes and one balloon, the speed and direction of travel of which you can control intermittently. It includes some rather primitive sound effects and occasionally a voice will admonish you with "Look out!" when there is an imminent collision. It requires TI EXTENDED BASIC to produce the graphics.

### OSHKOSH

A game with four airplanes and a balloon.

---

```

100 REM GAME WITH AIRCRAFT
110 REM TI EXTENDED BASIC
120 CALL CLEAR :: PRINT "-----"
130 PRINT TAB(12); "Oshkosh" :: PRINT "-----"
 -----"
140 FOR X=1 TO 8 :: PRINT :: NEXT X
150 INPUT "Press >ENTER< ":E$
160 ROW1=20 :: COL1=20
170 ROW2=60 :: COL2=60
180 ROW3=100 :: COL3=100
190 ROW4=110 :: COL4=110
200 ROW5=200 :: COL5=200
210 SC=6
220 UD1=0 :: LR1=40
230 UD2=50 :: LR2=0
240 UD3=-1 :: LR3=60
250 UD4=-1 :: LR4=-60
260 UD5=-1 :: LR5=0
270 CALL CLEAR
280 GOSUB 650
290 B$="3C1B1B1BFFFF1B1B"
300 D$="00008EFF00000000"
310 C$="3C7E7E7E3C1B001B"
320 E$="000071FF00000000"
330 CALL CHAR(96,C$)
340 CALL CHAR(97,B$)
350 CALL CHAR(98,D$)
360 CALL CHAR(99,E$)
370 CALL CHAR(100,D$)
380 CALL SPRITE(#1,100,TINT1,ROW1,COL1)
390 CALL SPRITE(#2,97,TINT2,ROW2,COL2)
400 CALL SPRITE(#3,98,TINT3,ROW3,COL3)
410 CALL SPRITE(#4,99,TINT4,ROW4,COL4)

```

(continued)

---

```
420 CALL SPRITE(#5,96,TINT5,ROWS,COLS)
430 CALL MOTION(#1,UD1,LR1)
440 CALL MOTION(#2,UD2,LR2)
450 CALL MOTION(#3,UD3,LR3)
460 CALL MOTION(#4,UD4,LR4)
470 CALL MOTION(#5,UD5,LR5)
480 Z=0
490 CALL COINC(#1,#2,20,CC)
500 CALL COINC(#1,#3,20,CC)
510 CALL COINC(#1,#4,20,CC)
520 CALL COINC(#1,#5,20,CC)
530 CALL COINC(#2,#3,20,CC)
540 CALL COINC(#2,#4,20,CC)
550 CALL COINC(#2,#5,20,CC)
560 CALL COINC(#3,#4,20,CC)
570 CALL COINC(#3,#5,20,CC)
580 CALL COINC(#4,#5,20,CC)
590 IF CC=-1 THEN GOSUB 990
600 FOR PAUSE=1 TO 1000 :: NEXT PAUSE
610 Z=Z+1 :: GOSUB 1040 :: IF Z=4 THEN 630
620 GOTO 490
630 GOSUB 740
640 GOTO 290
650 INPUT "COLOR, PLANE #1? ":TINT2
660 INPUT "COLOR, PLANE #2? ":TINT3
670 INPUT "COLOR, PLANE #3? ":TINT4
680 INPUT "COLOR, PLANE #4? ":TINT1
690 INPUT "COLOR, BALLOON ? ":TINT5
700 CALL SCREEN(SC)
710 CALL CLEAR
720 CALL MAGNIFY(2)
730 RETURN
740 INPUT "CHANGE SPEED/DIRECTION?(Y/N)":YN$:: IF YN$=
 "N" THEN CALL CLEAR :: RE
 TURN ELSE CALL CLEAR :: GOTO 750
750 PRINT 1;" PLANE GOING DOWN?"
760 PRINT 2;" PLANE GOING EAST?"
770 PRINT 3;" PLANE GOING EAST?"
780 PRINT 4;" PLANE GOING WEST"
790 PRINT 5;" WIND DIR/VEL" :: PRINT "-----"
 "-----"
800 INPUT "WHICH? ":WHICH
810 CALL CLEAR
820 ON WHICH GOTO 830,860,1010,890,920
830 INPUT "SPEED DOWN? ":UD2
840 INPUT "SPEED R/L? ":LR2
850 CALL CLEAR :: RETURN
860 INPUT "SPEED U/D? ":UD3
870 INPUT "SPEED EAST? ":LR3
880 CALL CLEAR :: RETURN
890 INPUT "SPEED U/D? ":UD4
900 INPUT "SPEED WEST? ":LR4
910 CALL CLEAR :: RETURN
```

---

(continued)

```

920 INPUT "WIND VEL (E=-,W=+) ":WV :: CALL CLEAR
930 LR1=LR1+WV
940 LR2=LR2+WV
950 LR3=LR3+WV
960 LR4=LR4+WV
970 LR5=LR5+WV
980 RETURN
990 CALL SAY("LOOK OUT")
1000 RETURN
1010 INPUT "SPEED U/D? ":UD1
1020 INPUT "SPEED EAST? ":LR1
1030 CALL CLEAR :: RETURN
1040 CALL SOUND(4250,-4,0)
1050 CALL SOUND(4250,-8,0)
1060 RETURN

```

---

After displaying its title, the program asks you to determine the colors for the four planes and the balloon. You may choose any color except 1 (transparent) or 6 (light blue), which is the color of the sky. As soon as the colors have been keyed in, the screen changes to light blue and the sky comes alive with action, with two planes traveling east (right) and one traveling west (left) and another apparently performing a never-ending series of loops, while the hot-air balloon rises slowly from the bottom of the screen. After a few seconds a line appears across the bottom of the screen, asking:

**CHANGE SPEED/DIRECTION? (Y/N)**

---

and if you type N, the line disappears, only to reappear after another time period during which the action continues. If you type Y, the line disappears and is replaced by:

```

1 PLANE GOING DOWN?
2 PLANE GOING EAST?
3 PLANE GOING EAST?
4 PLANE GOING WEST?
5 WIND DIR/VEL?

```

WHICH?

giving you a choice of correcting the speed at which any of the four planes is traveling, or changing the wind direction and velocity. Depending on your selection, the above will be replaced with one of five displays:

```

SPEED DOWN?
SPEED R/L?
SPEED U/D?
SPEED EAST?
SPEED U/D?
SPEED WEST?
WIND VEL (E=-, W=+)

```

---

where the SPEED EAST display appears for both east-bound aircraft. The R/L refers to right or left for the aircraft flying the loops. To move it to the left, use a negative number (—1). To move it to the right, use a positive number (1) without the + sign. The U/D controls the climb and descent ratio for aircraft traveling east and west. To make them climb, use a negative number. For descent, use a positive number. The wind direction and velocity affect all aircraft. If you key in an east wind of, say, a value of 10 (—10), the west-bound aircraft will speed up and the east-bound ones will slow down while, at the same time, the looping aircraft and the balloon will both move westward. The wind entries are cumulative (the others are not), meaning that if you want to stop the wind at your next opportunity to enter data, you would have to enter 10 in order to offset the previous —10.

The program will continue indefinitely. To stop it, type FCTN 4 (CLEAR). If you want sound effects and the occasional voice warnings, assuming you have a voice synthesizer, be sure to turn up the volume on your monitor.

Line by line:

**Lines 100 and 110** are REMarks.

**Lines 120–150** place the title of the program into display and ask you to press >ENTER< to continue.

**Lines 160–200** assign the row and column numbers that represent the starting positions to numeric variables.

**Line 210** assigns light blue to a numeric variable.

**Lines 220–260** assign the starting speeds to numeric variables.

**Lines 270 and 280** clear the screen and send the computer to the subroutine (lines 650–730) where you're asked to key in the color numbers for the aircraft. Line 700 controls the color of the sky (screen), and line 720 uses CALL MAGNIFY to double the size of the sprites that represent the aircraft.

**Lines 290–320** assign the different aircraft shapes to string variables.

**Lines 330–370** create the characters that represent those shapes.

**Lines 380–420** create the five sprites.

**Lines 430–470** control the speed with which the aircraft cavort in the sky.

---

**Line 480** assigns the value of zero to the numeric variable Z, which is used later in line 610.

**Lines 490–580** detect imminent collisions, assigning the value of –1 to the numeric variable CC if such a condition is detected. (Depending on the speed of travel, not all impending collisions are detected. The faster the movement, the less the chance of detection.)

**Line 590** sends the computer to a subroutine (lines 990 and 1000) if an impending collision has been detected. That subroutine causes the voice synthesizer to say “Look out!”

**Line 600** produces a pause.

**Line 610** increases the pause. During the pause the action continues. The purpose of the pause is to avoid having a line of text on the bottom of the screen at all times. Each time this line is encountered, the computer is sent to the subroutine that produces the sound effects (lines 1040–1060). At the end of the pause the computer is sent to line 630.

**Line 620** returns the computer to line 490 to detect additional collisions.

**Line 630** sends the computer to a subroutine (starting with line 740) that asks if you want to make changes in the speed and direction of travel.

**Line 640** returns the computer to line 290 to continue the action of the aircraft.

**Lines 650–730** are the subroutine discussed above.

**Line 740** asks if you want to make a change. If not, it returns the computer to line 640. If you do, it goes to the next line.

**Lines 750–800** offer you the choice of aircraft, the speed or direction of which you want to change.

**Line 820** sends the computer to one of five lines, depending on your selection.

**Lines 830–850, 860–880, 890–980, or 1010–1030** are called into play to give you the chance to type in the desired changes. In lines 930–970 the left/right speeds are changed in accordance with the keyed-in wind direction and velocity factor. At the end of any of these five sections, the copy lines are cleared from the screen and the computer is returned to line 640.

---

**Lines 990 and 1000** are the subroutine that activates the speech synthesizer.

**Lines 1040-1060** produce the intermittent sound effects.

Once you become familiar with the use of the family of sprite subprograms, you can easily create all manner of fascinating games.



---

# 13

---

## ***Programs for Business or Profession***

---

---

Although your TI-99/4A Home Computer, as the name implies, is designed primarily for use in the home, it does offer ample capabilities for use in connection with business matters or in the pursuit of a profession. The programs in this chapter provide examples of the kinds of programs you might want to write in order to deal with problems that may be unique to your work.

All of the programs in this chapter are written in TI EXTENDED BASIC, but with very few changes they can easily be translated into TI BASIC if the extended version is not available. Several programs assume that the system includes a line printer—they are designed to produce certain documents automatically while the program is being executed.

### **LOAN, SAVING, AND INVESTMENT PROGRAM**

This program deals with the cost of borrowing money and the returns that can be expected from savings or investment under any given set of circumstances. It consists of three subprograms. The first asks you to enter the amount available for investment plus the

---

available rate of interest and the compounding period. It then requires you to key in a time period, after which it displays the value of your investment at the end of that period (see Figure 13-1 on page 340).

### LOAN SAVING AND INVESTMENT PROGRAM

A program that deals with investment and loan or mortgage data.

---

```

100 REM SAVINGS AND LOAN DATA
110 REM TI EXTENDED BASIC
120 H=0
130 CALL CLEAR
140 GOTO 180
150 PRINT "-----" :: RETURN
160 FOR X=1 TO 8 :: PRINT :: NEXT X :: RETURN
170 INPUT "Press >ENTER< ":E$:: RETURN
180 LL$="-----"
-----"
190 GOSUB 150
200 PRINT "This program analyzes loan, mortgage, saving
and invest-ment data" :: PRINT
210 PRINT "You have three choices:" :: PRINT
220 PRINT 1;" Loan/mortgage analysis"
230 PRINT 2;" Savings/investment data"
240 PRINT 3;" Cost comparisons" :: GOSUB 150
250 INPUT "Which? ":WHICH
260 PRINT :: PRINT :: INPUT "Printout? (Y/N) ":NN$
270 CALL CLEAR :: ON WHICH GOTO 610,280,950
280 Y=0 :: M=0 :: D=0
290 PRINT "Savings/investment data" :: GOSUB 150
300 INPUT "Present value? $":L
310 INPUT "Annual interest rate? %":I
320 PRINT :: PRINT "Compounding periods:" :: GOSUB 150
330 PRINT 1,"Daily"
340 PRINT 2,"Monthly"
350 PRINT 3,"Yearly" :: GOSUB 150
360 INPUT "Which? ":PERIOD
370 I=I/100 :: IF H=1 THEN 1050
380 CALL CLEAR :: PRINT "Term to be examined:" :: PRINT
390 INPUT "Number of years? ":Y
400 INPUT "Number of months? ":M
410 INPUT "Number of days? ":D
420 CALL CLEAR :: ON PERIOD GOTO 430,440,470
430 I=I/365.25 :: GOTO 450
440 I=I/12 :: GOTO 470
450 CP=(Y*365.25)+(M*30.44)+D
460 GOTO 480

```

(continued)

---

```
470 CP=Y+(M/12)+(D/365.25)
480 FV=L*(1+I)^CP :: FV=INT(FV*100+.5)/100
490 CALL CLEAR :: GOSUB 150 :: PRINT "Future value= $"
 :FV :: GOSUB 150
500 GOSUB 150
510 IF NN$="Y" THEN GOSUB 540
520 PRINT :: INPUT "Another run? (Y/N) ":Y$:: CALL CLEAR
530 IF Y$="Y" THEN 180 ELSE END
540 OPEN #1:"RS232"
550 PRINT #1:"Present value.....$";L
560 J=I*100 :: IF PERIOD=1 THEN J=J*365.25 :: IF PERIOD=2
 THEN J+J*12
570 PRINT #1:"Annual interest rate.....";J;"%"
580 PRINT #1:"Period ";Y;" years, ";M;" months, ";D;" days
 ." :: PRINT #1:LL$
590 PRINT #1:"Future value.....$";FV :: PRINT
 #1:LL$
600 CLOSE #1 :: RETURN
610 CALL CLEAR :: GOSUB 150
620 PRINT "Loan/mortgage data" :: GOSUB 150
630 INPUT "Amount of loan or mortgage? ":N
640 INPUT "Annual interest rate? %":C :: C=C/100
650 IF H=2 THEN 680
660 D=N*C/12 :: D=INT(D*100+.5)/100 :: CALL CLEAR
670 PRINT "The absolute minimum payment to cover the
 interest= $";D :: GOSUB 150
680 INPUT "The monthly payment that you feel you can
 afford? $":E :: G=1
690 F=C/12 :: I=F*N+N-E
700 G=G+1
710 I=I*F+I-E
720 IF H=2 THEN 1320
730 IF I<E THEN 750
740 IF I>E THEN 700
750 IF H=2 THEN 1320
760 CALL CLEAR :: PRINT "Number of payments= ":G :: PRINT
770 I=INT(I*100+.5)/100
780 PRINT "Plus a final payment of $";I
790 V=E*G+I :: V=INT(V*100+.5)/100
800 W=V-N :: W=INT(W*100+.5)/100 :: GOSUB 150
810 PRINT "Total cost of loan= $";V :: PRINT
820 PRINT "Total interest= $";W
830 IF NN$="Y" THEN GOSUB 850
840 GOTO 520
850 OPEN #1:"RS232"
860 PRINT #1:"Amount of loan or mortgage.....$";N :: C1
 =C*100
870 PRINT #1:"Annual interest rate.....";C1;"%"
 :: PRINT #1:LL$
880 PRINT #1:"Minimum payment to cover interest.$";D
890 PRINT #1:"Payment you want to make.....$";E ::
 PRINT #1:LL$
```

(continued)

```

900 PRINT #1:"Number of payments.....";G
910 PRINT #1:"Plus a final payment of.....$";I ::
 PRINT #1:LL$
920 PRINT #1:"Total cost of loan or mortgage....$";V
930 PRINT #1:"Total interest.....$";W ::
 PRINT #1:LL$
940 CLOSE #1 :: RETURN
950 VI$="Value of investment, year #" :: CALL CLEAR ::
 GOSUB 150
960 PRINT "Loan or mortgage cost versus investment returns"
 :: GOSUB 150
970 INPUT "Capital to invest? $":L
980 INPUT "Available interest rate? ":I
990 H=1
1000 OPEN #1:"RS232"
1010 PRINT #1:"Investment.....$";L
1020 J=I
1030 PRINT #1:"Interest rate.....";J;"%" ::
 PRINT #1:LL$
1040 GOTO 320
1050 CALL CLEAR
1060 IF PERIOD=1 THEN 1090
1070 IF PERIOD=2 THEN 1100
1080 IF PERIOD=3 THEN 1110
1090 CP=365.25 :: I=I/CP :: GOTO 1230
1100 CP=12 :: I=I/CP :: GOTO 1250
1110 CP=1
1120 READ K
1130 CP=1
1140 CP=CP*K :: FV=L*(1+I)^CP :: FV=INT(FV*100+.5)/100
1150 PRINT VI$;K;" $";FV
1160 IF NN$="N" THEN 1180
1170 PRINT #1:VI$;K;" $";FV
1180 IF K=20 THEN 1270
1190 IF K=10 THEN GOSUB 170
1200 IF PERIOD=1 THEN 1230
1210 IF PERIOD=2 THEN 1250
1220 IF PERIOD=3 THEN 1120
1230 READ K
1240 CP=365.25 :: GOTO 1140
1250 READ K
1260 CP=12 :: GOTO 1140
1270 PRINT
1280 INPUT "For loan data press >ENTER< ":Y$
1290 H=2
1300 LB$="Loan balance for month #" :: CALL CLEAR :: GOSUB
 150
1310 GOTO 630
1320 IF NN$="N" THEN 1370
1330 PRINT #1:"Amount of loan or mortgage.....$";N
1340 C1=C*100
1350 PRINT #1:"Annual interest rate.....";C1;"%"

```

(continued)

---

```
1360 PRINT #1:"Monthly payment.....$";E ::
 PRINT #1:LL$
1370 I=INT(I*100+.5)/100 :: PRINT
1380 PRINT LB$;G;" $";I
1390 IF NN$="N" THEN 1410
1400 PRINT #1:LB$;G;" $";I
1410 IF I<0 THEN 1430
1420 GOTO 700
1430 CLOSE #1 :: GOTO 520
1440 DATA 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,
 20
1450 END
```

---

|                                   |
|-----------------------------------|
| Present value.....\$ 5000         |
| Annual interest rate..... 7.25 %  |
| Period 5 years, 6 months, 0 days. |
| -----                             |
| Future value.....\$ 7449.53       |
| -----                             |

**Figure 13-1.** Printout showing the future value of savings or an investment.

The second subprogram deals with loans and mortgages. It calls for the entry of the loan or mortgage principal and the annual percentage rate. With that information it calculates the absolute minimum monthly payment that must be made in order to cover just the interest. After that it asks you to key in the payment amount you feel you can afford to make, an amount that obviously must be larger than the one calculated by the computer. It then displays the number of monthly payments and the amount of the final payment needed in order to retire the loan or mortgage. Figure 13-2 shows a sample printout, and Figure 13-3 shows the loan balance as calculated for the first 12 months of the mortgage.

---

```

Amount of loan or mortgage.....$ 25000
Annual interest rate..... 12.25 %

Minimum payment to cover interest.$ 255.21
Payment you want to make.....$ 1000

Number of payments..... 29
Plus a final payment of.....$ 10.77

Total cost of loan or mortgage....$ 29010.77
Total interest.....$ 4010.77

```

Figure 13-2. Printout showing analysis of a \$25,000 mortgage.

```

Amount of loan or mortgage.....$ 25000
Annual interest rate..... 12.25 %
Monthly payment.....$ 1000

Loan balance for month # 2 $ 23502.81
Amount of loan or mortgage.....$ 25000
Annual interest rate..... 12.25 %
Monthly payment.....$ 1000

Loan balance for month # 3 $ 22742.73
Amount of loan or mortgage.....$ 25000
Annual interest rate..... 12.25 %
Monthly payment.....$ 1000

Loan balance for month # 4 $ 21974.9
Amount of loan or mortgage.....$ 25000
Annual interest rate..... 12.25 %
Monthly payment.....$ 1000

Loan balance for month # 5 $ 21199.23
Amount of loan or mortgage.....$ 25000
Annual interest rate..... 12.25 %
Monthly payment.....$ 1000

Loan balance for month # 6 $ 20415.64
Amount of loan or mortgage.....$ 25000
Annual interest rate..... 12.25 %
Monthly payment.....$ 1000

```

(continued)

```

Loan balance for month # 7 $ 19624.05
Amount of loan or mortgage.....$ 25000
Annual interest rate..... 12.25 %
Monthly payment.....$ 1000

Loan balance for month # 8 $ 18824.38
Amount of loan or mortgage.....$ 25000
Annual interest rate..... 12.25 %
Monthly payment.....$ 1000

Loan balance for month # 9 $ 18016.55
Amount of loan or mortgage.....$ 25000
Annual interest rate..... 12.25 %
Monthly payment.....$ 1000

Loan balance for month # 10 $ 17200.47
Amount of loan or mortgage.....$ 25000
Annual interest rate..... 12.25 %
Monthly payment.....$ 1000

Loan balance for month # 11 $ 16376.06
Amount of loan or mortgage.....$ 25000
Annual interest rate..... 12.25 %
Monthly payment.....$ 1000

Loan balance for month # 12 $ 15543.23
Amount of loan or mortgage.....$ 25000
Annual interest rate..... 12.25 %
Monthly payment.....$ 1000

```

**Figure 13-3.** Printout showing the first 12 months in the life of a mortgage.

The third subprogram requires similar data input but then displays the future value of the investment at the end of each of 20 years as well as the remaining loan balance at the end of each month throughout the life of the loan or mortgage (see Figure 13-4). All three subprograms can be run repeatedly in order to compare the results achieved with different variables, and all include the option of having the results sent to a line printer for a hard copy.

```

Investment.....$ 5000
Interest rate..... 7.25 %

Value of investment, year # 1 $ 5375.93
Value of investment, year # 2 $ 5780.11
Value of investment, year # 3 $ 6214.69
Value of investment, year # 4 $ 6681.95
Value of investment, year # 5 $ 7184.33
Value of investment, year # 6 $ 7724.48
Value of investment, year # 7 $ 8305.25
Value of investment, year # 8 $ 8929.68
Value of investment, year # 9 $ 9601.06
Value of investment, year # 10 $ 10322.91
Value of investment, year # 11 $ 11099.04
Value of investment, year # 12 $ 11933.52
Value of investment, year # 13 $ 12830.75
Value of investment, year # 14 $ 13795.43
Value of investment, year # 15 $ 14832.64
Value of investment, year # 16 $ 15947.83
Value of investment, year # 17 $ 17146.87
Value of investment, year # 18 $ 18436.06
Value of investment, year # 19 $ 19822.17
Value of investment, year # 20 $ 21312.51

```

**Figure 13-4.** Printout showing annual results for an investment for 20 years.

Line by line:

**Lines 100 and 110** are REMarks.

**Line 120** makes sure that no leftover value is assigned to the numeric variable H.

**Lines 130 and 140** clear the screen and cause the computer to jump over the subroutines in the next three lines.

**Lines 150–170** are three subroutines used frequently throughout the program, placing a dashed line into display, causing text to be centered on the screen, and asking you to press >ENTER< in order to continue program execution.

**Line 180** assigns a dashed line to the string variable LL\$, which is used when the line printer option is called up.

---



**Lines 190–260** place the explanation of the program and the three choices into display, asking you to decide which program you want to use and whether or not you want the resulting data printed. There is a hyphen in “invest-ment” because the line wraps around at that point when that line of copy is displayed.

**Line 270** clears the screen and sends the computer to one of three line numbers depending on your choice of subprograms.

**Line 280** assigns zero to three numeric variables.

**Lines 290–600** deal with savings or investment data, asking you to key in the present value representing the amount of capital available for the purpose, the available annual interest rate, and the compounding period, assigning the present value to L, the interest rate to I, and the compounding period to PERIOD.

**Line 370** divides the interest by 100 for use in subsequent calculations and checks on the value assigned to H, which has to do with the third subprogram, because it uses some of the same input lines.

**Lines 380–410** ask you to key in the time period for which you want the data to be figured.

**Line 420** sends the computer to one of three line numbers depending on the compounding period.

**Lines 430–480** perform the required calculations, and line 490 displays the result.

**Line 510** checks whether you want the results printed, in which case the computer is sent to the subroutine consisting of lines 540–600.

**Lines 520 and 530** check whether you want to run one or another portion of the program again in which case the computer is returned to line 180. If not, the program terminates.

**Lines 610–940** represent the portion of the program that deals with loans and mortgages, asking you first to key in the principal amount and the rate of interest, assigning the data to the variables N and C.

**Line 660** calculates the minimum payment, which covers the interest but does not reduce the principle.

**Lines 670 and 680** display the result and then ask you to key in the payment amount you want to make. The lack of spaces between words (paymentto, youfeel) again has to do with the fact

---

that the copy wraps around at that point. Your entry is assigned to the variable E, and the value 1 is assigned to the variable G, which is used only in the third subprogram.

**Lines 690–750** perform a number of calculations and send the computer to a number of different line numbers, depending on the values of several variables.

**Lines 760–820** display the results and perform some additional calculations.

**Lines 830–840** check if you want the results printed and send the computer to the subroutine (lines 850–940) that activates the printer, or back to line 520, which asks if you want to run the program again.

**Lines 950–1500** represent the third subprogram, which uses portions of the two previous programs.

**Line 950** assigns a string to a string variable.

**Line 960** prints the title of the subprogram.

**Lines 970 and 980** ask you to key in the capital and interest data.

**Line 990** assigns the value of 1 to the variable H, which later tells the computer that we're dealing with the first part of the third subprogram.

**Lines 1000–1030** cause your input data to be printed.

**Line 1040** sends the computer to line 320, where you're asked to key in the compounding period, and in line 370 the computer finds that the value assigned to H is 1, sending it back to line 1050.

**Lines 1050–1080** send the computer to one of three line numbers depending on the compounding periods.

**Lines 1090–1110** perform a group of calculations.

**Line 1120** is used to READ the numbers stored in the DATA block that represent the year numbers 1 through 20.

**Lines 1130 and 1140** calculate the future value, and line 1150 causes the result to be displayed.

**Line 1160** checks if you want the results printed and, if yes, line 1170 sends the data to the line printer.

**Lines 1180 and 1190** check the value of K. If it's 10, then you're asked to press >ENTER< because the screen can display only so many lines at a time. If it's 20, the computer is told to go to line 1270, which is the start of the second portion of the third subprogram.

**Lines 1200–1260** repeat the above.

---

**Lines 1270–1430** represent the loan/mortgage portion of this section.

**Line 1290** assigns the value of 2 to the variable H to be used later to tell the computer which is the active program portion.

**Line 1300** assigns a string to a string variable.

**Line 1310** sends the computer to line 630, where you're asked to enter the principal and interest data. In line 650 the computer finds the value assigned to H and is sent to line 680 for additional input. Lines 720 and 750 again use the value of H to send the computer back to line 1320.

**Line 1320** checks if you want the results printed. Here the answer should be in the affirmative, because the results will involve many more lines than can be displayed at one time.

**Lines 1330–1400** perform the printing function or, if printing is rejected, line 1380 causes the data to be sent to the screen.

**Line 1410** checks the value of I, which, at this point, represents the remaining loan balance. If the balance is less than zero (a negative figure), then the computer is sent to line 520 to ask if you want another run. Otherwise it is sent to line 700 to continue the progressive calculations.

**Line 1440** is the DATA block, and line 1450 is the END line.

## **CURRENCY CONVERSION PROGRAM**

Next is a very short program that comes in handy if you're involved in importing or exporting products or if you often travel in foreign countries. Its purpose is to figure the value of foreign currencies in terms of dollars or vice versa. You have the choice of converting *to* or *from* any foreign currency, but there is one hitch: If the foreign currency denomination is valued at more than one dollar (for instance, the English pound), then the to/from selection must be reversed, which is a bit awkward, but not serious. It would have been possible, of course, to add another section to take care of this problem, as is shown at the end of the line-by-line explanation.

---

## CONVERTING CURRENCIES PROGRAM

A program that converts foreign currencies to U.S. dollars and vice versa.

---

```

100 REM CONVERTING FOREIGN CURRENCIES
110 REM TI EXTENDED BASIC
120 CALL CLEAR
130 GOTO 170
140 PRINT "-----" :: RETURN
150 FOR X=1 TO 8 :: PRINT :: NEXT X :: RETURN
160 INPUT "Press >ENTER< ":E$:: RETURN
170 GOSUB 140
180 PRINT "This program can be used to convert foreign
 currencies." :: GOSUB 140 :: GOSUB 150 :: GOSUB 160
190 CALL CLEAR :: PRINT "Menu:" :: GOSUB 140
200 PRINT 1;" Convert TO U.S. money" :: PRINT
210 PRINT 2;" Convert FROM U.S. money" :: GOSUB 140
220 PRINT 3;" Exit the program" :: GOSUB 140
230 INPUT "Which? ":WHICH
240 RE$="Today's rate of exchange" :: AC$="Amount to be
 converted"
250 TM$="Name of currency"
260 CALL CLEAR
270 ON WHICH GOTO 280,330,380
280 PRINT TM$:: INPUT NM$:: PRINT
290 PRINT RE$:: INPUT R :: PRINT
300 PRINT AC$:: INPUT A :: GOSUB 140 :: GOSUB 150
310 C=A/R :: C=INT(C*100+.5)/100
320 PRINT A;NM$;"= $";C :: GOSUB 140 :: GOSUB 160 :: GOTO
 190
330 PRINT TM$:: INPUT NM$:: PRINT
340 PRINT RE$:: INPUT R :: PRINT
350 PRINT AC$:: INPUT A :: GOSUB 140 :: GOSUB 150
360 C=A*R :: C=INT(C*100+.5)/100
370 PRINT "$";A;"= ";NM$;C :: GOSUB 140 :: GOSUB 160 ::
 GOTO 190
380 GOSUB 140 :: PRINT TAB(12);"End." :: GOSUB 140 :: G
 GOSUB 150 :: END

```

---

When you run the Converting Currencies Program it asks whether you want to convert *to* or *from* the U.S. dollar. It then asks

---

you to key in the name of the foreign currency (yen, lira, etc.) and the amount to be converted. After that it displays either:

`$xx.xx = xx.xx (currency name)`

or

`xx.xx = (currency name = $xx.xx`

And then it returns to the menu to offer you the choice of making another conversion or exiting the program.

Line by line:

**Lines 100–160** are the usual REMark lines and the subroutines.

**Lines 170–230** display the program title and the menu.

**Lines 240 and 250** assign strings to string variables.

**Line 270** tells the computer where to go, based on your selection from the menu.

**Lines 280–300** are used if you convert *to* U.S. currency, asking you to key in the name of the foreign currency, the current rate of exchange, and the amount of foreign currency to be converted.

**Line 310** performs the conversion, limiting the display to two decimal places.

**Line 320** displays the result and then returns the menu to the screen.

**Lines 330–350** are used if you want to convert *from* U.S. currency, asking the same set of questions.

**Line 360** performs the conversion.

**Line 370** displays the result, and line 380 is the END line.

---

If you feel that you would like to add a section that can deal with the English pound (and other currencies with denominations valued in excess of one dollar), make the following changes and additions:

```
235 Z=0
270 CALL CLEAR::ON WHICH GOTO 275,325,380
275 Z=1::GOTO 400
305 IF Z=1 THEN 360
315 IF Z=2 THEN 370
325 Z=2::GOTO 400
355 IF Z=2 THEN 310
365 IF Z=1 THEN 320
400 PRINT 1;" U.S.$ is worth more than other currency"
410 PRINT 2;" U.S. $ worth less than other
 currency "::GOSUB 140
420 INPUT "Which? ":W
430 CALL CLEAR
440 IF Z=1 THEN ON W GOTO 330,280 ELSE ON W GOTO
 280,330
```

With these lines added to the existing program, the computations and displays will be correct regardless of the relationship of the two currencies.

## **INVOICE WRITER**

The next program, requires that a line printer be part of the system and loaded with paper and turned on when it is run, because it prints an invoice while you're typing, calculating all sub-totals and totals automatically. Figure 13-5 on page 353 is an example of what the final invoice looks like. When you execute the program, it immediately prints the letterhead and then asks you to type in the addressee information, the order number, and the date of shipment. It then calls for the actual billing information, and then

---

asks if additional entries are to be made. When all billing items have been typed in, it asks:

Shipping and handling?  
Tax % (if applicable)  
Amount prepaid?

---

### INVOICE WRITER PROGRAM

---

A program that produces printed invoices.

---

```
100 REM INVOICE WRITER
110 REM TI EXTENDED BASIC
120 CALL CLEAR
130 GOTO 170
140 PRINT #1:"-----"
150 PRINT "-----" :: RETURN
160 FOR X=1 TO 8 :: PRINT :: NEXT X :: RETURN
170 GOSUB 150
180 PRINT "This program prints invoiceswhile you are
 typing. Be sure that the printer is ready!" ::
 GOSUB 150 :: GOSUB 160
190 INPUT "Press >ENTER< ":E$
200 OPEN #1:"RS232"
210 PRINT #1:"
 COMPANY"
220 PRINT #1:"
 STREET"
230 PRINT #1:"
 87501"
 ABC
 101 MAIN
 SANTA FE, NM
```

(continued)

---

```

240 PRINT #1:"
1212"
250 GOSUB 140
260 CLOSE #1
270 INPUT "Customer company name? " : CC$
280 INPUT "Attention? " : TA$
290 INPUT "Street or P.O. Box? " : PB$
300 INPUT "City, State, Zip? " : CS$
310 OPEN #1:"RS232"
320 PRINT #1
330 PRINT #1:"
I N V O I C E"
340 PRINT #1:"
=="
350 PRINT #1:CC$
360 PRINT #1:"Att.: " ; TA$
370 PRINT #1:PB$
380 PRINT #1:CS$
390 GOSUB 140
400 CLOSE #1
410 INPUT "Your order number? " : YO$
420 INPUT "Date shipped? " : DS$
430 N$="No." :: O$="ORDERED" :: S$="SHIPPED" :: D$=
"DESCRIPTION" :: U$="UNIT"
440 P$="PRICE" :: UU$="ITEMS PER" :: A$="AMOUNT" :: T$=
"Total= $"
450 OPEN #1:"RS232"
460 PRINT #1:"Your order number: " ; YO$
470 PRINT #1:"Date shipped: " ; DS$
480 GOSUB 140
490 PRINT #1:N$;" " ; N$;" " ; D$;" " ; U$;"
";UU$;" " ; A$
500 PRINT #1:O$;" " ; S$;" " ; P$;" " ; U$
510 GOSUB 140 :: GOSUB 140
520 CLOSE #1
530 INPUT "Number ordered? " : NN
540 INPUT "Number shipped? " : SS
550 INPUT "Description? " : DD$
560 INPUT "Unit price? $" : UP
570 INPUT "Units per item? " : UI
580 A=SS*UP :: AA=AA+A
590 A=INT(A*100+.5)/100 :: AA=INT(AA*100+.5)/100
600 OPEN #1:"RS232"
610 PRINT #1:NN;TAB(9);SS;TAB(18);DD$;TAB(34);"$";UP;TAB
(46);UI;TAB(60);"$";A
620 CLOSE #1
630 PRINT "Do you want to add more items?" :: INPUT
"Y/N " : YN$
640 IF YN$="Y" THEN 530
650 OPEN #1:"RS232"
660 GOSUB 140
670 PRINT #1:"This order, " ; T$;AA

```

(continued)

---



```
680 GOSUB 140
690 CLOSE #1
700 SH$="Shipping and handling" :: TT$="Sales tax"
710 CALL CLEAR
720 INPUT "Shipping and handling? ":SH
730 INPUT "Tax % (if applicable) ":TT
740 INPUT "Amount prepaid? $":PP
750 TT=AA*(TT/100):: TT=INT(TT*100+.5)/100
760 OPEN #1:"RS232"
770 PRINT #1:SH$;" $";SH
780 PRINT #1:TT$;" $";TT
790 PRINT #1
800 AA=AA+TT+SH :: AA=INT(AA*100+.5)/100
810 PRINT #1:T$;AA
820 PRINT #1:"Amount prepaid $";PP
830 PRINT #1
840 GOSUB 140
850 AA=AA-PP
860 PRINT #1:"Amount due $";AA
870 GOSUB 140
880 PRINT #1
890 PRINT #1:"Thank you."
900 CLOSE #1
910 END
```

---

and, once the information has been entered, the computer calculates the totals and prints the rest of the invoice.

Line by line:

**Lines 100–160** are the usual REMark lines and subroutines.

**Lines 180 and 190** remind you that the printer must be ready.

The missing space (invoiceswhile) is there because of the wraparound. You're then asked to press >ENTER< to start the program.

**Lines 200–260** cause the letterhead to be printed. Those lines will, of course, have to be rewritten and respaced to fit the information that is applicable to your operation.

**Lines 270–300** ask you to key in the customer name, address, etc.

**Lines 310–400** print the word INVOICE and the customer data.

**Lines 410 and 420** ask for order number and shipping date.

**Lines 430 and 440** assign a number of strings to string variables.

**Line 450** activates the printer again.

---

|                                                                                     |     |             |            |                |           |
|-------------------------------------------------------------------------------------|-----|-------------|------------|----------------|-----------|
| ABC COMPANY<br>101 MAIN STREET<br>SANTA FE, NM 87501<br>505 555 1212                |     |             |            |                |           |
| -----                                                                               |     |             |            |                |           |
| <b>I N V O I C E</b><br>=====                                                       |     |             |            |                |           |
| XYZ Manufacturing Company<br>Att.: Hector Jones<br>P.O.Box 1234<br>Anytown CA 91234 |     |             |            |                |           |
| -----                                                                               |     |             |            |                |           |
| Your order number: 527<br>Date shipped: 8/25/83                                     |     |             |            |                |           |
| -----                                                                               |     |             |            |                |           |
| No.                                                                                 | No. | DESCRIPTION | UNIT PRICE | ITEMS PER UNIT | AMOUNT    |
| -----                                                                               |     |             |            |                |           |
| 25                                                                                  | 22  | WIDGETS     | \$ 24.95   | 1              | \$ 548.9  |
| 15                                                                                  | 12  | GADGETS     | \$ 12.95   | 1              | \$ 155.4  |
| 50                                                                                  | 48  | GIMMICKS    | \$ 99.5    | 1              | \$ 4776   |
| -----                                                                               |     |             |            |                |           |
| This order, Total=                                                                  |     |             |            |                | \$ 5480.3 |
| -----                                                                               |     |             |            |                |           |
| Shipping and handling \$ 12.5<br>Sales tax \$ 369.92                                |     |             |            |                |           |
| -----                                                                               |     |             |            |                |           |
| Total=                                                                              |     |             | \$ 5862.72 |                |           |
| Amount prepaid                                                                      |     |             | \$ 50      |                |           |
| -----                                                                               |     |             |            |                |           |
| Amount due                                                                          |     |             | \$ 5812.72 |                |           |
| -----                                                                               |     |             |            |                |           |
| Thank you.                                                                          |     |             |            |                |           |

Figure 13-5. A sample invoice produced by INVOICE WRITER.

**Lines 460 and 470** print the above information.

**Lines 490-510** print the column headings and print two dashed lines.

**Lines 530-570** ask you to enter the billing data.

**Lines 580 and 590** calculate the subtotal for each item.

**Lines 600-620** send the billing data along with the subtotal to the line printer.

**Lines 630 and 640** ask whether or not you want to add more items to the invoice.

**Lines 650–690** print the total of all subtotals.

**Line 700** assigns additional strings to string variables.

**Lines 720–740** ask for additional data input.

**Lines 750, 800, and 850** perform the additional calculations to take shipping and handling, taxes, and the prepaid amount into account.

**Lines 760–900** cause the results of those calculations to be printed, and line 910 is the END line.

I might point out that I included all those OPEN #1 and CLOSE #1 statements to clarify the action of the printer. In practice all but the first OPEN #1 and the last CLOSE #1 can be eliminated, because the printer will not pay attention to any program lines that start with something other than PRINT #1.

## ***A LEDGER SHEET PROGRAM***

While we're talking about programs that automatically print a document, here is another. The Ledger Program prints an eight-column ledger sheet, where one column is used for job description and seven columns can be assigned to contain either debit or credit data. Figure 13-6 on page 358 is a sample of what the final printout looks like. Here columns 1 and 2 were used for credit data and columns 3 through 7 for debit data. In addition to printing the ledger and the entered data and calculating and printing the various subtotals and totals, the program creates two separate data files that remember which column was assigned to which type of data, and all of the final subtotals and totals. Thus, the next time the program is run, it starts by displaying the previous balances.

---

## LEDGER SHEET PROGRAM

A program that produces a multicolumn ledger sheet and a data file that records previous balances.

---

```

100 REM LEDGER
110 REM TI EXTENDED BASIC
115 CALL CLEAR
120 GOTO 200
130 PRINT "-----" :: RETURN
140 FOR X=1 TO 8 :: PRINT :: NEXT X :: RETURN
150 INPUT "Press >ENTER< ":E$:: RETURN
180 PRINT #1:"-----" :: RETURN
 "-----" :: RETURN
200 GOSUB 130
210 PRINT "This program prints a multi-column ledger and
 it enters data while you are typing. Be sure that the
 printer is ready."
220 GOSUB 130 :: GOSUB 140 :: GOSUB 150
230 GOTO 600
235 CALL CLEAR
240 PRINT "You have eight columns, one for job description,
 seven for data entry." :: PRINT
250 PRINT "Please decide which columns to use for credits
 and whichfor debit entries." :: GOSUB 140 :: GOSUB 150
260 C$="Cq1." :: CR$=", credit/debit?(C/D)"
270 PRINT C$;1;CR$:: INPUT CD1$
280 PRINT C$;2;CR$:: INPUT CD2$
290 PRINT C$;3;CR$:: INPUT CD3$
300 PRINT C$;4;CR$:: INPUT CD4$
310 PRINT C$;5;CR$:: INPUT CD5$
320 PRINT C$;6;CR$:: INPUT CD6$
330 PRINT C$;7;CR$:: INPUT CD7$
370 E$="Expenses= $" :: R$="Receipts= $" :: GT$="Grand
 total= $" :: T$="Total= $" :: B$="Last balance= $"
400 CALL CLEAR :: PRINT "Enter last balances:" :: GOSUB
 130
410 PRINT C$;1 :: INPUT "$":C1
420 PRINT C$;2 :: INPUT "$":C2
430 PRINT C$;3 :: INPUT "$":C3
440 PRINT C$;4 :: INPUT "$":C4
450 PRINT C$;5 :: INPUT "$":C5
460 PRINT C$;6 :: INPUT "$":C6
470 PRINT C$;7 :: INPUT "$":C7
480 CALL CLEAR :: GOTO 2100
500 PRINT "Make new entries:" :: GOSUB 130
510 INPUT "Job description? ":JD$
520 PRINT C$;1 :: INPUT "$":CC1
530 PRINT C$;2 :: INPUT "$":CC2
540 PRINT C$;3 :: INPUT "$":CC3
550 PRINT C$;4 :: INPUT "$":CC4

```

(continued)

---

```
560 PRINT C$;5 :: INPUT "$":CC5
570 PRINT C$;6 :: INPUT "$":CC6
580 PRINT C$;7 :: INPUT "$":CC7
590 CALL CLEAR :: GOTO 1100
600 CALL CLEAR :: PRINT "Menu:" :: GOSUB 130
610 PRINT 1;"Enter new balances"
620 PRINT 2;"Use previous balances" :: GOSUB 130
630 INPUT "Which? ":WHICH
640 ON WHICH GOTO 235,2000
1000 OPEN #1:"RS232"
1005 PRINT #1:"DESCRIPTION";TAB(15);C$;1;TAB(23);C$;2;TAB
(31);C$;3;TAB(39);C$;4;TAB(47);C$;5;TAB(55);C$;6;TAB
(63);C$;7
1006 GOSUB 180
1010 PRINT #1:TAB(15);C1;TAB(23);C2;TAB(31);C3;TAB(39);C4;
TAB(47);C5;TAB(55);C6;TAB(63);C7
1015 GOSUB 180
1020 GOTO 500
1100 CALL CLEAR
1105 PRINT #1:JD$;TAB(15);CC1;TAB(23);CC2;TAB(31);CC3;TAB
(39);CC4;TAB(47);CC5;TAB(55);CC6;TAB(63);CC7
1110 A=A+1
1120 INPUT "Another entry? (Y/N) ":YY$
1130 IF A>1 THEN 1150
1140 T1=C1+CC1 :: T2=C2+CC2 :: T3=C3+CC3 :: T4=C4+CC4 ::
T5=C5+CC5 :: T6=C6+CC6 :: T7=C7+CC7
1145 IF A<2 THEN 1160
1150 T1=T1+CC1 :: T2=T2+CC2 :: T3=T3+CC3 :: T4=T4+CC4 ::
T5=T5+CC5 :: T6=T6+CC6 :: T7=T7+CC7
1160 IF YY$="N" THEN 1200 ELSE 510
1200 IF CD1$="C" THEN 1310 ELSE 1410
1210 IF CD2$="C" THEN 1320 ELSE 1420
1220 IF CD3$="C" THEN 1330 ELSE 1430
1230 IF CD4$="C" THEN 1340 ELSE 1440
1240 IF CD5$="C" THEN 1350 ELSE 1450
1250 IF CD6$="C" THEN 1360 ELSE 1460
1260 IF CD7$="C" THEN 1370 ELSE 1470
1310 GOTO 1210
1320 TT=T1+T2 :: GOTO 1220
1330 TT=TT+T3 :: GOTO 1230
1340 TT=TT+T4 :: GOTO 1240
1350 TT=TT+T5 :: GOTO 1250
1360 TT=TT+T6 :: GOTO 1260
1370 TT=TT+T7 :: GOTO 1500
1410 T1=T1-(T1*2):: GOTO 1310
1420 T2=T2-(T2*2):: GOTO 1320
1430 T3=T3-(T3*2):: GOTO 1330
1440 T4=T4-(T4*2):: GOTO 1340
1450 T5=T5-(T5*2):: GOTO 1350
1460 T6=T6-(T6*2):: GOTO 1360
1470 T7=T7-(T7*2):: GOTO 1370
```

(continued)

```

1500 GOSUB 180 :: GOTO 2200
1510 PRINT #1:T$;TAB(15);T1;TAB(23);T2;TAB(31);T3;TAB(39);
 T4;TAB(47);T5;TAB(55);T6;TAB(63);T7 :: PRINT #1
1520 IF T1<1 THEN E1=T1 ELSE R1=T1
1530 IF T2<1 THEN E2=T2 ELSE R2=T2
1540 IF T3<1 THEN E3=T3 ELSE R3=T3
1550 IF T4<1 THEN E4=T4 ELSE R4=T4
1560 IF T5<1 THEN E5=T5 ELSE R5=T5
1570 IF T6<1 THEN E6=T6 ELSE R6=T6
1580 IF T7<1 THEN E7=T7 ELSE R7=T7
1590 EE=E1+E2+E3+E4+E5+E6+E7 :: EE=EE-(EE+EE):: RR=R1+R2+
 R3+R4+R5+R6+R7
1600 PRINT #1:E$;EE
1601 PRINT #1 :: PRINT #1:R$;RR
1602 PRINT #1 :: PRINT #1:GT$;TT
1605 CLOSE #1
1610 END
2000 OPEN #2:"DSK1.BALANCE"
2010 IF EOF(2) THEN 2030
2020 INPUT #2:T1,T2,T3,T4,T5,T6,T7
2030 CLOSE #2
2040 GOTO 2400
2100 T1=C1 :: T2=C2 :: T3=C3 :: T4=C4 :: T5=C5 :: T6=C6 ::
 T7=C7
2110 OPEN #2:"DSK1.BALANCE"
2120 PRINT #2:T1,T2,T3,T4,T5,T6,T7
2130 CLOSE #2
2140 GOTO 1000
2200 OPEN #2:"DSK1.BALANCE"
2210 PRINT #2:T1,T2,T3,T4,T5,T6,T7
2220 CLOSE #2
2230 GOTO 1510
2300 OPEN #3:"DSK1.TOTALS"
2310 PRINT #3:CD1$,CD2$,CD3$,CD4$,CD5$,CD6$,CD7$
2320 CLOSE #3
2330 GOTO 370
2400 OPEN #3:"DSK1.TOTALS"
2410 IF EOF(3) THEN 2430
2420 INPUT #3:CD1$,CD2$,CD3$,CD4$,CD5$,CD6$,CD7$
2430 CLOSE #3
2440 C1=T1 :: C2=T2 :: C3=T3 :: C4=T4 :: C5=T5 :: C6=T6 ::
 C7=T7
2450 GOTO 1000

```

---

| DESCRIPTION     | Col. 1 | Col. 2 | Col. 3 | Col. 4 | Col. 5  | Col. 6  | Col. 7  |
|-----------------|--------|--------|--------|--------|---------|---------|---------|
|                 | 428.76 | 44.82  | 3.05   | 11.75  | 23.51   | 29.07   | 17.15   |
| JOB A           | 97.23  | 85.39  | 14.88  | 17.34  | 76.21   | 61.99   | 50.03   |
| JOB B           | 93.11  | 68.94  | 43.65  | 0      | 19.54   | 66.12   | 34.93   |
| Total= \$       | 619.1  | 199.15 | -61.58 | -29.09 | -119.26 | -157.18 | -102.11 |
| Expenses= \$    | 469.22 |        |        |        |         |         |         |
| Receipts= \$    | 818.25 |        |        |        |         |         |         |
| Grand total= \$ | 349.03 |        |        |        |         |         |         |

Figure 13-6. A sample printout produced by the ledger sheet program.

Line by line:

**Lines 100–170** are the usual.

**Line 190** prints the purpose of the program and reminds you that the printer must be ready.

**Line 210** sends the computer to line 540 to display the menu, giving you the choice of entering a new set of balances, such as would have to be done during the first run, or to use the previous balances.

**Lines 230–430** are used only if this is the first run or if you have decided to enter a new set of balances and/or redesign the column layout.

**Lines 230 and 240** describe the layout and ask you to decide which columns to use for what.

**Line 250** assigns some strings to string variables.

**Lines 260–330** asks you to enter either C (credit) or D (debit) for each of the seven columns. At the end of line 330 the computer is sent to line 1240, where the data are transferred to a separate data file.

**Line 340** assigns a group of strings to string variables.

**Lines 350–430** ask you to key in balance figures for each of the seven columns. At the end of line 430 the computer is sent to line 1150 to store the entered data in another separate data file.

**Lines 440–530** are used on subsequent runs, asking you to type in the job descriptions and the associated data to be added to or deducted from the previous balances.

**Lines 540–580** display the menu and send the computer to one of two line numbers depending on your reply. If it goes to line 1100, the previously created data files are accessed and the column categories and last balances are transferred into the computer RAM.

**Line 590** accesses the line printer.

**Line 600** prints the first line of the printout, identifying the eight columns.

**Line 620** prints the previous balances or the set of new balances you have typed in.

**Line 640** sends the computer to line 440 to permit you to enter new data.

**Line 660** prints those data.

---



**Line 670** increases the value assigned to A by 1, because a different set of calculations is used if more than one set of entries is made.

**Line 680** asks if you want to make another entry.

**Line 690** checks the value of A and tells the computer to skip the next line if it is 1.

**Line 700** performs a series of calculations, adding entries to previous balances.

**Line 710** performs the same task as line 690.

**Line 720** performs the same task as line 700.

**Line 730** checks for your answer with reference to additional entries, sending the computer to one of two line numbers to either enter new data or to go on to the final set of calculations.

**Lines 740-800** check on the column designations for each column and send the computer to one of two line numbers in each case.

**Lines 820-870** are used to add data in the credit columns.

**Lines 880-940** are used to compute the debit columns.

**Line 950** sends the computer to line 1200 to store the new balances in the data file.

**Line 960** prints the new balances for each column.

**Lines 970-1040** calculate the total expenses, total receipts, and the grand total.

**Lines 1050-1080** cause the final data to be sent to the line printer.

**Lines 1100-1330** access the two data files (#2 and #3) to either record or retrieve data.

In practice, you might want to add another print line to identify the individual columns not only by number, but also by the type of data for which they are to be used.

## **ANALYZING ADVERTISING COST VERSUS RETURN**

The next two programs deal with different aspects of advertising. The first (Advertising Cost Program) is designed to determine the cost per prospect and the return per advertising dollar resulting

---

## ADVERTISING COST PROGRAM

A program that analyzes cost and return data for seven advertising media.

---

```

100 REM ADVERTISING
110 REM TI EXTENDED BASIC
120 CP$="Cost per prospect= $"
130 SD$="Sales per adv.dollar=$"
140 RPS$="# of resultant prospects= "
150 RSS$="$ resultant sales= $"
160 ZERO$="For 0 prospects/sales type 1"
170 SP$="$ spent on "
180 RP$="# of resultant prospects?"
190 RS$="$ resultant sales?"
200 CALL CLEAR
210 GOSUB 1720
220 PRINT TAB(5);"To advertise or" :: PRINT
230 PRINT TAB(5);"not to advertise?" :: GOSUB 1720
240 GOSUB 1730 :: GOSUB 1740
250 CALL CLEAR
260 PRINT "You can select any or all" :: PRINT
270 PRINT "of the following categories:" :: GOSUB 1720
280 PRINT 1;" NEWSPAPERS"
290 PRINT 2;" MAGAZINES"
300 PRINT 3;" YELLOW PAGES"
310 PRINT 4;" RADIO"
320 PRINT 5;" TELEVISION"
330 PRINT 6;" DIRECT MAIL"
340 PRINT 7;" MISCELLANEOUS" :: GOSUB 1720
350 PRINT 8;" See totals" :: GOSUB 1720
360 PRINT 9;" Exit the program" :: GOSUB 1720
370 INPUT "Which? ":WHICH
380 CALL CLEAR :: ON WHICH GOTO 390,500,610,720,830,940,
 1050,1160,1750
390 PRINT ZERO$:: GOSUB 1720
400 PRINT SP$;"newspapers?" :: PRINT
410 INPUT N :: PRINT
420 PRINT RP$:: PRINT
430 INPUT NP :: PRINT
440 PRINT RS$:: PRINT
450 INPUT NS :: GOSUB 1720
460 PN=N/NP :: SN=NS/N
470 PN=INT(PN*100+.5)/100 :: SN=INT(SN*100+.5)/100
480 PRINT :: PRINT CP$;PN :: PRINT
490 PRINT SD$;SN :: GOSUB 1720 :: GOSUB 1740 :: GOTO 250
500 PRINT ZERO$:: GOSUB 1720
510 PRINT SP$;"magazines?" :: PRINT
520 INPUT M :: PRINT
530 PRINT RP$:: PRINT
540 INPUT MP :: PRINT
550 PRINT RS$:: PRINT

```

(continued)

---

```
560 INPUT MS :: GOSUB 1720
570 PM=M/MP :: SM=MS/M
580 PM=INT(PM*100+.5)/100 :: SM=INT(SM*100+.5)/100
590 PRINT :: PRINT CP$;PM :: PRINT
600 PRINT SD$;SM :: GOSUB 1720 :: GOSUB 1740 :: GOTO 250
610 PRINT ZERO$:: GOSUB 1720
620 PRINT SP$;"Yellow Pages?" :: PRINT
630 INPUT Y :: PRINT
640 PRINT RP$:: PRINT
650 INPUT YP :: PRINT
660 PRINT RS$:: PRINT
670 INPUT YS :: GOSUB 1720
680 PY=Y/YP :: SY=YS/Y
690 PY=INT(PY*100+.5)/100 :: SY=INT(SY*100+.5)/100
700 PRINT :: PRINT CP$;PY :: PRINT
710 PRINT SD$;SY :: GOSUB 1720 :: GOSUB 1740 :: GOTO 250
720 PRINT ZERO$:: GOSUB 1720
730 PRINT SP$;"radio?" :: PRINT
740 INPUT R :: PRINT
750 PRINT RP$:: PRINT
760 INPUT RP :: PRINT
770 PRINT RS$:: PRINT
780 INPUT RS :: GOSUB 1720
790 PR=R/RP :: SR=RS/R
800 PR=INT(PR*100+.5)/100 :: SR=INT(SR*100+.5)/100
810 PRINT :: PRINT CP$;PR :: PRINT
820 PRINT SD$;SR :: GOSUB 1720 :: GOSUB 1740 :: GOTO 250
830 PRINT ZERO$:: GOSUB 1720
840 PRINT SP$;"television?" :: PRINT
850 INPUT T :: PRINT
860 PRINT RP$:: PRINT
870 INPUT TP :: PRINT
880 PRINT RS$:: PRINT
890 INPUT TS :: GOSUB 1720
900 PT=T/TP :: ST=TS/T
910 PT=INT(PT*100+.5)/100 :: ST=INT(ST*100+.5)/100
920 PRINT :: PRINT CP$;PT :: PRINT
930 PRINT SD$;ST :: GOSUB 1720 :: GOSUB 1740 :: GOTO 250
940 PRINT ZERO$:: GOSUB 1720
950 PRINT SP$;"direct mail?" :: PRINT
960 INPUT D :: PRINT
970 PRINT RP$:: PRINT
980 INPUT DP :: PRINT
990 PRINT RS$:: PRINT
1000 INPUT DS :: GOSUB 1720
1010 PD=D/DP :: SD=DS/D
1020 PD=INT(PD*100+.5)/100 :: SD=INT(SD*100+.5)/100
1030 PRINT :: PRINT CP$;PD :: PRINT
1040 PRINT SD$;SD :: GOSUB 1720 :: GOSUB 1740 :: GOTO 250
1050 PRINT ZERO$:: GOSUB 1720
1060 PRINT SP$;"miscellaneous?" :: PRINT
1070 INPUT I :: PRINT
1080 PRINT RP$:: PRINT
```

(continued)

---

```

1090 INPUT IP :: PRINT
1100 PRINT RS$:: PRINT
1110 INPUT IS :: GOSUB 1720
1120 II=I/IP :: SI=IS/I
1130 II=INT(II*100+.5)/100 :: SI=INT(SI*100+.5)/100
1140 PRINT :: PRINT CP$;II :: PRINT
1150 PRINT SD$;SI :: GOSUB 1720 :: GOSUB 1740 :: GOTO 250
1160 TT=N+M+Y+R+T+D+I
1170 PP=NP+MP+YP+RP+TP+DP+IP
1180 SS=NS+MS+YS+RS+TS+DS+IS
1190 PRINT "Total cost= $";TT :: PRINT
1200 PRINT RP$;PP :: PRINT
1210 PRINT RS$;SS :: GOSUB 1720
1220 QQ=PP :: WW=SS
1230 PP=TT/PP :: SS=SS/TT
1240 PP=INT(PP*100+.5)/100 :: SS=INT(SS*100+.5)/100
1250 PRINT CP$;PP :: PRINT
1260 PRINT SD$;SS :: GOSUB 1720
1270 INPUT "Printout? (Y/N) ":YN$
1280 IF YN$="Y" THEN 1290 ELSE 1750
1290 OPEN #1:"RS232"
1300 PRINT #1:SP$;"Newspapers= $";N
1310 PRINT #1:RPS$;NP
1320 PRINT #1:RSS$;NS
1330 PRINT #1:CP$;PN
1340 PRINT #1:SD$;SN :: GOSUB 1710
1350 PRINT #1:SP$;"Magazines= $";M
1360 PRINT #1:RPS$;MP
1370 PRINT #1:RSS$;MS
1380 PRINT #1:CP$;PM
1390 PRINT #1:SD$;SM :: GOSUB 1710
1400 PRINT #1:SP$;"Yellow pages= $";Y
1410 PRINT #1:RPS$;YP
1420 PRINT #1:RSS$;YS
1430 PRINT #1:CP$;PY
1440 PRINT #1:SD$;SY :: GOSUB 1710
1450 PRINT #1:SP$;"Radio= $";R
1460 PRINT #1:RPS$;RP
1470 PRINT #1:RSS$;RS
1480 PRINT #1:CP$;PR
1490 PRINT #1:SD$;SR :: GOSUB 1710
1500 PRINT #1:SP$;"Television= $";T
1510 PRINT #1:RPS$;TP
1520 PRINT #1:RSS$;TS
1530 PRINT #1:CP$;PT
1540 PRINT #1:SD$;ST :: GOSUB 1710
1550 PRINT #1:SP$;"Direct mail= $";D
1560 PRINT #1:RPS$;DP
1570 PRINT #1:RSS$;DS

```

(continued)

---

```
1580 PRINT #1:CP$;PD
1590 PRINT #1:SD$;SD :: GOSUB 1710
1600 PRINT #1:SP$;"Miscellaneous=$";I
1610 PRINT #1:RPS$;IP
1620 PRINT #1:RSS$;IS
1630 PRINT #1:CP$;II
1640 PRINT #1:SD$;SI :: GOSUB 1710 :: GOSUB 1710
1650 PRINT #1:"Total cost= $";TT
1660 PRINT #1:RPS$;QQ
1670 PRINT #1:RSS$;WW
1680 PRINT #1:CP$;PP
1690 PRINT #1:SD$;SS :: GOSUB 1710
1700 CLOSE #1 :: GOTO 1750
1710 PRINT #1:"-----" :: RETURN
1720 PRINT "-----" :: RETURN
1730 FOR X=1 TO 8 :: PRINT :: NEXT X :: RETURN
1740 INPUT "Press >ENTER< ":E$:: RETURN
1750 GOSUB 1720 :: PRINT TAB(12);"End." :: GOSUB 1720 ::
 GOSUB 1730 :: END
```

---

from advertising in seven categories of media. The program assumes that you have a means of determining which advertising medium is responsible for a given prospect or sale. When it is activated, it displays:

To advertise or  
not to advertise?

-----  
You can select any or all  
of the following categories:

-----  
1 NEWSPAPER  
2 MAGAZINES  
3 YELLOW PAGES  
4 RADIO  
5 TELEVISION  
6 DIRECT MAIL  
7 MISCELLANEOUS

-----  
8 See totals

-----  
9 Exit the program  
-----

Which?

---

After a category has been selected, it asks you to key in the amount spent for advertising in that category, the number of prospects, and the amount of resulting sales. It then displays the cost per prospect and the sales per advertising dollar. When all data have been entered, it displays the total amounts and asks if you want the results printed. Figure 13-7 shows a sample printout from this program.

```

$ spent on Newspapers= $ 1438.22
of resultant prospects= 63
$ resultant sales= $ 3784.66
Cost per prospect= $ 22.83
Sales per adv.dollar=$ 2.63

$ spent on Magazines= $ 4285.76
of resultant prospects= 13
$ resultant sales= $ 12870.55
Cost per prospect= $ 329.67
Sales per adv.dollar=$ 3

$ spent on Yellow pages= $ 175
of resultant prospects= 22
$ resultant sales= $ 438.75
Cost per prospect= $ 7.95
Sales per adv.dollar=$ 2.51

$ spent on Radio= $ 2000
of resultant prospects= 21
$ resultant sales= $ 1547.86
Cost per prospect= $ 95.24
Sales per adv.dollar=$.77

$ spent on Television= $ 3279.45
of resultant prospects= 176
$ resultant sales= $ 6422.11
Cost per prospect= $ 18.63
Sales per adv.dollar=$ 1.96

$ spent on Direct mail= $ 175.22
of resultant prospects= 5
$ resultant sales= $ 288.53
Cost per prospect= $ 35.04
Sales per adv.dollar=$ 1.65

```

(continued)

---

```
$ spent on Miscellaneous=$ 50
of resultant prospects= 2
$ resultant sales= $ 150
Cost per prospect= $ 25
Sales per adv.dollar=$ 3

Total cost= $ 11403.65
of resultant prospects= 302
$ resultant sales= $ 25502.46
Cost per prospect= $ 37.76
Sales per adv.dollar=$ 2.24

```

Figure 13-7. A sample printout from the advertising program.

Line by line:

**Lines 100 and 110** are REMarks.

**Lines 120–190** assign a number of frequently used strings to string variables.

**Lines 200–370** display the program title and the menu.

**Line 380** sends the computer to one of nine line numbers, depending on your selection.

**Lines 390–450** ask you to key in the data for newspaper advertising.

**Line 460** performs the calculations.

**Lines 470–490** display the results and then send the computer back to line 250 to display the menu for the next selection.

**Lines 500–1150** are repetitions of the above for the other six categories.

**Lines 1160–1180** perform the calculations that produce the final totals.

**Lines 1190–1260** display the final totals.

**Line 1270** asks if you want the data sent to the line printer.

**Line 1280** sends the computer to one of two line numbers, depending on your answer.

**Lines 1290–1700** send the data for all seven categories and the totals to the line printer.

**Lines 1710–1740** are four frequently used subroutines.

**Line 1750** is the END line.

---

## ANALYZING DIRECT MAIL ADVERTISING

The second advertising program analyzes data associated with direct mail. When provided with the necessary input data, it determines the number of sales that must be made to break even, the percent return needed to break even, and the profit or loss based on actual sales. The program is titled Direct Mail Cost Program.

### DIRECT MAIL COST PROGRAM

This program analyzes direct mail advertising data.

---

```

100 REM DIRECT MAIL
110 REM TI EXTENDED BASIC
120 CALL CLEAR
130 GOSUB 690
140 PRINT "This program determines the cost factors
 associated with direct mail advertising" :: GOSUB 690
150 GOSUB 700 :: GOSUB 710
160 CALL CLEAR
170 PRINT "Cost of mailing list?" :: INPUT "$":ML
180 PRINT "Cost of mailing piece?" :: INPUT "$":MP
190 PRINT "Multiple use OK? (Y/N)" :: INPUT YN$
200 IF YN$="N" THEN 240
210 PRINT "Number of mailings, (list)?" :: INPUT NM
220 PRINT "Number of uses (mail piece)?" :: INPUT NU
230 ML=ML/NM :: MP=MP/NU
240 PRINT "Number of names on list?" :: INPUT A
250 AD=A
260 PRINT "Postage (each piece)?" :: INPUT "$":A
270 PRINT "List price per item?" :: INPUT "$":P
280 PRINT "Mfg. cost per item?" :: INPUT "$":I
290 CALL CLEAR
300 PP=P-I :: MU=(PP/I)*100 :: MU=INT(MU*10+.5)/10
310 GOSUB 690
320 PRINT "Profit per item= $";PP :: PRINT
330 PRINT "Markup= ";MU;"%" :: GOSUB 690 ::
 GOSUB 700 :: GOSUB 710
340 CALL CLEAR
350 PRINT "Mailing/handling per item?" :: INPUT "$":MH
360 PRINT "Mailing/handling charge?" :: INPUT "$":MHC
370 MH=MH-MHC
380 AB=(A*M)+ML+MP :: AA=AB/(PP-MH) :: Q=(AA/AD)*100
390 AA=INT(AA*10+.5)/10 :: Q=INT(Q*10+.5)/10

```

(continued)

---



```
400 CALL CLEAR
410 PRINT "No. sales to break even=";AA :: PRINT
420 PRINT "% return to break even=";Q;"%" :: GOSUB 700
430 PRINT "Number of sales?" :: INPUT NS
440 EX=ML+(A*M)+MP+(NS*MH):: EX=INT(EX*100+.5)/100
450 CALL CLEAR
460 PRINT "Total mailing expenses= " :: PRINT :: PRINT TAB
(15);"$";EX :: GOSUB 6
470 NR=(NS*PP)-EX
480 PRINT "Net profit/loss=" :: PRINT :: PRINT TAB(15);"$"
;NR
490 GOSUB 690 :: GOSUB 700 :: GOSUB 710
500 CALL CLEAR
510 INPUT "Printout? (Y/N) ":NY$
520 IF NY$="Y" THEN 560
530 CALL CLEAR
540 INPUT "Another run? (Y/N) ":YY$
550 IF YY$="N" THEN 720 ELSE 160
560 OPEN #1:"RS232"
570 PRINT #1:"Item price= $";P
580 PRINT #1:"Item cost= $";I
590 PRINT #1:"Item profit= $";PP
600 PRINT #1:"Markup= ";MU;"%"
610 PRINT #1:"Break-even sales= ";AA
620 PRINT #1:"Break-even % return ";Q;"%"
630 PRINT #1:"-----"
640 PRINT #1:"Sales made= ";NS
650 PRINT #1:"Mail/handling= $";EX
660 PRINT #1:"Net profit/loss= $";NR
670 CLOSE #1
680 GOTO 720
690 PRINT "-----" :: RETURN
700 FOR X=1 TO 8 :: PRINT :: NEXT X :: RETURN
710 INPUT "Press >ENTER< ":E$:: RETURN
720 CALL CLEAR :: GOSUB 690 :: PRINT TAB(12);"End." ::
GOSUB 690 :: GOSUB 700 :: END
```

---

When it is run it first asks you to type in the cost of the mailing list and the cost of producing the mailing piece including envelopes. It then wants to know if the mailing list can be used for more than one mailing and if you anticipate using the mailing piece for more than one mailing. If the answer to either question is yes, it asks the number of anticipated uses of the list and of the mailing piece. Next it needs to know the number of names on the list. After that you're asked to enter the postage for each individual mailing piece, the list price of the item being advertised, and the manufacturing cost of that item. It then displays the profit per item and the markup

---

percentage. Next you must enter the mailing and handling cost to you for shipping the item, and the mailing and handling charge that will be added to the list price.

Once all these data have been entered, it calculates the number of sales and the percent return needed to break even. Then you're asked to key in the actual number of resulting sales. The program then calculates the total mailing and shipping expenses and the profit or loss based on actual sales. Finally, you're asked if you want the resulting data printed. Figure 13-8 shows a sample printout.

```

Item price= $ 24.95
Item cost= $ 13.11
Item profit= $ 11.84
Markup= 90.3 %
Break-even sales= 42.9
Break-even % return 4.3 %

Sales made= 57
Mail/handling= $ 492.08
Net profit/loss= $ 182.8

```

Figure 13-8. A sample printout from the direct mail program.

Line by line:

**Lines 100 and 110** are REMarks.

**Lines 120-160** display the purpose of the program.

**Lines 170-190** display the first three questions.

**Line 200** tells the computer to skip the next three lines if multiple use of the list or mailing piece is not anticipated.

**Lines 210-230** are used if multiple use is anticipated, performing the necessary calculations in line 230.

**Lines 240-310** display an additional set of questions. Line 300 then performs a series of calculations to produce interim totals.

**Lines 320 and 330** display those interim totals.

**Lines 350 and 360** ask two more questions.

---

**Lines 370–390** perform the calculations needed to determine the breakeven figures.

**Lines 410 and 420** display the breakeven figures.

**Line 430** asks you to key in the number of actual sales.

**Lines 440 and 470** perform another series of calculations.

**Lines 460 and 480** display the results of those calculations.

**Lines 510 and 520** ask if you want the results printed, and line 540 asks if you want to run the program again with different variables.

**Lines 560–670** produce the printout.

**Lines 690–710** are three subroutines, and line 720 is the END line.

## INTRODUCING A NEW PRODUCT

Developing and introducing a new product is another task in which a reasonably comprehensive computer program can determine whether or not there is a chance of making a profit. The program, New Product Projection reproduced here should work for most types of products, though not all of the items being considered may be applicable in each case, and other considerations may have to be added.

### NEW PRODUCT PROJECTION PROGRAM

This program analyzes the data involved in the introduction and marketing of a new product.

---

```
100 REM INTRODUCING A NEW PRODUCT
110 REM TI EXTENDED BASIC
120 CALL CLEAR
130 GOTO 170
140 PRINT "-----" :: RETURN
150 FOR X=1 TO 8 :: PRINT :: NEXT X :: RETURN
160 INPUT "Press >ENTER< ":E$:: RETURN
170 I$="Fee paid inventor"
180 M$="Cost of redesign"
190 SM$="Cost,machinery"
200 T$="Months to prepare"
210 RM$="Raw material/unit"
220 TU$="Hours to make 1 unit"
```

(continued)

---

```

230 L$="Labor $ (hour)"
240 OM$="Overhead/month"
250 AB$="Advertising/month"
260 PB$="Publicity/month"
270 SC$="Shipping cost/unit"
280 SP$="Selling price/unit"
290 AS$="Anticipated sales, "
300 S1$="year 1 " :: S2$="year 2 " :: S3$="year 3 " :: S4$
 ="year 4 " :: S5$="year 5 "
310 CU$="Cost per unit "
320 PU$="Profit per unit "
330 TP$="Total profit "
340 IC$="Cost amortized "
350 GS$="Gross sales: "
360 CM$="Mfr.cost "
370 RP$="Royalty % rate "
380 PL$="Profit/loss "
390 TD$="Total to date "
400 LL$="-----"
 "
410 GOSUB 140
420 PRINT "This program calculates the costs and profit/
 loss data associated with the develop-ment of a new
 product."
430 GOSUB 140 :: GOSUB 150 :: INPUT "Printout? ":YY$::
 PRINT :: PRINT
440 CALL CLEAR
450 PRINT "Product description" :: INPUT DP$:: CALL CLEAR
460 PRINT I$:: INPUT "$":I
470 PRINT M$:: INPUT "$":M
480 PRINT SM$:: INPUT "$":SM
490 PRINT T$:: INPUT T
500 GOSUB 140
510 PRINT RM$:: INPUT "$":RM
520 PRINT TU$:: INPUT TU
530 PRINT L$:: INPUT "$":L
540 PRINT OM$:: INPUT "$":OM
550 GOSUB 140
560 PRINT AB$:: INPUT "$":AB
570 PRINT PB$:: INPUT "$":PB
580 PRINT SC$:: INPUT "$":SC
590 PRINT SP$:: INPUT "$":SP
600 GOSUB 140
610 PRINT RP$:: INPUT "%":RP
620 GOSUB 140
630 PRINT AS$;S1$:: INPUT S1
640 PRINT AS$;S2$:: INPUT S2
650 PRINT AS$;S3$:: INPUT S3
660 PRINT AS$;S4$:: INPUT S4
670 PRINT AS$;S5$:: INPUT S5
680 CALL CLEAR
690 IC=I+M+SM+(T*OM)

```

(continued)

---

```
700 CU=(IC/S1)+RM+(TU*L)+((OM*12)/S1)+(((AB+PB)*12)/S1)+SC
710 CU=CU-(SP*(RP/100)):: PU=SP-CU
720 GS1=SP*S1 :: CM1=CU*S1 :: PL1=GS1-IC-CM1
730 GS2=SP*S2 :: CM2=CU*S2 :: PL2=GS2-CM2 :: TTD2=PL1+PL2
740 GS3=SP*S3 :: CM3=CU*S3 :: PL3=GS3-CM3 :: TTD3=TTD2+PL3
750 GS4=SP*S4 :: CM4=CU*S4 :: PL4=GS4-CM4 :: TTD4=TTD3+PL4
760 GS5=SP*S5 :: CM5=CU*S5 :: PL5=GS5-CM5 :: TTD5=TTD4+PL5
770 PRINT DP$:: GOSUB 140
780 PRINT I$;" "$:I
790 PRINT M$;" "$:M
800 PRINT SM$;" "$:SM
810 PRINT T$;" "$:T;" months" :: GOSUB 140
820 PRINT RM$;" "$:RM
830 PRINT TU$;" "$:TU;" hours"
840 PRINT L$;" "$:L
850 PRINT OM$;" "$:OM :: GOSUB 140
860 PRINT AB$;" "$:AB
870 PRINT PB$;" "$:PB
880 PRINT SC$;" "$:SC
890 PRINT SP$;" "$:SP :: GOSUB 140 :: GOSUB 160 :: CALL
 CLEAR
900 PRINT RP$:RP;"%" :: GOSUB 140
910 PRINT AS$;" "$:S1$;" "$:S1
920 PRINT AS$;" "$:S2$;" "$:S2
930 PRINT AS$;" "$:S3$;" "$:S3
940 PRINT AS$;" "$:S4$;" "$:S4
950 PRINT AS$;" "$:S5$;" "$:S5 :: GOSUB 140 :: GOSUB 160 ::
 CALL CLEAR
960 PRINT IC$;" "$:S1$;" "$:IC :: GOSUB 140
970 CU=INT(CU*100+.5)/100 :: PU=INT(PU*100+.5)/100
980 PRINT GS$;" "$:S1$;" "$:GS1
990 PRINT CM$;" "$:S1$;" "$:CM1
1000 PRINT PL$;" "$:S1$;" "$:PL1 :: GOSUB 140 :: GOSUB 160
 :: CALL CLEAR
1010 PRINT GS$;" "$:S2$;" "$:GS2
1020 PRINT CM$;" "$:S2$;" "$:CM2
1030 PRINT PL$;" "$:S2$;" "$:PL2
1040 PRINT TD$;" "$:S2$;" "$:TTD2 :: GOSUB 140 :: GOSUB 160
 :: CALL CLEAR
1050 PRINT GS$;" "$:S3$;" "$:GS3
1060 PRINT CM$;" "$:S3$;" "$:CM3
1070 PRINT PL$;" "$:S3$;" "$:PL3
1080 PRINT TD$;" "$:S3$;" "$:TTD3 :: GOSUB 140 :: GOSUB 160
 :: CALL CLEAR
1090 PRINT GS$;" "$:S4$;" "$:GS4
1100 PRINT CM$;" "$:S4$;" "$:CM4
1110 PRINT PL$;" "$:S4$;" "$:PL4
1120 PRINT TD$;" "$:S4$;" "$:TTD4 :: GOSUB 140 :: GOSUB 160
 :: CALL CLEAR
1130 PRINT GS$;" "$:S5$;" "$:GS5
1140 PRINT CM$;" "$:S5$;" "$:CM5
1150 PRINT PL$;" "$:S5$;" "$:PL5
```

(continued)

```

1160 PRINT TD$;" ";S5$;" $";TTD5 :: GOSUB 140 :: GOSUB 160
1170 IF YY$="N" THEN 1630 ELSE 1180
1180 OPEN #1:"RS232"
1190 PRINT #1:DP$:: PRINT #1:LL$
1200 PRINT #1:I$;" $";I
1210 PRINT #1:M$;" $";M
1220 PRINT #1:SM$;" $";SM
1230 PRINT #1:T$;" ";T;" months"
1240 PRINT #1:LL$
1250 PRINT #1:RM$;" $";RM
1260 PRINT #1:TU$;" ";TU;" hours"
1270 PRINT #1:L$;" $";L
1280 PRINT #1:OM$;" $";OM
1290 PRINT #1:LL$
1300 PRINT #1:AB$;" $";AB
1310 PRINT #1:PB$;" $";PB
1320 PRINT #1:SC$;" $";SC
1330 PRINT #1:SP$;" $";SP
1340 PRINT #1:LL$
1350 PRINT #1:AS$;" ";S1$;" $";S1
1360 PRINT #1:AS$;" ";S2$;" $";S2
1370 PRINT #1:AS$;" ";S3$;" $";S3
1380 PRINT #1:AS$;" ";S4$;" $";S4
1390 PRINT #1:AS$;" ";S5$;" $";S5
1400 PRINT #1:LL$:: PRINT #1:LL$
1410 PRINT #1:IC$;" ";S1$;" $";IC
1420 PRINT #1:LL$
1430 PRINT #1:GS$;" ";S1$;" $";GS1
1440 PRINT #1:CM$;" ";S1$;" $";CM1
1450 PRINT #1:PL$;" ";S1$;" $";PL1 :: PRINT #1:LL$
1460 PRINT #1:GS$;" ";S2$;" $";GS2
1470 PRINT #1:CM$;" ";S2$;" $";CM2
1480 PRINT #1:PL$;" ";S2$;" $";PL2
1490 PRINT #1:TD$;" ";S2$;" $";TTD2 :: PRINT #1:LL$
1500 PRINT #1:GS$;" ";S3$;" $";GS3
1510 PRINT #1:CM$;" ";S3$;" $";CM3
1520 PRINT #1:PL$;" ";S3$;" $";PL3
1530 PRINT #1:TD$;" ";S3$;" $";TTD3 :: PRINT #1:LL$
1540 PRINT #1:GS$;" ";S4$;" $";GS4
1550 PRINT #1:CM$;" ";S4$;" $";CM4
1560 PRINT #1:PL$;" ";S4$;" $";PL4
1570 PRINT #1:TD$;" ";S4$;" $";TTD4 :: PRINT #1:LL$
1580 PRINT #1:GS$;" ";S5$;" $";GS5
1590 PRINT #1:CM$;" ";S5$;" $";CM5
1600 PRINT #1:PL$;" ";S5$;" $";PL5
1610 PRINT #1:TD$;" ";S5$;" $";TTD5 :: PRINT #1:LL$
1620 CLOSE #1
1630 CALL CLEAR :: PRINT TAB(12);"End." :: GOSUB 140 ::
 GOSUB 150 :: END

```

---

During execution the program presents you with a checklist of more than a dozen subjects for which data may have to be entered in order to produce the desired information:

Fee paid to inventor?  
 Cost of redesign for production?  
 Cost of special machinery?  
 Months to prepare for production?  
 Raw material cost per unit?  
 Hours to manufacture one unit?  
 Labor cost per hour?  
 Overhead per month?  
 Advertising cost per month?  
 Publicity cost per month?  
 Shipping cost per unit?  
 Selling price per unit?  
 Royalty percentage paid inventory?  
 Anticipated sales for each of the first  
 five years?

Given these data, the program shows the gross sales, manufacturing cost, profit or loss, and accumulated profit or loss for each of 5 years, the assumption being that the initial expense of acquisition and preparation is amortized during the first year. A sample printout appears in Figure 13-9.

PRODUCT A/10

-----  
 Fee paid inventor \$ 500  
 Cost of redesign \$ 700  
 Cost, machinery \$ 2250  
 Months to prepare 6 months  
 -----  
 Raw material/unit \$ 25.55  
 Hours to make 1 unit 3.5 hours  
 Labor \$ (hour) \$ 18  
 Overhead/month \$ 3500  
 -----

(continued)

```

Advertising/month $ 1500
Publicity/month $ 500
Shipping cost/unit $ 5
Selling price/unit $ 175

Anticipated sales, year 1 $ 1000
Anticipated sales, year 2 $ 2000
Anticipated sales, year 3 $ 2500
Anticipated sales, year 4 $ 2500
Anticipated sales, year 5 $ 2000

Cost amortized year 1 $ 24450

Gross sales: year 1 $ 175000
Mfr.cost year 1 $ 166500
Profit/loss year 1 $ -15950

Gross sales: year 2 $ 350000
Mfr.cost year 2 $ 333000
Profit/loss year 2 $ 17000
Total to date year 2 $ 1050

Gross sales: year 3 $ 437500
Mfr.cost year 3 $ 416250
Profit/loss year 3 $ 21250
Total to date year 3 $ 22300

Gross sales: year 4 $ 437500
Mfr.cost year 4 $ 416250
Profit/loss year 4 $ 21250
Total to date year 4 $ 43550

Gross sales: year 5 $ 350000
Mfr.cost year 5 $ 333000
Profit/loss year 5 $ 17000
Total to date year 5 $ 60550

```

Figure 13-9. A printout showing new product introduction data.

Line by line:

**Lines 100 and 110** are REMarks.

**Line 130** causes the computer to skip the usual three subroutines.

**Lines 170–400** assign a series of strings to string variables.

**Lines 410–440** place the purpose of the program into display and ask if you want the data sent to the line printer.

---



**Lines 450–670** are the section of the program that asks you to key in the variable data.

**Line 690** calculates the cost of acquisition and preparation.

**Lines 700 and 710** calculate the cost of manufacturing one unit, assigning it to the numeric variable CU, and the profit margin per unit, assigning it to the variable PU.

**Lines 720–760** calculate the gross sales, the total manufacturing cost, the profit or loss, and the total to date for each of 5 years.

**Lines 770–950** once more display your input data to make sure there are no errors.

**Line 960** displays the total acquisition and preparation expense, which is to be amortized during the first year of production and sales.

**Line 970** rounds the values assigned to CU and PU to two decimal positions.

**Lines 980–1160** display the results for each of 5 years.

**Line 1170** asks if you want the data printed.

**Lines 1180–1620** send the input data as well as the results to the line printer.

**Line 1630** is the END line.

## **BUSINESS PROFIT/LOSS ANALYSIS**

The last program in this chapter deals with the costs associated with running a profitable business. It, too, starts with several lengthy checklists designed to remind you to enter all the cost items that represent every type of business expense. It then calculates the amount of gross business that must be achieved during a given period in order to break even or produce a profit. Next it uses actual sales figures to determine the true profit or loss (before taxes) for that given time period.

Included in the Business Analysis Program is a subprogram that determines the straight-line depreciation schedule for major capital expenditures, displaying the annual depreciation, the accumulated depreciation, and the remaining book value for any year within the period of time of the depreciation schedule.

---

## BUSINESS ANALYSIS PROGRAM

A program that deals with potential profit or loss from operating a business.

---

```

100 REM BUSINESS ANALYSIS
110 REM TI EXTENDED BASIC
120 CALL CLEAR
130 GOTO 170
140 PRINT "-----" :: RETURN
150 FOR X=1 TO 8 :: PRINT :: NEXT X :: RETURN
160 INPUT "Press >ENTER< ":E$:: RETURN
170 GOSUB 140
180 PRINT "This program analyzes business
profitability." :: GOSUB 140 :: GOSUB 150 :: GOSUB 160
:: CALL CLEAR
190 PRINT "The program includes a separate subprogram
that displays the straight-line depreciation
schedule for capital outlays."
200 GOSUB 140 :: GOSUB 150 :: GOSUB 160 :: CALL CLEAR
210 PRINT 1;" Main program"
220 PRINT 2;" Depreciation schedule" :: GOSUB 140 :: GOSUB
150
230 INPUT "Which? ":WHICH :: CALL CLEAR
240 ON WHICH GOTO 250,680
250 PRINT "Direct monthly overhead:" :: GOSUB 140
260 PRINT "Office rent?" :: INPUT O
270 PRINT "Electricity?" :: INPUT E
280 PRINT "Heating/airconditioning?" :: INPUT H
290 PRINT "Telephone (base rate)?" :: INPUT T
300 PRINT "Maintenance/cleaning/repair?" :: INPUT MR
310 PRINT "Equipment rentals?" :: INPUT ER
320 PRINT "Miscellaneous?" :: INPUT M
330 GOSUB 140
340 OT=E+O+H+MR+T+M+ER
350 PRINT "Total fixed costs= $";OT :: GOSUB 140 :: GOSUB
160 :: CALL CLEAR
360 PRINT "Monthly salaries:" :: GOSUB 140
370 PRINT "Owner or executive(s)?" :: INPUT PO
380 PRINT "Office manager?" :: INPUT OM
390 PRINT "Executive secretary?" :: INPUT EX
400 PRINT "Sales personnel?" :: INPUT SP
410 PRINT "Other office staff?" :: INPUT OP
420 PRINT "Part-time help?" :: INPUT PH
430 GOSUB 140 :: OS=OM+EX+SP+OP+PH+PO
440 PRINT "Monthly salaries= $";OS :: GOSUB 140 :: OS=OT+
OS
450 PRINT "Fixed overhead= $";OO
460 GOSUB 140 :: GOSUB 160 :: CALL CLEAR
470 PRINT "Variable costs per month:" :: GOSUB 140
480 PRINT "Materials?" :: INPUT MP
490 PRINT "Telephone (exc.base)?" :: INPUT LD
500 PRINT "Advertsing?" :: INPUT AC

```

(continued)

---

```
510 PRINT "Publicity?" :: INPUT PC
520 PRINT "Entertainment?" :: INPUT EN
530 PRINT "Postage/shipping?" :: INPUT PS
540 PRINT "Miscellaneous?" :: INPUT PM :: ZZ=PC+MP+AC+EN+
 LD+PS+PM :: GOSUB 140
550 PRINT "Total variable costs= $";ZZ :: GOSUB 140 :: WW=
 ZZ+OT+OS
560 PRINT "Total, this month= $";WW :: GOSUB 140 :: GOSUB
 160 :: CALL CLEAR
570 PRINT "Average markup (job/sale?" :: INPUT MU
580 PRINT "Sales commission %" :: INPUT SC
590 GS=WW+(WW*(MU/100))+(GS*(SC/100))
600 NS=GS-(GS*(MU/100)) :: NS=GS-NS :: NS=NS-(GS*(SC/100))
610 NS=INT(NS*100+.5)/100 :: GS=INT(GS*100+.5)/100 :: CALL
 CLEAR :: GOSUB 140
620 PRINT "Gross needed= $";GS :: PRINT
630 PR=GS-WW-GS*(SC/100) :: PR=INT(PR*100+.5)/100
640 PRINT "Profit= $";PR :: GOSUB 140 :: GOSUB
 150 :: GOSUB 160 :: CALL CLEAR
650 PRINT "Gross sales, this month?" :: INPUT SM :: PRINT :
 PRINT
660 MM=SM-GS :: PRINT "Above break/even= $";MM :: GOSUB
 140 :: GOSUB 150 :: GOSUB 160 :: CALL CLEAR
670 INPUT "Another run? (Y/N) " : Y$:: IF Y$="Y" THEN
 CALL CLEAR :: GOTO 210 ELSE END
680 PRINT "Depreciation schedule." :: GOSUB 140
690 PRINT "Office bldg./factory?" :: INPUT BLD
700 PRINT "Interior decoration?" :: INPUT ID
710 PRINT "Furnishings?" :: INPUT F
720 PRINT "Typewriters?" :: INPUT TW
730 PRINT "Computers?" :: INPUT CH
740 PRINT "Machinery?" :: INPUT MA
750 PRINT "Miscellaneous?" :: INPUT MI
760 IC=ID+F+TW+CH+MA+BLD+MI
770 GOSUB 140 :: PRINT "Total purchase cost= $";IC
780 PRINT :: PRINT :: PRINT "To be amortized over how
 many years?" :: INPUT Y :: GOSUB 140
790 INPUT "Useful life? " : UL
800 INPUT "Salvage value? $":SV :: PRINT :: PRINT
810 PRINT "Which year do you wish to examine?" :: PRINT
 :: INPUT YN
820 DE=(IC-SV)/UL :: DE=INT(DE*100+.5)/100
830 RD=(UL-YN)*DE :: RD=INT(RD*100+.5)/100 :: RB=RD+SV
840 CALL CLEAR :: PRINT "Annual depreciation= $";DE ::
 PRINT
850 PRINT "Remaining book value= $";RB :: PRINT
860 AC=DE*YN :: GOSUB 140 :: GOSUB 150
870 PRINT "Accumulated depreciation= $";AC :: PRINT ::
 PRINT
880 PRINT "Evaluate other years? (Y/N) " :: PRINT :: IN
 PUT YN$
890 IF YN$="Y" THEN CALL CLEAR :: GOTO 680 ELSE END .
```

---

To start with, the program displays its purpose and then asks if you want the main program or the depreciation schedule. If the main program is selected, it asks you to enter data relative to monthly office expenses, monthly salaries, and actual or anticipated variable costs for any given month. A total of 23 input categories are used to calculate the gross amount of business that must be achieved in order to produce a profit. Once that has been established, the program asks you to key in the amount of gross business that was actually done during the month in question or the anticipated gross business for that period, after which it displays the net profit or loss.

If the straight-line depreciation program is selected, it uses seven input categories to determine the total capital outlay to be depreciated. It then asks that the depreciation period be entered as well as the useful life and the salvage value at the end of the depreciation period. Based on that information it calculates the annual depreciation, the accumulated depreciation for any year, and the remaining book value for any year.

In both programs, not all the input categories must be answered if some are not applicable to your operation. Conversely, you may use some of the categories for alternative subjects. The only thing that is important is that the final totals realistically reflect your business data.

Line by line:

**Lines 100 and 110** are REMarks.

**Lines 120 and 130** clear the screen and cause the computer to skip three lines containing the usual subroutines.

**Lines 170-230** display the purpose of the program and ask which of the two programs you want to run.

**Line 240** sends the computer to one of two line numbers based on your selection.

**Lines 250-580** contain the input categories, with lines 340, 430, and 540 performing the calculations that produce the interim subtotals displayed in lines 350, 440, 550, and 560.

**Lines 590-610 and 630** perform a series of calculations, using the interim subtotals, to produce the figures that represent the needed gross and the minimum profit, if any, that would result from that figure.

**Lines 620 and 640** display the results of those calculations.

---

**Line 650** asks you to key in the actual or anticipated gross figure for a given month.

**Line 660** calculates the amount above or below the breakeven point represented by that figure. (A below-breakeven figure would be a negative figure.)

**Line 670** asks if you want to return to the menu for another run of either of the two programs, sending the computer back to line 210 or terminating the program.

**Lines 680-750** represent the input categories for the depreciation program.

**Lines 760 and 770** calculate the total and cause it to be displayed.

**Lines 780-800** ask you to key in the number of years over which the capital outlay is to be depreciated, the useful life, and the salvage value.

**Line 810** asks which year within the depreciation period you want to examine.

**Lines 820, 830, and 860** perform several calculations to determine the final data.

**Lines 840, 850 and 870** display the annual depreciation, the remaining book value, and the accumulated depreciation for the selected year.

**Lines 880 and 890** ask if you want to examine other years, and then send the computer to line 600 or cause the program to terminate.

---

---

# 14

---

## ***Advanced TI BASIC Statements***

---

---

In this chapter we'll look at some programs that illustrate the usefulness of some of the more esoteric statements available with the TI-99/4A. Some of these are rather difficult to understand at first, and using them requires a bit of practice and experience.

### ***DEFINED FUNCTIONS***

In the first program we'll be dealing with built-in as well as defined (or derived) functions. Built-in functions are statements such as INT (X), which returns the integer of the value assigned to X. Defined functions are functions that you designed and assign to a function name of your choice. The best way to illustrate these functions is to look at the Defined Function Program listing along with the explanations of what takes place in the individual lines.

The program consists of 35 subprograms dealing with arithmetic, trigonometric, and related problems. Any of these subprograms can be called up at anytime. The computer will perform the appropriate calculation and then return you to the menu to make another selection or exit the program.

## DEFINED FUNCTION PROGRAM

A program of defined functions.

```

100 CALL CLEAR
110 REM DEFINED FUNCTIONS
120 REM TI EXTENDED BASIC
130 DR=PI/180 :: RR$="The result is "
140 GOTO 1680
150 PRINT "-----" :: RETURN
160 CALL CLEAR :: RETURN
170 FOR Q=1 TO 5 :: PRINT :: NEXT Q :: RETURN
180 PRINT :: INPUT "Press >ENTER< ":E$:: RETURN
190 GOSUB 160 :: GOSUB 150 :: PRINT "End." :: GOSUB 150 ::
 GOSUB 170 :: END
200 GOSUB 150 :: GOSUB 180 :: GOTO 1680
210 INPUT "Any number? ":X :: RETURN
220 PRINT RR$;B :: GOTO 200
230 DEF ADD(A)=X+Z
240 GOSUB 210
250 INPUT "Number to add? ":Z :: GOSUB 150 :: GOSUB 170
260 B=ADD(X)
270 GOTO 220
280 DEF SBN(A)=X-Z :: GOSUB 210
290 INPUT "Number to subtract? ":Z :: GOSUB 150 :: GOSUB
 170
300 B=SBN(X)
310 GOTO 220
320 DEF MUL(A)=X*Z :: GOSUB 210
330 INPUT "Number to use to multiply? ":Z :: GOSUB 150
 :: GOSUB 170
340 B=MUL(X)
350 GOTO 220
360 DEF DIV(A)=X/Z :: GOSUB 210
370 INPUT "Number by which to divide? ":Z :: GOSUB 150
 :: GOSUB 170
380 B=DIV(X)
390 GOTO 220
400 DEF XPN(A)=X^Z :: GOSUB 210
410 INPUT "Exponent? ":Z :: GOSUB 150 :: GOSUB 170
420 B=XPN(X)
430 GOTO 220
440 DEF ROT(A)=X^(1/Z):: GOSUB 210
450 INPUT "Which root? ":Z :: GOSUB 150 :: GOSUB 170
460 B=ROT(X)
470 GOTO 220
480 INPUT "Find sine of? ":X :: GOSUB 150 :: GOSUB 170
490 X=X*DR :: B=SIN(X)
500 GOTO 220
510 INPUT "Find the arcsine of? ":X :: GOSUB 150 :: GOSUB
 170

```

(continued)

```

520 X=X*DR :: DEF ARCSIN(A)=ATN(A/SQR(-A*A+1))
530 B=ARCSIN(X)
540 GOTO 220
550 INPUT "Find the hyperbolic sine of? ":X :: GOSUB
 150 :: GOSUB 170
560 X=X*DR :: DEF SINH(A)=(EXP(A)-EXP(-A))/2
570 B=SINH(X)
580 GOTO 220
590 INPUT "Find the cosine of? ":X :: GOSUB
 150 :: GOSUB 170
600 X=X*DR :: B=COS(X)
610 GOTO 220
620 INPUT "Find the arccosine of? ":X :: GOSUB 150 ::
 GOSUB 170
630 X=X*DR :: DEF ARCCOS(A)=-ATN(A/SQR(-A*A+1))+1.5708
640 B=ARCCOS(X)
650 GOTO 220
660 INPUT "Find the hyperbolic cosine of ":X :: GOSUB
 150 :: GOSUB 170
670 X=X*DR :: DEF COSH(A)=(EXP(A)+EXP(-A))/2
680 B=COSH(X)
690 GOTO 220
700 INPUT "Find the tangent of? ":X :: GOSUB 150 :: GOSUB
 170
710 X=X*DR :: B=TAN(X)
720 GOTO 220
730 INPUT "Find the arctangent of? ":X :: GOSUB 150 ::
 GOSUB 170
740 X=X*DR :: B=ATN(X)
750 GOTO 220
760 INPUT "Find the cotangent of? ":X :: GOSUB 150 ::
 GOSUB 170
770 X=X*DR :: DEF COT(A)=1/TAN(A)
780 B=COT(X)
790 GOTO 220
800 INPUT "Find the inverse cotangent of ":X :: GOSUB
 150 :: GOSUB 170
810 X=X*DR :: DEF ARCCOT(A)=ATN(A)+1.5708
820 B=ARCCOT(X)
830 GOTO 220
840 INPUT "Find the hyperbolic tangent of ":X :: GOSUB
 150 :: GOSUB 170
850 X=X*DR :: DEF TANH(A)=EXP(-A)/(EXP(A)+EXP(-A))*2+1
860 B=TANH(X)
870 GOTO 220
880 INPUT "Find the hyperbolic cotangent of ":X :: GOSUB
 150 :: GOSUB 170
890 X=X*DR :: DEF COTH(A)=EXP(-A)/(EXP(A)-EXP(-A))*2+1
900 B=COTH(X)
910 GOTO 220

```

(continued)



```
920 INPUT "Find the secant of ":X :: GOSUB 150 :: GOSUB
 170
930 X=X*DR :: DEF SEC(A)=1/COS(A)
940 B=SEC(X)
950 GOTO 220
960 INPUT "Find the cosecant of ":X :: GOSUB 150 :: GOSUB
 170
970 X=X*DR :: DEF CSC(A)=1/SIN(A)
980 B=CSC(X)
990 GOTO 220
1000 INPUT "Find the inverse secant of? ":X :: GOSUB
 150 :: GOSUB 170
1010 X=X*DR :: DEF ARCSEC(A)=ATN(A/SQR(A*A-1))+SGN(SGN(A)
 -1)*1.5708
1020 B=ARCSEC(X)
1030 GOTO 220
1040 INPUT "Find the inverse cosecant of ":X :: GOSUB
 150 :: GOSUB 170
1050 X=X*DR :: DEF ARCCSC(A)=ATN(A/SQR(A*A-1))+(SGN(A)-1)*
 1.5708
1060 B=ARCCSC(X)
1070 GOTO 220
1080 INPUT "Find the hyperbolic secant of ":X :: GOSUB
 150 :: GOSUB 170
1090 X=X*DR :: DEF SECH(A)=2/(EXP(A)+EXP(-A))
1100 B=SECH(X)
1110 GOTO 220
1120 INPUT "Find the hyperbolic cosecant of? ":X :: GOSUB
 150 :: GOSUB 170
1130 X=X*DR :: DEF CSCH(A)=2/(EXP(A)-EXP(-A))
1140 B=CSCH(X)
1150 GOTO 220
1160 INPUT "Find the inverse hyperbolic sine of? ":X ::
 GOSUB 150 :: GOSUB 170
1170 X=X*DR :: DEF ARCSINH(A)=LOG(A+SQR(A*A+1))
1180 B=ARCSINH(X)
1190 GOTO 220
1200 INPUT "Find the inverse hyperbolic cosine of? ":X ::
 GOSUB 150 :: GOSUB 170
1210 X=X*DR :: DEF ARCCOSH(A)=LOG(A+SQR(A+A-1))
1220 B=ARCCOSH(X)
1230 GOTO 220
1240 INPUT "Find the inverse hyperbolic tangent of? ":X ::
 GOSUB 150 :: GOSUB 170
1250 X=X*DR :: DEF ARCTANH(A)=LOG((1+A)/(1-A))/2
1260 B=ARCTANH(X)
1270 GOTO 220
1280 INPUT "Find the inverse hyperbolic secant of? ":X ::
 GOSUB 150 :: GOSUB 170
1290 X=X*DR :: DEF ARCSECH(A)=LOG((SQR(-A*A+1)+1)/A)
1300 B=ARCSECH(X)
```

(continued)

```

1310 GOTO 220
1320 INPUT "Find the inverse hyperbolic cotangent of? ":X
 :: GOSUB 150 :: GOSUB 170
1330 X=X*DR :: DEF ARCCOTH(A)=LOG((A+1)/(A-1))/2
1340 B=ARCCOTH(X)
1350 GOTO 220
1360 INPUT "Find the inverse hyperbolic cosecant of? ":X ::
 GOSUB 150 :: GOSUB 170
1370 X=X*DR :: DEF ARCCSCH(A)=LOG(SGN(A)*SQR(A*A+1)+1)/A
1380 B=ARCCSCH(X)
1390 GOTO 220
1400 INPUT "Find the natural logarithm of? ":X :: GOSUB 150
 :: GOSUB 170
1410 B=LOG(X)
1420 GOTO 220
1430 INPUT "Find the reciprocal of? ":X :: GOSUB 150 ::
 GOSUB 170
1440 DEF RCL(A)=1/A
1450 B=RCL(X)
1460 GOTO 220
1470 INPUT "Number to be rounded to X decimals? ":X
1480 INPUT "How many decimal places? ":XX :: GOSUB 150 ::
 GOSUB 170
1490 DEC=1 :: FOR V=1 TO XX :: DEC=DEC*10 ::NEXT V :: B=INT
 (X*DEC+.5)/DEC
1500 GOTO 220
1510 INPUT "Find the integers of? ":X :: GOSUB 150 :: GOSUB
 170
1520 B=INT(X):: PRINT RR$;B :: GOSUB 150 :: PRINT
1530 BB=INT(X+.5)
1540 PRINT "and the rounded-off integer of ";X;" is: ";BB
1550 GOTO 200
1560 PRINT 1,"Convert degrees to radians" :: PRINT
1570 PRINT 2,"Convert radians to degrees" :: GOSUB 150
1580 INPUT "Which? ":WH :: GOSUB 160
1590 ON WH GOTO 1600,1640
1600 INPUT "Number of degrees? ":X :: GOSUB 150 :: GOSUB
 170
1610 DEF DEG(A)=A*(PI/180)
1620 B=DEG(X)
1630 GOTO 220
1640 INPUT "Number of radians? ":X :: GOSUB 150 :: GOSUB
 170
1650 DEF RAD(A)=A/(PI/180)
1660 B=RAD(X)
1670 GOTO 220
1680 GOSUB 160
1690 PRINT 1;" Addition"
1700 PRINT 2;" Subtraction"
1710 PRINT 3;" Multiplication"
1720 PRINT 4;" Division"

```

(continued)

```
1730 PRINT 5;" Exponents"
1740 PRINT 6;" Roots"
1750 PRINT 7;" Sine"
1760 PRINT 8;" Arcsine"
1770 PRINT 9;" Hyperbolic sine"
1780 PRINT 10;" Cosine"
1790 PRINT 11;" Arccosine"
1800 PRINT 12;" Hyperbolic cosine"
1810 PRINT 13;" Tangent"
1820 PRINT 14;" Arctangent"
1830 PRINT 15;" Hyperbolic tangent" :: GOSUB 150
1840 INPUT "See more categories? (Y/N) ":YY$
1850 IF YY$="Y" THEN GOSUB 160 :: GOTO 1860 ELSE 2090
1860 PRINT 16;" Cotangent"
1870 PRINT 17;" Arccotangent"
1880 PRINT 18;" Hyperbolic cotangent"
1890 PRINT 19;" Secant"
1900 PRINT 20;" Cosecant"
1910 PRINT 21;" Inverse secant"
1920 PRINT 22;" Inverse cosecant"
1930 PRINT 23;" Hyperbolic secant"
1940 PRINT 24;" Hyperbolic cosecant"
1950 PRINT 25;" Inverse hyperbolic sine"
1960 PRINT 26;" Inverse hyperb.cosine"
1970 PRINT 27;" Inverse hyperb.tangent"
1980 PRINT 28;" Inverse hyperb.secant"
1990 PRINT 29;" Inv.hyperb.cotangent"
2000 PRINT 30;" Inverse hyperb.cosecant"
2010 GOSUB 150 :: INPUT "See more categories? (Y/N) ":NN$
2020 IF NN$="Y" THEN GOSUB 160 :: GOTO 2030 ELSE 2090
2030 PRINT 31;" Natural logarithm"
2040 PRINT 32;" Reciprocal numbers"
2050 PRINT 33;" Rounding to X decimals"
2060 PRINT 34;" Integers"
2070 PRINT 35;" Degrees vs. radians" :: GOSUB 150
2080 PRINT 36;" Exit program" :: GOSUB 150
2090 INPUT "Which? ":WHICH :: GOSUB 160
2100 IF WHICH>26 THEN 2110 ELSE 2120
2110 WHICH=WHICH-26 :: GOTO 2130
2120 ON WHICH GOTO 230,280,320,360,400,440,480,510,550,
 590,620,660,700,730,840,760,800,880,920,960,1000,
 1040,1080,1120,1160,1200,2130
2130 ON WHICH GOTO 1240,1280,1320,1360,1400,1430,1470,
 1510,1560,190
```

---

**Lines 130** uses the built-in value of PI (available only in EXTENDED BASIC. In regular BASIC you would have to add  $PI = 3.1415927$ ). It assigns the value of  $PI/180$  (0.0174533) to the numeric variable

---

DR, which is used later to convert degrees to radians. Next it assigns a string to the string variable RR\$.

**Line 140** sends the computer to lines 1680–2100 to display the 36 choices available to the user.

**Lines 150–210** are subroutines used throughout the program.

**Line 220** is the line used at the end of each subprogram to display the result and then return the computer to the menu.

**Line 230** creates a defined function called ADD by using the DEF statement. Once this defined function,  $ADD(A) = X + Z$ , has been entered, it can be used over and over again, causing the values assigned to X and Z to be added.

**Line 240** sends the computer to the subroutine that asks you to key in any number.

**Line 250** asks you to key in the number to be added.

**Line 260** uses the previously defined function to perform the calculation and then assigns the result to the numeric variable B. During this process the A used when defining the function is replaced by the X to which the keyed-in value was assigned.

**Line 270** sends the computer to line 220, which causes the result to be displayed.

Since all the subprograms are quite similar in construction, I will from here on only discuss those lines that contain that types of statements and functions the program is designed to illustrate.

**Lines 280, 320, 360, 400, and 440** create five defined functions that perform the tasks of subtracting, multiplying, dividing, exponentiating, and obtaining a root. They are  $SBN(A) = X - Z$  for subtraction,  $MUL(A) = X * Z$  for multiplication,  $DIV(A) = X / Z$  for division,  $XPN(A) = X^Z$  for exponentiation, and  $ROT(A) = X^{(1/Z)}$  to find the root.

**Lines 300, 340, 380, 420, and 460** use those defined functions to perform the calculations and then assign the various results to the numeric variable B. The  $ROT(A) = X^{(1/Z)}$  function creates an exponent that is the reciprocal of the root figure, producing the correct result.

**Line 490** multiplies the keyed-in figure, representing degrees, by the value previously assigned to DR in order to convert degrees to

---

radians, because the trigonometric functions are designed to deal with radians rather than degrees. Next the built-in function SIN(X) is used to produce the result.

**Line 520** defines a function ARCSIN(A) to produce the arcsine of the keyed-in number. The expression itself is  $ATN(A/SQR(-A*A + 1))$ , using the two built-in functions. What happens is this: The value of A (or in this case X, which replaces the A in line 530) is divided by the positive square root (SQR) of  $-A*A + 1$ , and the built-in ATN function produces the arctangent of that figure, representing the arcsine of the keyed-in figure.

**Line 560** defines a function that determines the hyperbolic sine of the keyed-in value:  $SINH(A) = (EXP(A) - EXP(-A))/2$ . Here the built-in function EXP is used twice. EXP produces the exponential value of 2.718281828459, using the value assigned to the numeric variable in parentheses as the exponent. Thus, assuming that the value assigned to X (which replaces the A in line 570) is 25, the EXP function produces  $2.718281828459^{25}$  or, in the second use of the statement,  $e^{-25}$ , which results in  $7.20049E+10$  and  $1.38879E-11$ , respectively. The complete calculation performed by this function, assuming that the keyed-in value is 25, is  $(7.20049E+10 - 1.38879E-11)/2$ .

**Line 600** uses the built-in function COS(A) to find the cosine of the keyed-in value.

**Line 630** defines a new function used in the expression  $ARC-COS(A) = ATN(A/SQR(-A*A + 1)) + 1.5708$  to find the arccosine of the keyed-in value, using the two built-in functions ATN (arctangent) and SQR (square root) that we've already discussed.

**Line 670** defines another function used in the expression  $COSH(A) = (EXP(A) + EXP(-A))/2$ , which is similar to the one in line 560, in this version producing the hyperbolic cosine of the keyed-in value.

**Lines 710 and 740** use the built-in function TAN(X) (tangent) and ATN(X) (arctangent) to produce the desired results.

**Lines 770, 810, 850, 890, 930, and 970** define additional functions, each using some of the built-in functions we've discussed earlier.

**Lines 1010 and 1050** utilize another built-in function, SGN(A), which produces 1 if the value assigned to the numeric variable is positive, 0 if the value is zero, or -1 if the value is negative.

---

**Lines 1090 and 1130** again use previously discussed built-in functions to define the numeric expressions represented by the defined functions.

**Lines 1170, 1210, 1250, 1290, 1330, and 1370** use the built-in function  $\text{LOG}(A)$ , which returns the natural logarithm of the numeric variable in defining additional functions.

**Line 1410** uses the same built-in function  $\text{LOG}(X)$  to find the natural logarithm of the value assigned to  $X$ .

**Line 1440** defines a new function designed to produce the reciprocal value of the one keyed in and assigned to  $X$ . The reciprocal value can be used to multiply instead of divide or vice versa, for example,  $5 \times 100 = 500$  and  $5 / .01 = 500$ , where  $.01$  is the reciprocal of  $100$ . Reciprocals of any value are produced by  $1/\text{value}$ .

**Line 1490** causes the numeric variable  $\text{DEC}$  to represent  $10$ ,  $100$ ,  $1000$ ,  $10000$ , etc., depending on the keyed-in value assigned to  $\text{XX}$  through the use of a  $\text{FOR} \dots \text{TO} \dots \text{NEXT}$  loop, where the number of zeros represents the number of decimal places.

**Lines 1520 and 1530** use the built-in function  $\text{INT}(X)$  to return the ordinary integer and  $\text{INT}(X + .5)$  to return the rounded-off integer.

**Lines 1610 and 1650** define two functions used in the expressions  $\text{DEG}(A) = A \times (\text{PI}/180)$  and  $\text{RAD}(A) = A / (\text{PI}/180)$  to convert degrees to radians and vice versa.

Using defined functions rather than repeatedly typing in entire numeric expressions not only saves a lot of typing, it also reduces the chance of making typing errors in the mathematical formulas, which could produce incorrect results that might go unnoticed or, if noticed, would often be difficult to correct because the error would be hard to find.

In the above program, the use of the trigonometric functions in conjunction with certain keyed-in values may produce a **BAD ARGUMENT** error message, because these functions cannot be used with certain values. When that happens the computer automatically exits the program and you'll have to start over again by typing **RUN**.

---

## NUMEROLOGY

Now let's look at another group of functions. Our second program deals with numerology; for those who don't know what numerology is all about (I didn't until I wrote the program), I'll first explain the principle involved. The idea of numerology is that the combination of the letters in your name and the digits representing your date of birth produce a single-digit number that is supposed to have some influence on your life. To find this single-digit number, the letters in your name are assigned numeric values from 1 to 9, with the value assigned to each letter shown in lines 210-230 in the program (A = 1, B = 2, etc.). The numbers that correspond to the name are then added: JONATHAN is represented as  $1 + 6 + 5 + 1 + 2 + 8 + 1 + 5 = 29$ . Now we add the two resulting digits,  $2 + 9 = 11$ , and then we once more add the two resulting digits,  $1 + 1 = 2$ , to arrive at the numerology figure for the name JONATHAN. This routine is repeated over and over for the middle name(s), the last name, the birth month, birth day, and birth year, until the final result is a single-digit number.

Although the Numerology Program is of no practical value except to numerology aficionados, it is ideally suited to demonstrate certain statements.

---

### NUMEROLOGY PROGRAM

---

A program that calculates the numerical values of your name and birth date.

---

```
100 REM NUMEROLOGY
110 REM TI EXTENDED BASIC
120 CALL CLEAR
130 GOTO 200
140 PRINT "-----" :: RETURN
150 CALL CLEAR :: RETURN
160 FOR Q=1 TO 5 :: PRINT :: NEXT Q :: RETURN
170 PRINT :: INPUT "Press >ENTER< =" :E$:: RETURN
180 GOSUB 150 :: GOSUB 140 :: PRINT TAB(8);"End." :: GOSUB
140 :: GOSUB 160
190 END
200 PRINT "This program determines the numerical
values of your name and birth date" :: GOSUB 140 ::
GOSUB 170 :: GOSUB 160
```

(continued)

---

```

210 DATA A,1,B,2,C,3,D,4,E,5,F,6,G,7,H,8,I,9
220 DATA J,1,K,2,L,3,M,4,N,5,O,6,P,7,Q,8,R,9
230 DATA S,1,T,2,U,3,V,4,W,5,X,6,Y,7,Z,8
240 GOSUB 150
250 PRINT "Use no single-letter initials!" :: PRINT
260 PRINT "For no middle name type NMI" :: PRINT
270 PRINT "Use no hyphens!" :: GOSUB 140
280 INPUT "First name (to 10 letters) ":FF$
290 INPUT "Middle name(s) (to 10 let.s)":MN$
300 INPUT "Last name (to 10 letters) ":LN$:: GOSUB 140
310 INPUT "Birth month (1 or 2 digits) ":MM
320 INPUT "Birth day (1 or 2 digits) ":DD
330 INPUT "Birth year (4 digits) ":YY :: GOSUB 140
340 LL=LEN(FF$):: LLL=LEN(MN$):: LLLL=LEN(LN$)
350 P=1 :: LL=LL-1
360 ON LL GOTO 510,530,560,590,630,670,720,770,830
370 RESTORE
380 IF P>LL THEN 890
390 READ N$
400 ON P GOTO 410,420,430,440,450,460,470,480,490,
500
410 IF N$=LETTER1$ THEN 1310 ELSE 380
420 IF N$=LETTER2$ THEN 1320 ELSE 380
430 IF N$=LETTER3$ THEN 1330 ELSE 380
440 IF N$=LETTER4$ THEN 1340 ELSE 380
450 IF N$=LETTER5$ THEN 1350 ELSE 380
460 IF N$=LETTER6$ THEN 1360 ELSE 380
470 IF N$=LETTER7$ THEN 1370 ELSE 380
480 IF N$=LETTER8$ THEN 1380 ELSE 380
490 IF N$=LETTER9$ THEN 1390 ELSE 380
500 IF N$=LETTER0$ THEN 1400 ELSE 380
510 LETTER1$=SEG$(FF$,1,1):: LETTER2$=SEG$(FF$,2,1)
520 GOTO 370
530 LETTER1$=SEG$(FF$,1,1):: LETTER2$=SEG$(FF$,2,1)
540 LETTER3$=SEG$(FF$,3,1)
550 GOTO 370
560 LETTER1$=SEG$(FF$,1,1):: LETTER2$=SEG$(FF$,2,1)
570 LETTER3$=SEG$(FF$,3,1):: LETTER4$=SEG$(FF$,4,1)
580 GOTO 370
590 LETTER1$=SEG$(FF$,1,1):: LETTER2$=SEG$(FF$,2,1)
600 LETTER3$=SEG$(FF$,3,1):: LETTER4$=SEG$(FF$,4,1)
610 LETTER5$=SEG$(FF$,5,1)
620 GOTO 370
630 LETTER1$=SEG$(FF$,1,1):: LETTER2$=SEG$(FF$,2,1)
640 LETTER3$=SEG$(FF$,3,1):: LETTER4$=SEG$(FF$,4,1)
650 LETTER5$=SEG$(FF$,5,1):: LETTER6$=SEG$(FF$,6,1)
660 GOTO 370
670 LETTER1$=SEG$(FF$,1,1):: LETTER2$=SEG$(FF$,2,1)
680 LETTER3$=SEG$(FF$,3,1):: LETTER4$=SEG$(FF$,4,1)
690 LETTER5$=SEG$(FF$,5,1):: LETTER6$=SEG$(FF$,6,1)
700 LETTER7$=SEG$(FF$,7,1)

```

(continued)



```
710 GOTO 370
720 LETTER1$=SEG$(FF$,1,1):: LETTER2$=SEG$(FF$,2,1)
730 LETTER3$=SEG$(FF$,3,1):: LETTER4$=SEG$(FF$,4,1)
740 LETTER5$=SEG$(FF$,5,1):: LETTER6$=SEG$(FF$,6,1)
750 LETTER7$=SEG$(FF$,7,1):: LETTER8$=SEG$(FF$,8,1)
760 GOTO 370
770 LETTER1$=SEG$(FF$,1,1):: LETTER2$=SEG$(FF$,2,1)
780 LETTER3$=SEG$(FF$,3,1):: LETTER4$=SEG$(FF$,4,1)
790 LETTER5$=SEG$(FF$,5,1):: LETTER6$=SEG$(FF$,6,1)
800 LETTER7$=SEG$(FF$,7,1):: LETTER8$=SEG$(FF$,8,1)
810 LETTER9$=SEG$(FF$,9,1)
820 GOTO 370
830 LETTER1$=SEG$(FF$,1,1):: LETTER2$=SEG$(FF$,2,1)
840 LETTER3$=SEG$(FF$,3,1):: LETTER4$=SEG$(FF$,4,1)
850 LETTER5$=SEG$(FF$,5,1):: LETTER6$=SEG$(FF$,6,1)
860 LETTER7$=SEG$(FF$,7,1):: LETTER8$=SEG$(FF$,8,1)
870 LETTER9$=SEG$(FF$,9,1):: LETTER0$=SEG$(FF$,10,1)
880 GOTO 370
890 GOSUB 150 :: GOSUB 140 :: PRINT TAB(12);"WAIT!" ::
 GOSUB 140 :: GOSUB 160
900 QQ=QQ+1 :: IF QQ=1 THEN 930
910 IF QQ=2 THEN 940
920 IF QQ>2 THEN 950
930 GOSUB 960 :: FF$=MN$:: LL=LLL :: GOTO 350
940 GOSUB 960 :: FF$=LN$:: LL=LLLL :: GOTO 350
950 GOSUB 960 :: GOSUB 1080 :: GOSUB 170 :: GOTO 180
960 NN=N1+N2+N3+N4+N5+N6+N7+N8+N9+N0
970 NN$=STR$(NN):: NN1$=SEG$(NN$,1,1):: NN2$=SEG$(NN$,2,1)
980 NN1=VAL(NN1$):: NN2=VAL(NN2$):: NN=NN1+NN2
990 IF NN>9 THEN 1000 ELSE GOTO 1020
1000 NU$=STR$(NN):: NU1$=SEG$(NU$,1,1):: NU2$=SEG$(NU$,
 2,1)
1010 NU1=VAL(NU1$):: NU2=VAL(NU2$):: NN=NU1+NU2 :: GOTO
 990
1020 IF QQ=1 THEN 1050
1030 IF QQ=2 THEN 1060
1040 IF QQ=3 THEN 1070
1050 NN1=NN :: RETURN
1060 NN2=NN :: RETURN
1070 NN3=NN :: RETURN
1080 MM$=STR$(MM):: DD$=STR$(DD):: YY$=STR$(YY)
1090 MM1$=SEG$(MM$,1,1):: MM2$=SEG$(MM$,2,1)
1100 MM1=VAL(MM1$):: MM2=VAL(MM2$):: MM3=MM1+MM2
1110 DD1$=SEG$(DD$,1,1):: DD2$=SEG$(DD$,2,1)
1120 DD1=VAL(DD1$):: DD2=VAL(DD2$):: DD3=DD1+DD2
1130 YY1$=SEG$(YY$,1,1):: YY2$=SEG$(YY$,2,1):: YY4$=SEG
 $(YY$,3,1)
1140 YY5$=SEG$(YY$,4,1)
1150 YY1=VAL(YY1$):: YY2=VAL(YY2$):: YY4=VAL(YY4$):: YY
 5=VAL(YY5$)
1160 YY3=YY1+YY2+YY4+YY5
```

(continued)

---

```

1170 IF YY3>9 THEN 1180 ELSE 1200
1180 YY3=YY3*10 :: YY3$=STR$(YY3):: Y1$=SEG$(YY3$,1,1):
 : Y2$=SEG$(YY3$,2,1)
1190 Y1=VAL(Y1$):: Y2=VAL(Y2$):: YY3=Y1+Y2
1200 NN4=MM3+DD3+YY3
1210 IF NN4>9 THEN 1220 ELSE PRINT NN4 :: GOTO 1240
1220 NN5=NN4 :: NN5$=STR$(NN5):: NN6$=SEG$(NN5$,1,1)::
 NN7$=SEG$(NN5$,2,1)
1230 NN6=VAL(NN6$):: NN7=VAL(NN7$):: NN4=NN6+NN7 :: GOTO
 1210
1240 RESULT=2+NN1+NN2+NN3+NN4 :: IF MN$="NMI" THEN 1250
 ELSE 1260
1250 RESULT=RESULT-NN2
1260 IF RESULT>9 THEN 1270 ELSE 1290
1270 RESULT$=STR$(RESULT):: R1$=SEG$(RESULT$,1,1):: R2$
 =SEG$(RESULT$,2,1)
1280 R1=VAL(R1$):: R2=VAL(R2$):: RESULT=R1+R2 :: IF RESULT
 >9 THEN 1270 ELSE 1290
1290 GOSUB 150 :: GOSUB 140
1300 PRINT "Your numerology figure: ";RESULT :: GOSUB 140
 :: RETURN
1310 READ N1 :: P=2 :: GOTO 370
1320 READ N2 :: P=3 :: GOTO 370
1330 READ N3 :: P=4 :: GOTO 370
1340 READ N4 :: P=5 :: GOTO 370
1350 READ N5 :: P=6 :: GOTO 370
1360 READ N6 :: P=7 :: GOTO 370
1370 READ N7 :: P=8 :: GOTO 370
1380 READ N8 :: P=9 :: GOTO 370
1390 READ N9 :: P=10 :: GOTO 370
1400 READ N0 :: P=11 :: GOTO 370

```

**Lines 210–230** place the letters of the alphabet and the associated numbers into DATA lines.

**Lines 340** uses the LEN statement to determine the number of letters in the first (FF\$), middle (MN\$), and last (LN\$) names, assigning the results to the numeric variables LL, LLL, and LLLL. Thus, if the first name is JONATHAN, LL = LEN(FF\$) would result in assigning 8 (the number of letters in the name) to the variable LL.

**Line 350** assigns 1 to the variable P and reduces the number assigned to LL by 1 because the ON LL GOTO statement assumes the first value of LL to be 1, and the shortest possible name that can be used consists of two letters.

**Line 360** sends the computer to one of nine line numbers, depending on the value of LL.

**Line 370** makes sure that the DATA lines are READ from the beginning.

**Line 380** sends the computer to line 890 if the value of P is greater than that of LL.

**Line 390** causes the computer to READ an item from the DATA lines, assigning the item to the string variable N\$.

**Line 400** sends the computer to one of 10 line numbers depending on the value of P.

**Lines 410–500** compare the single-letter string assigned to the string variable N\$ with the single-letter string variables assigned to LETTER1\$ through LETTER0\$, sending the computer to one of two line numbers, depending on whether or not a match has been found.

**Line 510** is used if the name being examined consists of two letters. There `LETTER1$=SEG$(FF$,1,1)` assigns the first letter in the string, represented by FF\$, to the string variable, LETTER1\$, because the first digit inside the parentheses after FF\$ represents the consecutive position of the letter(s) to be separated by the SEG\$ statement, and the second digit represents the number of letters to be separated. Thus, in this line, the first letter is assigned to LETTER1\$ and the second to LETTER2\$.

**Lines 530–870** perform the same task for names from three to 10 letters.

**Line 890** comes into play while the computer is performing a long list of calculations, placing WAIT! into display because the process takes quite a bit of time and you might be tempted to think that the computer has malfunctioned if the screen were simply left blank.

**Line 1080** uses the `MM$=STR$(MM)` expression to convert the numeric value assigned to the numeric variable MM to the string variable MM\$, and the same is then done to the numeric variable DD and YY, transforming them to DD\$ and YY\$.

**Lines 1090, 1110, 1130, 1140, 1180, 1220, and 1270** once more use the SEG\$ statement to separate the individual characters represented by the string variables.

**Lines 1100, 1120, 1150, 1190, 1230, and 1280** use the `MM1=VAL(MM1$)` phrase to convert the string value assigned to MM1\$ to a numeric value, which is then assigned to MM1, and the same routine is followed with the other string variables.

---

**Line 1300** displays the resulting single-digit figure.

**Lines 1310–1400** assign different values to the numeric variable P as the computer READs the numeric items in the DATA lines, assigning them to N1 through N0, each time returning the computer to line 370.

The reason for all these conversions from string variables to numeric variables and vice versa is the fact that the SEG\$ statement can be used only with string variables. It would, therefore, be rather cumbersome to separate the individual digits that make up the original numeric values. It can, of course, be done by using a series of calculations. For instance, to separate the four digits in 1984, you could use this sequence of steps:

```
100 YY=1984
110 Y1=YY/1000::Y2=INT(Y1)::Y3=Y1-Y2::Y3=Y3*10
120 Y4=INT(Y3)::Y5=Y3-Y4::Y5=Y5*10
130 Y6=INT(Y5)::Y7=Y5-Y6::Y7=Y7*10
```

which would result in  $Y2 = 1$ ,  $Y4 = 9$ ,  $Y6 = 8$ , and  $Y7 = 4$ . It goes without saying that this would be a rather tiresome way of doing things, not to mention the ever-present chance of making some sort of typing error.

In the above program we used the DATA lines to associate certain numbers with letters. Here, too, an alternative method is available to us. We could use the ASC statement, which returns the ASCII numbers for the various letters:

```
110 NN=ASC("A")
120 NN=NN-64::PRINT NN
```

This would result in NN representing 1, because the ASCII code number for A is 65. Using this method, the number to be subtracted for letters A through I is 64, for letters J through R it is 73, and for letters S through Z it is 82.

---

Two other statements we have not yet used are MAX and MIN, which return either the largest or the smallest value assigned to two numeric variables that must be enclosed in parentheses. For example:

```
100 A = MAX(C,D)::PRINT A
110 B = MIN(C,D)::PRINT B
```

will assign the highest of the values assigned to C and D to the numeric variable A, and the smallest value to the variable B.

---

# 15

---

## ***Having fun with TI LOGO II***

---

---

LOGO, like BASIC, is a computer language that can be used to write all manner of programs. Its greatest advantage is the simplicity with which it is possible to create fascinating graphic designs and that, in turn, makes it an ideal means to permit youngsters of even preschool age to be introduced to the use of computers. Although LOGO is capable of being used to write programs other than graphics, it is this latter facet that we'll concentrate on in this chapter.

In order to use LOGO, you must have the LOGO module installed in your TI-99/4A Home Computer, and it must be equipped with the Extended Memory board that adds the needed 32K bytes to the standard 16K bytes of memory. With the computer so equipped, the best way to find out what LOGO is all about is to start right in.

### **TURTLE GRAPHICS**

Turn on the system in the usual manner, first the Peripheral Expansion System, then the monitor, and last the computer with the TI LOGO module in place. When the master title appears, press any key.

The display will show:

PRESS

1 FOR TI BASIC  
2 FOR TI LOGO II

Now press 2 and the screen will blank out. After a short pause you will see:

WELCOME TO TI LOGO  
? ---

because LOGO uses the underline as a prompt. The system is now ready to accept your first command. The primary graphic system used by LOGO is referred to as *turtle graphics*, and in order to get into that mode we now type:

TELL TURTLE

---

and press >ENTER<. Again the screen blanks and then there appears a small triangle in the center of the screen, and the question mark and underline are moved to a place about three-quarters down from the top at the left edge of the screen. That small triangle is the "turtle." In order to make the turtle move and, in doing so, draw pictures, we must now tell it where to go. Type:

FORWARD 50      >ENTER<

and see what happens. The turtle has drawn a vertical line because the top of the triangle that represents the nose of the turtle is pointing upward. Now type:

LEFT 90      >ENTER<

and note that the turtle has turned 90 degrees to the left. Now type:

```
FORWARD 50 >ENTER<
LEFT 90 >ENTER<
FORWARD 50 >ENTER<
LEFT 90 >ENTER<
FORWARD 50 >ENTER<
```

When you're through, the turtle will have drawn a square box. What we have discovered is that FORWARD plus a number causes the turtle to travel a given distance, leaving a trail in the form of a line. And we have learned that by typing RIGHT or LEFT and a number, the turtle will turn that many degrees in the desired direction. In this manner we can move it all over the place in order to draw any kind of design we have in mind.

Now we'll try something else. So far the lines drawn by the turtle have been straight. Here is a command that produces a circle. First type:

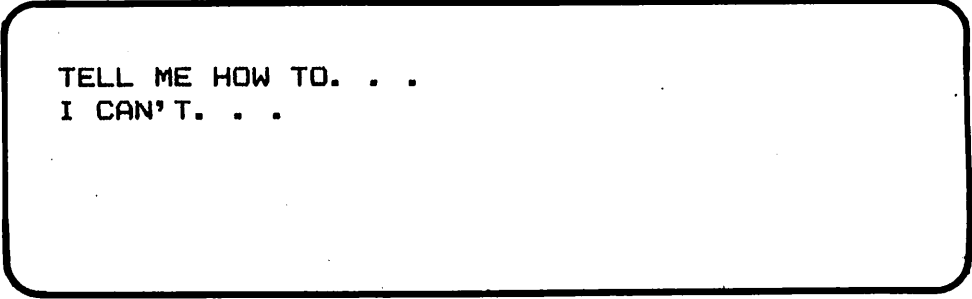
```
PENUP >ENTER<
FORWARD 20 >ENTER<
PENDOWN >ENTER<
REPEAT 36 [RIGHT 10 FORWARD 10] >ENTER<
```

---



and presto, we have a circle. Here we have learned several new words: PENUP, PENDOWN, and REPEAT, plus the use of square brackets. PENUP permits us to move the turtle without drawing a line, and PENDOWN reverses the previous command. REPEAT causes the turtle to repeat a given command as many times as the number that follows it. But REPEAT requires two inputs: the number of repetitions, and the action that is to be repeated. Here that action is RIGHT 10 and FORWARD 10, which must be enclosed in square brackets (FCTN R and FCTN T on the keyboard) in order to be treated by the REPEAT command as one series of actions, referred to in LOGO as a "list."

By now, if you've made a typing error somewhere along the line, you've met one of LOGO's many error messages, which are expressed in plain English, such as:



```
TELL ME HOW TO. . .
I CAN'T. . .
```

and so on, telling us clearly what our mistake was. For instance, if you type RIGHT90 with no space between the word and the number, the display will respond with:



```
TELL ME HOW TO RIGHT90
```

---

because the computer assumes that you had something special in mind by combining the word and the number.

At this point I suggest you play around with drawing straight and curved lines to get the feel of things before going on to the next stage.

## **LOGO PROCEDURES**

Just as we have learned in writing programs in BASIC how we can create entire subroutines that are identified by line numbers, so we can create various routines, referred to as *procedures* in LOGO, that are identified by a single word. To create a procedure, we use the word TO followed by the name of the procedure. Here is an example:

```
TO SQUARE
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
BACK (FCTN 9)
```

using >ENTER< after each command. (We'll assume from now on that you'll remember to press >ENTER<.) The first line tells LOGO that we want to define a procedure named SQUARE. The next seven lines constitute the action that comprises the procedure. The last line tells LOGO that the end of the procedure has been reached. When you type TO SQUARE, the screen changes color and in the upper left-hand corner you'll see:

```
TO SQUARE ----
END
```

---

When you press >ENTER<, the END will move down one line to permit you to type in the first line of the procedure, and it continues to move down as you add more lines. The reason is that every procedure must be closed with that END line, which is provided automatically. (Do not type END yourself, or you will be given an error message.) After you've closed the procedure by typing FCTN 9 (BACK), the screen returns to its original color, displaying the name of your procedure. You're now ready to use it. Simply type its name:

SQUARE

and the turtle will execute the procedure. You can now use your procedure as part of a new series of commands:

```
PENUP
FORWARD 25
PENDOWN
REPEAT 8 [RIGHT 45 SQUARE]
```

The turtle will draw eight squares, advancing 45 degrees to the right each time.

## TURTLE COMMANDS

Experimenting with different procedures can be a lot of fun, and you can, of course, use procedures within procedures. In order to get a clear picture of how much we can make the turtle do, let's take a quick look at the commands that are available. Many commands may be abbreviated to save typing, and these abbreviations are shown in parentheses.

**BACK (BK)** moves the turtle in a reverse direction as many steps as indicated by a number: BK 50.

**CLEARSCREEN (CS)** clears the screen of all but the turtle and the prompt.

**ERASE** can be used to delete a procedure: ERASE BOX.

---

**FORWARD (FD)** moves the turtle forward as many steps as indicated by a number: FD 50.

**HIDETURTLE (HT)** removes the turtle but not any lines drawn.

**HOME** moves the turtle to the starting position, center screen, facing up (north).

**LEFT (LT)** turns the turtle to the left by the number of degrees indicated :LT 90.

**NOTURTLE** exits turtle mode.

**PA** displays all procedures and procedure names currently in the computer memory.

**PENDOWN (PD)** reverses the PENUP command.

**PENERASE (PE)** causes the turtle to erase previously drawn lines.

**PENREVERSE (PR)** causes the turtle to draw lines except when passing over previously drawn lines, which will be erased.

**PENUP (PU)** causes the turtle to draw no lines when moving.

**PN** prints all names of procedures currently in memory.

**PO** prints the definition of a named procedure: PO BOX prints the steps that constitute BOX.

**REPEAT** requires two items of information, the number of REPEATs and the steps that are to be REPEATed: REPEAT 10 [RT 20 BOX] prints 10 shapes defined by the procedure BOX, rotating 20 degrees to the right each time. The steps to be repeated must be enclosed in square brackets.

**RIGHT (RT)** turns the turtle to the right by the indicated number of degrees: RT 90.

---

**SETHEADING (SH)** turns the turtle to the heading, in degrees, indicated by a number: SH 270 turns the turtle to the left (westward).

**SHOWTURTLE (ST)** restores the turtle after the HIDE TURTLE command has been used.

**TELL TURTLE** is used to enter the turtle graphics mode, and it must also be used under certain conditions when using color (see below).

**TO PROCEDURE** defines the following steps as a procedure named PROCEDURE (or any name you give it). When all procedure steps have been entered, type FCTN 9 (BACK) rather than END. (In subsequent procedures we'll omit typing FCTN 9, using END instead.)

There are many other commands, some of which we'll look at later, but the ones listed here are all you really need to experiment with the different aspects of turtle graphics.

So far we've been satisfied with drawing black lines on a bright background. But turtle graphics can also be used in color. For that there are these commands:

**SET COLOR (SC)** is used to determine the color in which the turtle draws. It is used either in combination with a number from 0 to 15 to identify the desired color (e.g., SC6 causes red lines to be drawn); or in conjunction with the name of the color, preceded by a colon (e.g., SC :RED).

**COLORBACKGROUND (CB)** is used in the same way, controlling the color of the background: CB 1 or CB :BLACK produces a black background. The available colors are:

|    |        |    |        |
|----|--------|----|--------|
| 0  | clear  | 1  | black  |
| 2  | green  | 3  | lime   |
| 4  | blue   | 5  | sky    |
| 6  | red    | 7  | cyan   |
| 8  | rust   | 9  | orange |
| 10 | yellow | 11 | lemon  |
| 12 | olive  | 13 | purple |
| 14 | gray   | 15 | white  |

---

Now let's quickly write a sample procedure that makes use of these commands:

```
TO SQUARE1
 CB 2
 FD 20 RT 90
 SC 6
 FD 20 RT 90
 CB 4 SC 10
 FD 20 RT 90
 SC 1
 FD 20 END
```

What we've done is to write a procedure that draws a square with different color sides, using the standard background color (2) to start with and then changing it in the middle to blue (4). Now type:

```
REPEAT 8 [RT 45 SQUARE1]
```

and see how the turtle draws eight squares, each time moving 45 degrees to the right, while the background flashes back and forth between green and blue. At this point you might use some of the procedures you've written previously and add color commands. Use your imagination. You'll soon get fascinated as you keep experimenting.

## ***PROCEDURES WITH INPUTS***

So far we've written procedures in which the values associated with the commands, such as the 20 in FD 20, are part of the procedures and cannot be changed unless you go into the edit mode (by typing TO PROCEDURE, where PROCEDURE is the name you gave your procedure) and change those values. But it is possible to write procedures in such a manner that you can input different values without having to rewrite the procedure itself. Such a "procedure with input" might look like this:

```
TO SQUARE2 :SIDE
 REPEAT 8 [FD :SIDE RT 45]
 END
```

---

When you now type SQUARE2, the display responds with TELL ME MORE, because you have failed to define what SIDE stands for. Now type SQUARE2 20, and the turtle will draw an octagonal shape with each side being 20 steps long. Now type SQUARE2 30 and then SQUARE2 40 and you'll see how increasingly larger versions of the same shape are drawn.

Inputs do not need to be limited to one per procedure. You can just as easily write procedures with multiple inputs: The only thing you must remember is the consecutive order of such inputs—which stands for what—because LOGO does not prompt you by displaying the name of the input (SIDE in the above example). Let's try one:

```
TO DESIGN :SIDE :DEGREE
FD :SIDE RT :DEGREE
DESIGN :SIDE :DEGREE
END
```

Now, to see what happens, type this command:

```
DESIGN 30 90
```

The turtle draws a square but refuses to stop. To stop its frantic action, type BACK (FCTN 9). Now use CS to clear the screen and then type this series of commands, each time stopping the turtle action and clearing the screen as before:

```
DESIGN 30 120
DESIGN 40 150
DESIGN 60 80
DESIGN 50 144
DESIGN 5 10
```

Experiment with using all kinds of different arbitrary values, always remembering that the first value refers to the number of forward steps taken by the turtle, and the second number refers to the number of degrees it turns to the right.

This ability of the turtle to continue its action until it is stopped by pressing BACK is referred to as *recursion*, and it can be used to

---

change some of the input values automatically. To demonstrate, let's start with this:

```
TO COUNT :VALUE
PRINT :VALUE
COUNT :VALUE - 1
END
```

Now, in turtle mode, type COUNT 100 and you will see numbers displayed, starting with 100 and reducing by 1 in quick succession until you stop the action in the usual manner. Now let's take this one step further by using that principle in a graphic procedure:

```
TO GROW :SIDE :ANGLE
FD :SIDE RT :ANGLE
GROW (:SIDE + 2) :ANGLE
END
```

Now type some of the following commands, always stopping the action and clearing the screen after each:

```
GROW 0 90
GROW 10 120
GROW 10 144
```

Try picking other values at random. Because of the + 2 in the third line, the number of forward steps is increased by 2 during each pass. Note that regular parentheses were used, because this is a calculation, not a list.

There is a way we can stop that continuous action automatically, by inserting a conditional statement similar to those used in BASIC. The statement uses IF. Thus, if we use the previous procedure and add an IF statement, we can cause the action to be stopped automatically:

```
TO GROW2 :SIDE :ANGLE
IF :SIDE > 100 THEN STOP
FD :SIDE RT :ANGLE
GROW2 (:SIDE + 2) :ANGLE
END
```

---



This is the same as the previous procedure, except that the action of the turtle stops automatically when the value of SIDE reaches 100 steps.

## **PROCEDURES WITHIN PROCEDURES**

As you will have gathered by now, you can use procedures within procedures and within other procedures. You might simply think of each (borrowing the terms used in BASIC) as a subroutine that can be used within other subroutines within a program. For instance, here is a procedure that uses two other procedures. First we must write the two others:

```
TO TRIANGLE
REPEAT 3 [FD 50 RT 120]
END
TO BOX
PU HOME LT 115 FD 15 RT 115 PD
REPEAT 4 [FD 60 RT 90]
PU HOME PD
END
```

And then we create a third procedure using these two:

```
TO CIRCLE
TRIANGLE BOX
PU HOME LT 90 FD 35 RT 90 FD 20 PD
REPEAT 36 [FD 10 RT 10]
PU HOME PD
END
```

Now when we type CIRCLE, the turtle draws a triangle within a square within a circle. All those PU. . .PD lines are used to move the turtle a certain distance from the home position without leaving a trail, prior to drawing the next shape. You could now develop another procedure that uses CIRCLE that would then automatically also include TRIANGLE and BOX. It must be remembered and understood that any name you give to a procedure becomes a com-

---

mand consisting of any number of predetermined steps. In this manner we can create literally hundreds of different commands that can subsequently be used as parts of other procedures, which then, in turn, become commands containing other commands. The instruction book that comes with the TI LOGO module is full of examples.

## **SPRITE GRAPHICS**

In addition to the turtle graphics capability, TI LOGO (unlike other versions of LOGO) permits us to use LOGO in combination with the sprite capability. LOGO includes five predetermined sprite shapes, but you can also define your own shapes by using the appropriate commands, as we'll see. Let's start by using some of the predetermined shapes, which are:

- 1 PLANE
- 2 TRUCK
- 3 ROCKET
- 4 BALL
- 5 BOX

To start the process, clear the turtle from the screen by typing:

```
NOTURTLE
HOME
```

followed by:

```
TELL SPRITE 1
SC 6
CARRY 1
HOME
```

You'll see a plane facing right. You can now move it all over the screen by using the FD, RT, LT, and BK commands though, although RT and LT control the direction of movement, they do not turn the nose of the airplane to a different direction, the way the tur-

---

tle does. In order to manipulate the sprites, we'll have to learn a number of additional commands, which, along with explanations, are listed below.

**BIG** doubles the size of a sprite. See **SMALL** and **SIZE**.

**CARRY** is used in conjunction with the name of a sprite (**CARRY :PLANE**) or with a number representing a given sprite shape (**CARRY 1**).

**COLOR** calls up the number that represents the color of the current sprite. In practice it might be used to produce changes in the color during the execution of a procedure: **SC (:COLOR + 1)**.

**EACH** causes all sprites to perform in accordance with associated inputs: **EACH [SH :WEST]** causes all sprites to be prepared to move to the left with the next **FD** command.

**FREEZE** cancels previous motion commands, holding all sprites in position. **THAW** reverses the action.

**HEADING** produces the heading (in degrees) of the active sprite (or the turtle). In practice it can be used in the same way as **COLOR**: **SH (:HEADING - 3)**.

**LOOKLIKE** is the same as **CARRY**.

**MAKECHAR (MC)** permits the definition or editing of the character shape represented by a number in the range from 0 to 255: **MC 52**. The character shapes represent the ASCII characters (see Appendix I) or shapes defined by you.

**MAKESHAPE (MS)** permits changing the shape of the sprite represented by a number in the range from 0 to 25: for instance, **MS 18**, where the number identifies the sprite whose shape is to be changed. A maximum of 26 sprites may be used at one time.

**NUMBEROF** produces the number of the active sprite when used with **WHO**: **PRINT NUMBEROF WHO**.

---

**SETSPEED (SS)** controls the speed of movement of sprites when used with a number in the range from  $-127$  to  $127$ , where low numbers produce a slow motion and large numbers result in fast motion. Negative numbers cause motion in the opposite direction of the last SH command, whereas positive numbers move the sprite in the last SH heading. To use the command with a specific sprite, it must be preceded by TELL and the sprite number: TELL SPRITE 15 SS  $-5$ .

**SHAPE** produces the shape number of the active sprite. It can be used to change the shape in the same manner in which COLOR is used.

**SIZE** produces one of two numbers: 16 if sprites are the normal size, or 32 if the BIG command was used to double the size of sprites.

**SMALL** reverses the action of BIG by reducing sprite size to normal.

**SPEED** produces the number of the speed at which sprites are moving. Use in the same manner as COLOR.

**SPRITE** is used in conjunction with TELL and a numeric input from 0 to 31 to direct commands to a given sprite: TELL SPRITE 3 SS 50.

**SV**, with two numbers, determines the velocity of vertical and horizontal movement of a given sprite: TELL SPRITE 1 SV 20 10 causes movement at a speed of 20 horizontally and 10 upward. Using negative numbers reverses the direction of travel.

**SX** is used with one number to move the sprite instantly to the horizontal coordinate represented by that number. After the relocation, previous motion continues.

**SXV** is used with one number from  $-127$  to  $127$  to control the speed of horizontal travel.

---

**SXY** moves the sprite to the position on the screen represented by two numbers which stand for the x/y coordinates, with 0 0 representing screen center.

**SY** instantly moves the sprite to the vertical (y) coordinate represented by a number. (0 is screen center, vertically). After the relocation, previous movement continues.

**SYV** is used with one number in the range from -127 to 127 to control the vertical speed of the sprite. Positive numbers move upward, negative numbers downward.

**TELL** directs the commands that follow to a given sprite: **TELL SPRITE 1** causes only that sprite to be affected by subsequent commands.

**THAW** restores motion after **FREEZE**.

**WHO** identifies the currently active sprite: **PRINT WHO** results in **SPRITE 1** or whichever sprite number is active.

**XVEL** produces the currently used horizontal velocity of the active sprite. Used in the same manner as **COLOR**.

**YVEL** is the same as above, relative to vertical velocity.

When we're using sprites, the **CLEARSCREEN (CS)** command does not affect the sprites, only the text on the screen. To get rid of the sprite in order to produce something new, we'll have to write a little procedure:

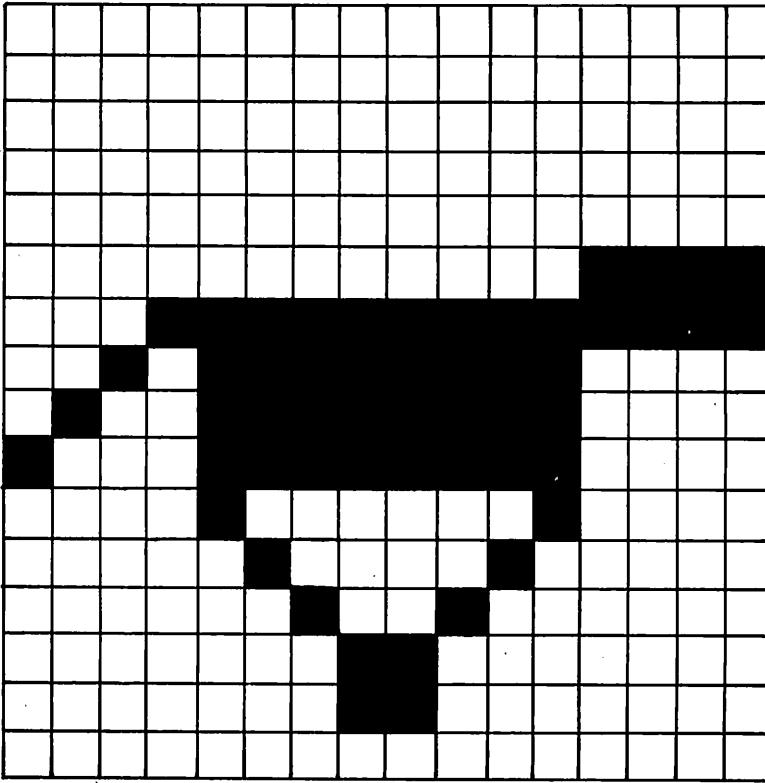
```
TO KILLSPRITE
 TELL :ALL SC 0
END
```

which changes the color of all sprites to 0 (clear), making them invisible. Or you can simply type **TELL :ALL SC 0**, and the effect will be the same.

Now let's have a little fun with sprites we design ourselves. First we've got to give our shapes a name and a number, which then represents the number of the sprite:

---





**Figure 15-2.** Another design, with the horse's legs, tail, and head in different positions.

design is complete, press BACK (FCTN 9), which returns you to the command mode. Now type:

**MS :HORSES**

and repeat the process with the second horse shape. When that is complete, it's time to devise the procedure:

```
TO GALLOP
SC 6
CARRY 6
FD 5
CARRY 7
FD 5
GALLOP
END
```

---

which sets the color as 6 (red) and then tells sprite 6 to move forward five spaces, after which sprite 7 moves forward five spaces and the next line (GALLOP) causes the routine to be repeated over and over again, giving the impression that the horse is galloping, moving its legs and swishing its tail up and down.

Now, if you want to make a horse race out of this, we can use the shapes we have created like this:

```
TO RACE
SC 6
CARRY 6 FD 3
CARRY 7 FD 3
TELL SPRITE 8 CARRY 6
TELL SPRITE 9 CARRY 7
TELL SPRITE 10 CARRY 6
TELL SPRITE 11 CARRY 7
TELL SPRITE 8 SY 5
TELL SPRITE 9 SY 5
TELL SPRITE 10 SY 10
TELL SPRITE 11 SY 10
TELL SPRITE 8 FD 5
TELL SPRITE 9 FD 5
TELL SPRITE 10 FD 7
TELL SPRITE 11 FD 7
RACE
END
```

This creates a total of six sprites, representing three horses moving at different speeds in the same direction.

The TI LOGO instruction book contains a number of additional ideas you might like to experiment with, after which you'll want to think up new procedures and shapes of your own.

## ***MUSICAL LOGO***

LOGO can also be used to play music, but for this it would be helpful, though not absolutely necessary, if you can read music. I cannot, so I won't create any actual music, but rather give you a simple example, using scales. The commands used are MUSIC and

---



PLAYMUSIC (PM), and in the three-plus octaves that LOGO offers in one of two music modes (CHROMATIC and MAJOR), each note is represented by a positive or negative number, all of which are listed on pages 128 and 132 of the LOGO manual. MUSIC requires two sets of inputs, the first set being the pitch of each note in a phrase, and the second representing the duration for which the note is held. Here is an example that plays a chromatic scale:

### MUSIC PROGRAM

This program plays a chromatic scale.

---

```
TO SCALE1
MUSIC [-15 -14 -13 -12] [2 2 2 2]
END
TO SCALE2
MUSIC [-11, -10 -9 -8] [2 2 2 2]
END
TO SCALE3
MUSIC [-7 -6 -5 -4] [2 2 2 2]
END
TO SCALE4
MUSIC [-3 -2 -1 0] [2 2 2 2]
END
TO SCALE5
MUSIC [1 2 3 4] [2 2 2 2]
END
TO SCALE6
MUSIC [5 6 7 8] [2 2 2 2]
END
TO SCALE7
MUSIC [9 10 11 12] [2 2 2 2]
END
TO SCALE8
MUSIC [13 14 15 16] [2 2 2 2]
END
TO SCALE9
MUSIC [17 18 19 20] [2 2 2 2]
END
TO SCALE10
MUSIC [21 22 23 24] [2 2 2 2]
END
TO PLAY
SCALE1 SCALE2 SCALE3 SCALE4 SCALE5 SCALE6 SCALE7
SCALE8 SCALE9 SCALE10
PLAY
END
```

---

With all that in the computer, turn up the volume on your monitor and type **PLAY**:

**PM**

and the entire range of musical notes available with **LOGO CHROMATIC** will be played over and over again. When you now want to stop the playing, you will find that pressing **BACK** or **AID** does not work. The only way to stop it is either to type **SETVOICE 0** or by entering the edit mode by typing **TO NAME**, where **NAME** can be anything. If you know anything about music, you can use such short blocks of notes to create all manner of melodies. All it takes is a bit of experimentation. As is true of the sprite mode, the music mode offers a number of commands:

**CHROMATIC** is the standard range of notes that we used in the example above. See also **MAJOR**.

**DRUM** creates a drum beat. It must be used with numeric inputs that determine the duration.

**LEGATO** controls pauses inserted between notes.

**LOOPMUSIC** causes music to be played over and over. To stop, use **SETVOICE 0** or enter edit mode. **LOGO** commands can be executed while the music is playing.

**MAJOR** changes the scales to full notes per unit rather than the half notes in **CHROMATIC**.

**MUSIC** is used with two sets of numeric expressions, where the first set controls the pitch and the second controls the duration.

**NOTE** places a single note into the buffer, with duration, pitch, and volume controlled by three digits (without square brackets): **NOTE 4 8 11**.

**PLAYNOTE** plays one note at a time, after which it waits for the duration of that note.

---

**PLAYMUSIC (PM)** plays the music stored in the buffer. While music is playing, other LOGO commands may be executed.

**REST** is used with one number to insert a pause, the length of which is determined by the number.

**SETTEMPO** is used with one number to set the tempo in counts per minute.

**SETVOICE** uses a single digit from 0 to 4 as input. Zero clears the music buffer; 1, 2, and 3 can be used to select one of three different voices, and 4 is used to generate noise.

**SETVOLUME** uses a single digit as input, ranging from 0 to 15, with 0 being the softest and 15 the loudest. The default value is 0.

**STACCATO** causes each note to be played for just 5/60 of a second, with dead pauses between notes. This contrasts with **LEGATO**, which is the default condition.

## **OTHER TI LOGO CAPABILITIES**

You can use TI LOGO to create so-called tiles, which are designs on an eight-by-eight grid, by using the **MAKECHAR (MC)** command. When used in conjunction with the numbers that represent the ASCII characters (see Appendix I), the command produces those characters. Or you can use numbers above 95 and create simple shapes of your own, or you can actually reshape the ASCII characters.

TI LOGO can also, of course, be used to write the type of programs that we normally write in BASIC. I believe, however, that such use is of secondary importance. Anyone wishing to use LOGO in that fashion will find reasonably clear instructions in the LOGO manual.

Finally, you will almost certainly want to save some of your LOGO shapes and procedures on disk or cassette. In the command mode, type **SAVE**, after which the display will ask whether you

---

want to save just the procedures, just the shapes (and tiles), or both. In most instances you'll want to select the latter. It then asks whether you're using a cassette, a diskette, or other. Type the appropriate number and the name of the procedure to be saved, and all will be recorded. Later on, if you want to reuse the same procedure along with the saved shapes, type `RECALL` and then follow the same routine, and your work will be transferred to the computer RAM.

---

---

# 16

---

## **Telecommunication: Modems and Data Banks**

---

---

One of the more fascinating aspects of computers is their ability to communicate with one another over telephone lines. The entire field of telecommunication is growing so rapidly at least in part because it provides computer users with a "window" to the entire world. Texas Instruments offers an optional *modem* that interfaces with your telephone. The word "modem" stands for *modulate/demodulate*, because the computer is a digital device whereas the telephone company uses analog technology to transmit. The analog technology consists of a variety of tones, similar to those you hear on pushbutton telephones, which control the electrical current and, in turn, the transmission. Since the digital computer uses only two symbols, zero and one, the modem translates these symbols into tones acceptable to telephone technology, and, at the receiving computer, another modem retranslates the analog tones into digital symbols. The speed at which all of this is accomplished is described in terms of *baud rate*, which is the number of bits transmitted in 1 second. The TI modem has a baud rate of 300, which means that it transmits approximately 30 letters or digits in each second.

Unfortunately, not every computer can "talk" to every other computer. The computer language used must be the same at both

---

ends, which means that, barring the availability of some relatively complicated translating devices, you'll only be able to communicate with TI-99/4A Home Computers or with data banks that are equipped to deal with the TI version of BASIC. The good news is that one of the leading data banks, The SOURCE (a subsidiary of Reader's Digest) has established something call TEXNET, which provides TI-99/4A users with access to the huge amount of information and services available from The Source.

The Source provides more than 1000 information and communication services that are instantly available to subscribers. Subscribing involves an initial fee plus a relatively small monthly fee plus charges based on the number of minutes you use in accessing the data bank during any given month. The aggregate charges are not too steep, but they can, of course, mount up with frequent use of the services. The time charges vary with the time of day, being higher during business hours and lowest in the middle of the night. Another consideration is the city or town in which you're located. The Source has local phone numbers available in most major cities, but if you live in a small community, you may have to use a number in a nearby city, and you'll have to add long-distance telephone charges to the other fees.

Before going into more detail about the ways in which telecommunication can be useful, let's look at the logistics of getting your computer "on line."

## **GETTING "ON LINE"**

The TI modem is what is known as an *acoustical modem*, meaning that it uses the telephone receiver to accomplish transmission and reception. The unit consists of a small board that contains the electronics and two rubber cups into which the telephone receiver is placed. The modem is attached to the same RS232 port as is used for the printer, and since only one port is available on the card, the printer must be disconnected while the modem is in use. This means that incoming data must be stored in the computer RAM or sent to the disk drive, to be printed later if a hard copy is desired. The modem receives its electrical power from a transformer that plugs into a regular wall outlet.

---

If you're going to communicate with someone in a distant location, try to do it at night when long-distance telephone rates are lowest, because you'll be billed at regular long-distance rates whenever you communicate through the telephone. Plan everything in advance to avoid wasting time while you're deciding what you want to do next.

Most of the time you'll probably use your modem to access The Source (or other such data banks if and when they become available to TI-99/4A users). To use the modem to contact The Source, you will need, in addition to the modem, a Terminal Emulator II module, which must be plugged into your computer. When you're ready, turn on the system in the usual manner, make sure the modem is set for FULL DUPLEX (F on the back of the unit) and OUTPUT (O on the back of the unit), and press any key to get to the main menu. That menu offers you several choices:

PRESS

- 1 FOR TI BASIC
- 2 FOR TERMINAL EMULATOR II
- 3 FOR DEFAULT OPTION TE II

For now, press 2. The display will change to:

TERMINAL EMULATOR II

and then to:

---

| PARAMETER    | OPTIONS                      | CHOICE |
|--------------|------------------------------|--------|
| BAUD RATE    | 1-300<br>2-110               | 1      |
| PARITY       | 1-EVEN<br>2-ODD<br>3-NONE    | 1      |
| DUPLEX       | 1-FULL<br>2-HALF             | 1      |
| RS-232 PORT  | 1-#1<br>2-#2                 | 1      |
| COLUMN WIDTH | 1-40<br>2-38<br>3-36<br>4-34 |        |
| AUTO LOG-ON  | FILE =                       | LOGON  |

with the cursor blinking on the first 1 under OPTION. In most instances you will be able to accept the default values by simply pressing >ENTER< for each item, though you might want to change the column width to 38 (type 2) in order to avoid wraparound of copy on the screen. When all that has been done, press >ENTER<, and a steady cursor will appear in the top left corner of the screen. Now dial the number that was given you by The Source, and when you hear the high-pitched tone, place the receiver into the cups of the modem, making sure that it is firmly in place. The display will change to L?, which indicates that you are now connected to The Source. Unless you've been given other instructions when signing up with The Source, you now need to type in the identifier code for your terminal, which is D1 >ENTER<. The screen now displays an @ sign, and you must type in the code for your system, which is most likely C 30128. The screen now tells



you that you are connected to The Source. You now type ID TI4917 PASSWORD, or whatever account number and password you were given when you signed up with The Source. At this point you're ready to ask for whatever information you're interested in, using the instructions provided in the voluminous instruction book that was sent you.

## **SERVICES OF THE SOURCE**

The type of services available through The Source are as useful to the family at home as they are to businesses or professional persons. You can do your shopping electronically by calling up categories of merchandise that interest you, and you'll be confronted by the best discounted prices along with information as to how to obtain your selection. In addition, you can trade and barter with other Source subscribers. You can ask for current airline schedules and rates for any city with airline service. There are restaurant guides available for most cities, and you can obtain advice on recipes, food and nutrition, wines, health care, medicine, home repair, or you can ask for reviews of the latest motion picture releases.

You can access a library of computer programs that are available free of charge. You can send letters, documents, or other material via The Source to be delivered immediately or delayed to one recipient or thousands. You can have access to a large number of games to be played at home, to a variety of educational programs in many languages. You have immediate access to the latest stock quotations, to the UPI news service, to business advice and programs. You can hold computer conferences, or you can simply chat with other Source subscribers. And, most of all, you can use The Source's research facilities to obtain information on virtually any subject imaginable. Anything that might be found in the latest edition of the best encyclopedia available is provided at a moment's notice, plus material and information dealing with specialized subjects that are usually not covered in most general-purpose research publications and therefore cannot be easily found.

---

You can even publish newsletters or an entire magazine via The Source electronically, without ever using paper or employing a printer. Your publication will be distributed to subscribers, and you'll receive a royalty.

At first glance, the very quantity of information and services being offered is somewhat intimidating. And, when contemplating the purchase of a modem, you might say to yourself that you don't really need any of this stuff. You may be right, but chances are that you're not, especially if your family includes school-age children, who always need answers to questions you may not be equipped to deal with.

Like just about anything associated with the growing field of computers, we must *do* in order to learn how best to make use of what is available to us. And that includes the field of telecommunication because, until you actually use the system, it's virtually impossible to obtain a clear picture of the advantages to be gained. My guess is that once a modem has been added to your system, you're likely to wonder how you ever managed to get along without it.

If you're at all interested in having the world at your fingertips, then, by all means, don't hesitate.

---



---

# Appendix I

---

## ***ASCII Character Codes***

---

The ASCII character codes produce the characters of the alphabet, digits, and a variety of symbols.

|    |                             |
|----|-----------------------------|
| 32 | (space)                     |
| 33 | ! (exclamation point)       |
| 34 | " (quotation mark)          |
| 35 | # (number or pound)         |
| 36 | \$ (dollar or string)       |
| 37 | % (percent)                 |
| 38 | & (ampersand)               |
| 39 | ' (apostrophe)              |
| 40 | ( (parenthesis open)        |
| 41 | ) (parenthesis close)       |
| 42 | * (asterisk)                |
| 43 | + (plus)                    |
| 44 | , (comma)                   |
| 45 | — (minus or hyphen)         |
| 46 | . (period or decimal point) |
| 47 | / (slash or divide)         |
| 48 | 0 (zero)                    |
| 49 | 1                           |
| 50 | 2                           |
| 51 | 3                           |

|    |                   |
|----|-------------------|
| 52 | 4                 |
| 53 | 5                 |
| 54 | 6                 |
| 55 | 7                 |
| 56 | 8                 |
| 57 | 9                 |
| 58 | : (colon)         |
| 59 | ; (semicolon)     |
| 60 | < (less than)     |
| 61 | = (equals)        |
| 62 | > (greater than)  |
| 63 | ? (question mark) |
| 64 | @ (at)            |
| 65 | A                 |
| 66 | B                 |
| 67 | C                 |
| 68 | D                 |
| 69 | E                 |
| 70 | F                 |
| 71 | G                 |
| 72 | H                 |
| 73 | I                 |
| 74 | J                 |
| 75 | K                 |
| 76 | L                 |
| 77 | M                 |
| 78 | N                 |
| 79 | O                 |
| 80 | P                 |
| 81 | Q                 |
| 82 | R                 |
| 83 | S                 |
| 84 | T                 |
| 85 | U                 |
| 86 | V                 |
| 87 | W                 |
| 88 | X                 |
| 89 | Y                 |

---

- 90    Z
  - 91    [ (bracket open)
  - 92    \ (reverse slash)
  - 93    ] (bracket close)
  - 94    ^ (exponent)
  - 95    — (underline)
-

---

# Appendix II

---

## **Speech Synthesizer Word List**

---

The TI Solid State Speech Synthesizer, Model PHP 1500, has a resident vocabulary of 373 words that the voice will pronounce. All words not included in this list, when used with the speech synthesizer, will be spelled.

A (long a)  
A1 (short a)  
ABOUT  
AFTER  
AGAIN  
ALL  
AM  
AN  
AND  
ANSWER  
ANY  
ARE  
AS  
ASSUME  
AT

B  
BACK  
BASE  
BE  
BETWEEN  
BLACK  
BLUE  
BOTH  
BOTTOM  
BUT  
BUY  
BY  
BYE

**C**

CAN  
CASSETTE  
CENTER  
CHECK  
CHOICE  
CLEAR  
COLOR  
COME  
COMES  
COMMA  
COMMAND  
COMPLETE  
COMPLETED  
COMPUTER  
CONNECTED  
CONSOLE  
CORRECT  
COURSE  
CYAN

**D**

DATA  
DECIDE  
DEVICE  
DID  
DIFFERENT  
DISKETTE  
DO  
DOES  
DOING  
DONE  
DOUBLE  
DOWN  
DRAW  
DRAWING

**E**

EACH  
EIGHT  
EIGHTY  
ELEVEN  
ELSE  
END  
ENDS  
ENTER  
ERROR  
EXACTLY  
EYE

**F**

FIFTEEN  
FIFTY  
FIGURE  
FIND  
FINE  
FINISH  
FINISHED  
FIRST  
FIT  
FIVE  
FOR  
FORTY  
FOUR  
FOURTEEN  
FOURTH  
FROM  
FRONT

**G**

GAMES  
GET  
GETTING

---



GIVE  
GIVES  
GO  
GOES  
GOING  
GOOD  
GOOD WORK  
GOODBYE  
GOT  
GRAY  
GREEN  
GUESS

H  
HAD  
HAND  
HANDHELD UNIT  
HAS  
HAVE  
HEAD  
HEAR  
HELLO  
HELP  
HERE  
HIGHER  
HIT  
HOME  
HOW  
HUNDRED  
HURRY

I  
I WIN  
IF  
IN  
INCH  
INCHES

INSTRUCTION  
INSTRUCTIONS  
IS  
IT

J  
JOYSTICK  
JUST

K  
KEYBOARD  
KNOW

L  
LARGE  
LARGER  
LARGEST  
LAST  
LEARN  
LEFT  
LESS  
LET  
LIKE  
LIKES  
LINE  
LOAD  
LONG  
LOOK  
LOOKS  
LOWER

M  
MADE  
MAGENTA  
MAKE  
ME  
MEAN

---

MEMORY  
 MESSAGE  
 MESSAGES  
 MIDDLE  
 MIGHT  
 MODULE  
 MORE  
 MOST  
 MUST

N  
 NAME  
 NEAR  
 NEED  
 NEGATIVE  
 NEXT  
 NICE TRY  
 NINE  
 NINETY  
 NO  
 NOT  
 NOW  
 NUMBER

O  
 OF  
 OFF  
 OH  
 ON  
 ONE  
 ONLY  
 OR  
 ORDER  
 OTHER  
 OUT  
 OVER

P  
 PART  
 PARTNER  
 PARTS  
 PERIOD  
 PLAY  
 PLAYS  
 PLEASE  
 POINT  
 POSITION  
 POSITIVE  
 PRESS  
 PRINT  
 PRINTER  
 PROBLEM  
 PROBLEMS  
 PROGRAM  
 PUT  
 PUTTING

Q

R  
 RANDOMLY  
 READ (reed)  
 READ (red)  
 READY TO START  
 RECORDER  
 RED  
 REFER  
 REMEMBER  
 RETURN  
 REWIND  
 RIGHT  
 ROUND

---

S

SAID  
SAVE  
SAY  
SAYS  
SCREEN  
SECOND  
SEE  
SEES  
SET  
SEVEN  
SEVENTY  
SHAPE  
SHAPES  
SHIFT  
SHORT  
SHORTER  
SHOULD  
SIDE  
SIDES  
SIX  
SIXTY  
SMALL  
SMALLER  
SMALLEST  
SO  
SOME  
SORRY  
SPACE  
SPELL  
SQUARE  
START  
STEP  
STOP  
SUM  
SUPPOSED  
SUPPOSED TO  
SURE

T

TAKE  
TEEN  
TELL  
TEN  
TEXAS INSTRUMENTS  
THAN  
THAT  
THAT IS INCORRECT  
THAT IS RIGHT  
THE (thee)  
THE1 (the, short e)  
THEIR  
THEN  
THERE  
THESE  
THEY  
THING  
THINGS  
THINK  
THIRD  
THIRTEEN  
THIRTY  
THIS  
THREE  
THREW  
THROUGH  
TIME  
TO  
TOGETHER  
TONE  
TOO  
TOP  
TRY  
TRY AGAIN  
TURN  
TWELVE  
TWENTY

---

TWO  
TYPE

U  
UHOH  
UNDER  
UNDERSTAND  
UNTIL  
UP  
UPPER  
USE

V  
VARY  
VERY

W  
WAIT  
WANT  
WANTS  
WAY  
WE  
WEIGH  
WEIGHT  
WELL  
WERE  
WHAT  
WHAT WAS THAT  
WHEN  
WHERE  
WHICH  
WHITE  
WHO  
WHY  
WILL  
WITH  
WON  
WORD

WORDS  
WORK  
WORKING  
WRITE

X

Y  
YELLOW  
YES  
YET  
YOU  
YOU WIN  
YOUR

Z  
ZERO  
0 (zero)  
1 (one)  
2 (two)  
3 (three)  
4 (four)  
5 (five)  
6 (six)  
7 (seven)  
8 (eight)  
9 (nine)  
- (negative)  
+ (positive)  
. (point)

---

---

# **Glossary of Computer Terms and Abbreviations**

---

This glossary includes the terms, abbreviations, and commands used in this book. Other frequently encountered computer terms and abbreviations are also included.

## **A**

**ACCEPT** — a statement and command used in TI EXTENDED BASIC that stops program execution until data have been keyed in. Similar to INPUT, except that it permits input data to be displayed at a certain location on the screen.

**Accumulator** — a word for memory or register, derived from the fact that subtotals are accumulated and retained in memory.

**Address** — the number or phrase used to recall previously stored data from the computer memory.

**Address register** — the register in the central processing unit that contains the address of the program in use.

---

**ADP** — automatic data processing; in other words, the activity of the computer itself.

**ALGOL** — a computer language that is best suited to the processing of mathematical and numerical problems. The acronym stands for ALGOrithmic Language.

**Alphanumeric** — a combination of alphabetic and numeric data.

**Analog** — one of two methods of solving problems electronically. In the analog method electrical signals are of variable voltages, and the slightest change may cause a significant difference in output. The word “analog” is derived from analogy. The alternative method of electronic problem solving is referred to as digital.

**Analog computer** — analog computers use electrical current fluctuations to represent data. Analog computers, once predominant in the computer field, have lost favor since the introduction of the microprocessor and the digital computer.

**ANSI** — American National Standards Institute, an organization that studies computer languages and codes in an effort to bring about a degree of standardization.

**AOS** — arithmetic operating system, the conventional system of mathematical problem solving that we use when figuring with pencil and paper. (See also RPN.)

**APL** — a programming language. The acronym stands for just that: A Programming Language.

**Applications program** — a program designed to solve a specific type of problem, in contrast to systems programs, such as CP/M, designed to control the operation of the computer itself.

**ARRAY** — a list of variables designed for quick and easy access by the computer.

**ASCII** — American Standard Code for Information Interchange. It is a code that represents the assembly and machine language

---

substitutes for upper- and lowercase letters, numbers, punctuation marks, mathematical function symbols, and so on. It is the standard used by most microcomputers.

**Assembler** — a program that translates assembly language into machine language.

**Assignment statement** — permits programmers to assign given numeric, alpha, or alphanumeric data to letters, letter combinations, or other symbols, such as  $A=6$  or  $B=5/(1+8)$  or  $A\$="WORD."$  Subsequently these symbols can be used in place of the data that have been assigned to them.

**Asterisk (\*)** — the sign used by computers to represent the command to multiply. It is used in order to permit the computer to differentiate between the multiplication command and the letter X, conventionally used to denote multiplication.

## B

**BASIC** — the most popular programming language. It's popularity is based on the fact that it is relatively easy to learn and use. It permits the use of simple and largely self-explanatory phrases to enter complicated instructions when writing programs. The acronym stands for Beginners' All-purpose Symbolic Instruction Code.

**Baud** — a unit of the speed with which information and data are transmitted. Thus, a baud rate of 5000 means that data are transmitted at a rate of 5000 bits per second.

**Binary** — represents the way in which computers use data. Computers can use only two digits, 0 and 1, representing the state of the electronic circuits: on and off. All data entered into the computer, numbers, letters, symbols or whatever, must be translated into machine language, consisting entirely of combinations of 0s and 1s. Here is a brief list of decimal numbers translated into their binary equivalents:

|   |       |   |       |    |       |    |       |
|---|-------|---|-------|----|-------|----|-------|
| 0 | 00000 | 5 | 00101 | 9  | 01001 | 13 | 01101 |
| 1 | 00001 | 6 | 00110 | 10 | 01010 | 14 | 01110 |
| 2 | 00010 | 7 | 00111 | 11 | 01011 | 15 | 01111 |
| 3 | 00011 | 8 | 01000 | 12 | 01100 | 16 | 10000 |
| 4 | 00100 |   |       |    |       |    |       |

---

**Bit** — the smallest possible unit of information. One bit is sufficient to tell the difference between 0 and 1, which may represent yes or no, right or wrong, and so on. Computers deal with all information in the form of individual bits.

**Block diagram** — a graphic representation of a program. In program development it often helps to clarify what the program is to accomplish at any given stage by preparing a block diagram, also referred to as a flowchart.

**Boot** — stands for “bootstrap,” derived from “pulling yourself up by your own bootstrap.” It is, in fact, a short and simple program that prepares the computer to accept information, or permits information from disk to be loaded into the computer. Computers are described as supporting “warm boot” and “cold boot” routines, meaning the manner in which a disk can be activated. Cold boot means that the disk is loaded into the disk drive with the power to the computer turned off. After it is in the drive, the computer is turned on and the disk drive is activated automatically. Warm boot means that the power to the computer is on when the disk is placed into the disk drive. Then a key or a combination of keys is pressed, and the drive is activated. Either way, the disk in question must contain a boot program, such as the TI initialization program.

**Branch** — a detour in a program, usually as the result of a conditional statement, such as IF A = 0 THEN GOTO and the line number to which the computer is supposed to go, assuming that A equals 0. If it doesn't the computer will ignore the command and go to the next program line. At the end of such a branch there must be an additional instruction that tells the computer either to return to the line just past the previous place in the program, or to some other line.

**BREAK** — A command or statement in TI BASIC that stops program execution.

**Buffer** — The temporary storage area for data, limited by the number of Ks in the computer's random-access memory (RAM). Anything typed into the computer, whether text or programs, is first stored in the memory buffer, which, after the information has been saved on disk, can be cleared to accept additional data.

---



**BYE** — a command in TI BASIC, terminating program execution.

**Byte** — any unit of information that consists of 8 bits, such as most individual letters or digits. Storage space in memory or on disk is usually referred to in terms of K bytes, with one K byte representing 1024 bytes.

## C

**Calculator mode** — the operating mode in which the computer accepts direct commands and performs them immediately. Thus, if `PRINT 2 + 2?` is typed in with no line number, the computer instantly responds with 4. Also referred to as “immediate execution mode.”

**CALL** — a statement in TI BASIC and TI EXTENDED BASIC that calls up any one of a large number of built-in subprograms.

**Cathode-ray tube (CRT)** — the video display of the monitor or TV set.

**Character** — any character, such as a letter, digit, mathematical function symbol, and so on.

**Chip** — an integrated circuit, or the tiny amount of silicon that makes up part of an integrated circuit.

**COBOL** — a programming language that is particularly suited to solving business-related problems. It is particularly adept at handling random-access files. The acronym stands for COmmon Business-Oriented Language.

**Column** — each individual character space on a program or text line. The current position of the cursor is indicated by line and column numbers.

**Compiler** — a program that translates the data, information, and commands entered in one of the common computer languages (high-level languages) into the kind of language the computer can deal with.

---

**Computer error** — there really is no such thing. Whenever people blame some kind of a goof on “computer error,” they are usually simply trying to hide some human error that was made while interacting with the computer.

**Computer languages** — languages the computer can accept because they were designed to be translated automatically into machine language.

**Conditional statement** — a statement in a computer program that causes the computer to execute one of several possible steps, depending on whether the condition contained in the statement is met (such as IF A=0 THEN GOSUB. . .).

**Constant** — a numeric or alphanumeric expression that remains unchanged. The opposite of a variable.

**CONTINUE** or **CON** — a command in TI BASIC that causes the computer to continue program execution after a BREAK command caused it to interrupt execution.

**cps** — characters per second. Used frequently to express the speed with which line printers print text.

**CPU** — central processing unit, the heart of the computer.

**CRT** — cathode-ray tube.

**Cursor** — the usually flickering mark that appears on the video display to indicate where the next printed character will appear.

## D

**Data** — any type of information, commands, letters, numbers, symbols, or graphics.

**DATA** — a statement in BASIC that permits the entry of any number of alphanumeric data in programs. These data are then accessed by using the READ statement.

---

**Data-base program** — a program designed to accumulate data of one kind or another, and from which these data can be retrieved at will in random fashion. Data-base programs are, in fact, random-access file programs designed to make data entry easy.

**Data field** — an area within a data form in a data-base program designed to accept entry of data.

**Data file** — a file created automatically during program execution in which data are stored in the order in which they were keyed into the computer. Data files can be sequential files or random-access files.

**Data link** — a means of electronically transferring data from one location to another. Data links can be telephone lines, radio waves, microwaves, coaxial cables, or laser beams.

**Data processing** — the manipulation of data; what the computer does.

**Debugging** — editing and correcting a program.

**Decimal** — the fraction represented by digits to the right of the decimal point.

**Decimal system** — the conventional arithmetic system, using numbers from zero to nine.

**DEF** — a TI EXTENDED BASIC statement that allows you to define your own functions.

**Default** — most computers are preprogrammed to a certain degree, causing them to assume that certain standard commands will be executed unless the operator keys in commands to the contrary.

**DELETE** — a TI BASIC command used to delete files from disk.

**Dialect** — computer languages are used by different computers with minor changes. Thus TI BASIC and TI EXTENDED BASIC are

---

dialects of BASIC that are accented by TI-99/4A computers. They include some commands that are not supported by other computer models.

**Digital** — using numbers rather than analog-type representations in the manipulation of data.

**Digital computer** — computers that operate by counting the ON/OFF sequences in electrical currents. All microcomputers are digital computers.

**DIM** — a TI BASIC command used to dimension arrays.

**Disk** — a thin disk of magnetic material that can be used for the storage of large amounts of data. There are so-called floppy disks, and hard disks.

**Disk drive** — the mechanical system designed to store data on disk or to retrieve data from disk.

**Diskette** — often used to describe 5.25-inch disks.

**DISPLAY AT** — a TI BASIC statement that causes data to be displayed at predetermined positions on the screen.

**Duplex** — a communication line that permits sending and receiving information simultaneously, such as an ordinary telephone line.

## E

**Edit mode** — a means of editing program lines without retyping the entire line. Type the line number and press FCTN X to enter edit mode.

**EDP** — electronic data processing.

**END** — a BASIC statement indicating the end of the program. It does not necessarily have to be the last program line.

---

**Error message** — any message displayed by the computer to tell the operator that he or she has made a mistake, or that something has gone wrong. Error messages are usually associated with information describing the error in question.

**Executive program** — a program designed to handle the management of the computer system.

**Expression** — a program statement.

## F

**File** — a collection of pieces of information, data, or text stored either in the computer's memory buffer or on disk. Files must be stored under a unique file name in order to facilitate retrieval.

**Fixed-length record files** — an expression denoting files in which each record consists of the identical number of bytes.

**Floppy disk** — see disk.

**Flow chart** — see block diagram.

**FOR. . .TO. . .(STEP)** — the BASIC statement that creates a loop.

**Formatting** — the term refers to the action of the computer when, in compliance with instructions given by the operator, it rearranges text or data into a predetermined format for display or printing.

**FORTRAN** — a high-level language best suited for the handling of algebraic problems, allowing exponentiation of up to three subscripts. The acronym stands for FORMula TRANslation.

**Function keys** — keys on the keyboard that perform certain predetermined functions rather than typing a letter, digit or symbol. Since many of the more sophisticated programs require the availability of dozens of unique functions, they usually use combinations of two or even three keys to execute functions.

---

## G

**Garbage** — incorrect information.

**GIGI** — *garbage in, garbage out*. An expression used to emphasize that faulty input information will produce faulty output information.

**Global search** — looking for a given piece of information, data, or text anywhere within the program or file.

**GOSUB** — a BASIC statement used in conjunction with a line number that tells the computer to go to a subroutine. The subroutine must end with a RETURN statement, telling the computer to go back where it came from.

**GOTO** — a BASIC command used to tell the computer to go to a line in the program other than the next line.

## H

**Hard copy** — printouts produced by the line printer.

**Hardware** — all actual equipment, including all peripherals, that makes up a complete computer system.

**Hexadecimal** — an arithmetic system using 16 digits. It is often used in computer language translations. It is used in TI BASIC to determine the shapes of graphic characters. Here is a list of 16 characters:

|   |   |   |   |    |   |    |    |
|---|---|---|---|----|---|----|----|
| 0 | 0 | 5 | 5 | 9  | 9 | 13 | D  |
| 1 | 1 | 6 | 6 | 10 | A | 14 | E  |
| 2 | 2 | 7 | 7 | 11 | B | 15 | F  |
| 3 | 3 | 8 | 8 | 12 | C | 16 | 10 |
| 4 | 4 |   |   |    |   |    |    |

**High-level language** — all programming languages other than assembly and machine language. Also referred to as procedure-oriented languages (POLs).

---

## I

**IF...THEN...(ELSE)** — in BASIC, a set of protected words used in conditional statements: IF A = 10 THEN. . .(ELSE. . .)

**IMAGE** — a TI EXTENDED BASIC statement used to specify the format in which numbers are to be displayed or printed.

**INPUT** — a BASIC statement that tells the computer to stop program execution until certain data have been keyed in. With TI-99/4A computers it can be used in conjunction with a prompt: INPUT "DATA? ";A\$, which would assign the keyed-in data to the symbol A\$.

**Integer** — any number stripped of its decimals; the integer of 123.456 is 123.

**Interactive** — a program that includes questions and prompts that must be acted upon by the operator.

**Interface** — any device that connects two pieces of hardware, causing them to be compatible.

**Interpreter** — a machine-language program that understands, interprets, and executes programs written in a high-level language.

**I/O** — input/output.

## K

**K** — short for kilo. Even though kilo usually stands for 1000, in the world of computers, K stands for 1024 (bytes).

**Keyboard** — the arrangement of keys that permits communication between the operator and the computer.

**Kilobaud** — 1000 baud.

---

## L

**Line printer** — any type of printer that uses 8.5-inch (plus 1 inch for the perforations) or wider paper. Line printers are either dot matrix printers or daisywheel printers.

**LINPUT** — a TI EXTENDED BASIC statement that assigns an entire line or record to a string variable.

**LIST** — a BASIC command that causes the computer to display the specified program lines on the CRT.

**Local search** — tells the computer to look for a given piece of information, data, or text only within a certain portion of the program.

**Logic** — an arithmetic method, such as AOS or RPN.

**Loop** — a portion of a program that is repeated time and again until a certain condition is met, such as:

```
10 FOR X=1 TO 100
20 A=A+1
30 NEXT X
```

where A will increase by 1 until it equals 100, after which program execution continues.

## M

**M** — mega or 1,000,000. One megabyte equals 1 million bytes.

**Machine language** — the only language the computer can understand without prior translation. It consists entirely of 0s and 1s.

**Memory** — all those cubbyholes in which the computer stores data, information, text, and commands.

**MERGE** — a TI BASIC command that tells the computer to use data recorded in several different files.

---



**Microcomputer** — the expression has nothing to do with size. It refers to computers based on microprocessor technology.

**Microprocessor** — the real magical heart and guts of the computer. It is a sophisticated version of an integrated circuit that is capable of performing most computer functions. Its development by Texas Instruments made today's personal computer possible.

**Microsecond** — one-millionth of a second.

**Millisecond** — one-thousandth of a second.

**Modem** — a piece of hardware needed for two computers to communicate with one another over telephone lines. The acronym stands for *modulate/demodulate*, because the computer data are modulated into sound, similar to the sounds generated by pushbutton telephones, and, on the other end, demodulated again into computer commands.

## N

**n,nn,nnn** — used frequently in instruction books to denote one-, two-, or three-digit numbers.

**NEXT** — the BASIC statement that is part of the loop sequence.

**Nonvolatile memory** — a computer or calculator memory that retains stored information even when the computer or calculator is turned off.

**NUMBER** or **NUM** — a TI BASIC command that automatically numbers program lines.

## O

**Octal** — an arithmetic system using only eight digits:

|   |    |   |    |    |    |    |    |
|---|----|---|----|----|----|----|----|
| 0 | 00 | 5 | 05 | 9  | 11 | 13 | 15 |
| 1 | 01 | 6 | 06 | 10 | 12 | 14 | 16 |
| 2 | 02 | 7 | 07 | 11 | 13 | 15 | 17 |
| 3 | 03 | 8 | 10 | 12 | 14 | 16 | 20 |
| 4 | 04 |   |    |    |    |    |    |

---

**OLD** — the TI BASIC command to load files from disk into the computer RAM: OLD DSK1.FILENAME.

**ON** — a portion of several TI BASIC statements that tells the computer to execute certain steps if certain conditions exist: ON (numeric variable) GOTO or GOSUB. Also ON BREAK. . ., ON ERROR. . ., ON WARNING.

**OPEN#** — a TI BASIC statement that accesses peripheral devices or opens data files.

**OPTION BASE** — the TI BASIC statement that controls the lowest subscript in arrays.

**Output** — computer results. Opposite to input.

**Overflow** — when a program or text file is too long to be handled by the available K, the balance must be handled and recorded separately. When a computation involves more digits than the computer can handle, the result is rounded off, though the true result may still be available in the computer memory.

## P

**Page** — a screenful of information. Has no relation to the average 8.5-by-11 piece of paper.

**Parsing** — scanning a string to look for its individual components.

**Pascal** — a structured high-level language named after the noted French scientist.

**Peripheral** — any device connected to the computer itself, such as the monitor, line printer, disk drives, and so on.

**Polish notation** — see RPN.

**PRINT** — the BASIC statement that causes data to be displayed on the screen. To send data to a line printer, it must first be accessed using OPEN #, and the PRINT # statement must then be used.

---

**Procedure** — subroutine or similar self-contained program segments.

**Program** — a sequence of instructions designed to permit the computer to perform a given task repeatedly, using different variables.

**Programming** — designing and writing a program.

**Protected word** — a word, phrase, or combination of letters that has a distinct meaning to the computer. All high-level languages consist of such protected words.

## R

**RAM** — random-access memory, the memory buffer available for the temporary storage of keyed-in data. Its size is measured in K.

**Random-access file** — a file that can accept data in any order, which can subsequently be retrieved by subject matter as requested by the operator.

**Random-access memory** — see RAM.

**RANDOMIZE** — the TI BASIC statement that assures variety in random-number generation.

**READ** — the BASIC statement used to retrieve data stored in DATA lines within a program.

**Read-only memory** — see ROM.

**Record** — a TI BASIC expression referring to the combination of data fields that contain the information relative to one subject.

**Register** — a specific location within a random-access memory.

**Relative file** — the name used by TI computers for random-access files.

---

**REM** — the BASIC statement denoting remark lines that are ignored by the computer.

**RESEQUENCE** or **RES** — the TI BASIC command that causes line numbers to be rearranged automatically.

**RESTORE** — the BASIC statement that is used to assure that DATA items are READ from the top. In TI BASIC it can also be used with files to have a file read from the beginning.

**RETURN** — the BASIC statement denoting the end of a subroutine.

**ROM** — read-only memory. Prestored program information that controls the manner in which the computer functions. The ROM of the average computer cannot be accessed by the operator.

**Round-off error** — the cumulative mathematical error that results when fractions are rounded off.

**RPN** — Reverse Polish Notation, named after the Polish mathematician Jan Lukasiewicz, is an arithmetic system that does not support equal signs or parentheses. Data must be entered into the computer in a given order, after which the computer is told what to do with those data. Thus:

$$2 + 2 = 4$$

would, in RPN, look like

2 ENTER 2 +

and the displayed result would again be 4. Certain calculators, like the HP-41CV, use RPN.

**RUN** — the BASIC command that starts program execution. It can be used with or without line numbers.

---

## S

**SAVE** — the BASIC command that causes programs or data to be recorded on disk or tape: SAVE DSK1.FILENAME.

**Screen-oriented program** — a program that displays all the necessary prompts to tell the operator what to do next in order to obtain the desired result.

**Simplex** — a communication transmission system that permits either transmission or reception, but not both at the same time.

**SIZE** — a TI BASIC command that displays the number of unused bytes in RAM.

**Software** — computer programs. The term usually refers to programs that are marketed commercially.

**Sprite** — a TI expression for graphic shapes.

**Storage** — memory.

**STOP** — same as END.

**String** — any sequence of alpha or numeric characters that is treated by the computer as a single unit. Digits can be used as strings, but then they cannot be used for mathematical calculations.

**Subroutine** — a separate part of the program that may have to be run repeatedly during the execution of a program. Subroutines are entered as the result of a GOSUB statement and must always terminate with a RETURN command.

**Syntax** — the rules that govern the way in which the computer accepts commands and other data. When these rules are violated by the operator, the display responds with SYNTAX ERROR or some other error message.

**System** — the computer and all its peripherals.

---

## T

**Terminal** — any of the means by which the computer outputs data and information, the monitor screen, the line printer, the disk drives, modems, and so on.

**Thermal printer** — a printer that uses specially prepared paper to print by impact without a ribbon.

**Toggle key** — a key or combination of keys that reverse their effect each time the command is keyed in, such as on/off, up/down, alpha/numeric, and so on.

**TRACE** — the BASIC command that displays program line numbers when a program is being run. To reverse the process, use UNTRACE.

**Truncation** — when there are too many digits in a number, the excess is chopped off and the last digit rounded off.

## U

**UNTRACE** — turns off the TRACE action.

## V

**Variable** — the data that must be keyed in anew each time the program is executed.

**Volatile memory** — a memory capable of retaining stored data only as long as the electrical current is not interrupted. Interruptions cause all stored data to be lost. Microcomputers have volatile memories.

## W

**Window** — a portion of the display that is reversed for the display of input data when other portions are occupied by fixed data such as help menus.

**Word** — the number of bits a computer can deal with at one time. Microcomputers usually have an 8-bit or 16-bit limit.

---

---

# Computer Magazines

---

There are a number of magazines being published, all of which are devoted entirely to subjects related to microcomputers, hardware, and software, and all of which are filled with articles, evaluations, and advertisements for all conceivable computer-related products.

Since it is often difficult to find these magazines at the average newsstand or magazine rack, here are the names and addresses of the ones that are most important and informative:

**99er Homecomputer Magazine**, P.O. Box 5537, Eugene, OR 97405

**Personal Computing**, Hayden Publishing Company, Inc., 50 Essex Street, Rochelle Park, NJ 07662

**Popular Computing**, Byte Publications, Inc., 70 Main Street, Peterborough, NH 03458

**Creative Computing**, Ahl Computing, Inc., a subsidiary of Ziff-Davis Publishing Company, 39 East Hanover Avenue, Morris Plains, NJ 07950

**Computel**, Small Systems Services, Inc., P.O. Box 5406, Greensboro, NC 27403

---

**Interface Age**, McPheters, Wolfe and Jones, 16704 Marquardt Avenue, Cerritos, CA 90701

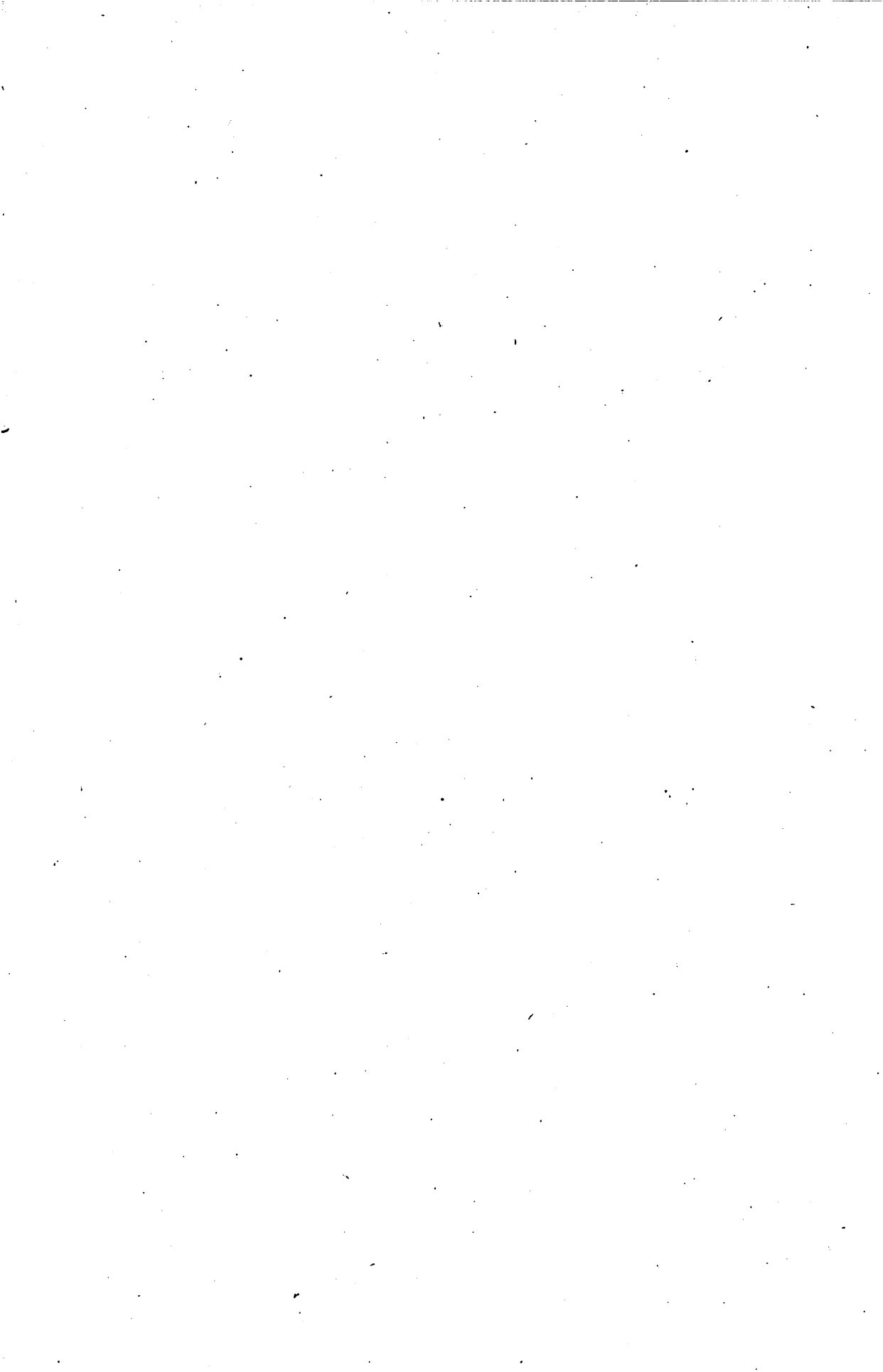
**Microcomputing**, Wayne Green, Inc., 80 Pine Street, Peterborough, NH 03458

**Byte**, Byte Publications, Inc., a subsidiary of McGraw-Hill, Inc., 70 Main Street, Peterborough, NH 03458

There are others, but by wading through the hundreds of pages of just some of the above, you'll be able to find whatever you may be looking for.

---





---

# *Index*

---

- Addition & subtraction I program  
(commercial), 49, 50
  - Addition & subtraction II program  
(commercial), 51
  - Advertising cost analysis program,  
360-366
  - Advertising, direct mail program,  
367-370
  - Airplane game program, 329-335
  - Alligator mix math program (commercial), 56
  - Alpiner game program (commercial),  
58, 59
  - Arithmetic, simple, program, 215-221
  - Arithmetic with voice program,  
244-249
  - Arrays, 140-147
  - ASCII character codes, 427-429
  - Authors and their works program,  
268-273
  
  - BASIC, 3
  - BASIC commands and statements,  
86-124
  - Baud rate, 420
  - Budget profit/loss analysis program,  
376-380
  
  - CALL CHAR, 285-288
  - CALL COINC, 304-306
  - CALL COLOR, 288-292, 302, 303
  - CALL commands, 107-124
  - CALL DELSPRITE, 300, 301
  - CALL DISTANCE, 303, 304
  - CALL HCHAR, 280-283
  - CALL LOCATE, 302
  - CALL MAGNIFY, 297, 298
  - CALL MOTION, 292-294, 298, 299
  - CALL PATTERN, 301
  - CALL SCREEN, 288-292
  - CALL SHAPE, 294, 295
  - CALL SPRITE, 295-297
  - CALL VCHAR, 283-285
  - Central processing unit, 3
  - Character definition, graphics,  
109-112
  - Checkbook keeping program, 155-161
  - Children's counting game program  
(graphics), 273-279
  - Comma (,), 25
  - Commands and statements, TI BASIC  
& EXTENDED BASIC, 86-102
  - Commands and statements, TI  
EXTENDED BASIC only,  
102-107
  - Compound interest, 136
-

- Computer math games program (commercial), 54, 55
  - Computer revolution, 1
  - Computer system, 2
  - Conversion to metric program, 258-268
  - Cookbook, electronic, program, 184-194
  - CPU, 3
  - Currency conversion program, 346-349
  - Cursor, 22
  
  - Data-base program, 126-132
  - Day-of-the-week program, 162-167
  - Deferred mode, 30
  - Defined functions, 381-389
  - Defined functions program, 382-386
  - Density altitude, 136, 137
  - Dice game, no voice (program), 312-315
  - Dice game with voice (program), 307-312
  - Digital, 3
  - Direct mail advertising program, 367-370
  - Derived functions, 381-389
  - Diskettes, 8
  - Disk, backup, 15
  - Disk catalog, 15
  - DISK FULL, 15
  - Disk, initializing, 9
  - Disk Manager module, 8
  - Disk rearranging contents, 18, 19
  - Disk, recording programs, 14
  - Disks, 8
  - Disk drive installation, 5, 6
  
  - Early reading and reading fun programs (commercial), 52, 53
  - Electronic cookbook program, 184-194
  - Extended memory card, 6
  
  - Floppies, 8
  - Floppy disks, 8
  - Formulas, standard calculation, 134-137
  
  - Garbage in, garbage out, 2
  - Graphics commands, 109-112
  
  - History lesson program, 254-258
  - Household budget program, 209-213
  - Household budget-management program (commercial), 46, 47
  
  - Immediate mode, 20
  - Initializing disks, 9
  - Investment, loan, savings program, 336-346
  - Invoice writer program, 349-354
  
  - Joysticks, 6
  
  - K, 5
  - Keyboard, 4
  - Kitchen timer program, 206-209
  
  - Ledger printing program, 354-360
  - Line printers, general, 61
  - Line printer, TI99/4A Impact Printer, 63, 64, 65
  - Loan, savings, investment program, 336-346
  - LOGO, 397-419
  - LOGO music, 415-418
  - LOGO sprites, 409-415
  - LOGO turtle graphics, 397-409
-

Metric conversion program, 258-268  
Microprocessors, 3  
Model railroad scale conversion program, 315-325  
Modem, 6, 420-424  
Monitor, 3  
Music, LOGO, 415-418

Name and address list program, 176-184  
New product introduction program, 370-376  
No-wordprocessor program, 82-85  
Numerology program, 390-395  
Numbers game program, 325-328  
Numeric variables, 28

Object recognition game program (graphics), 273-279  
Oshkosh, a game with aircraft (program), 328-335

Peripheral extension unit, 5  
Personal record-keeping program (commercial), 39-45  
Presidents of the U.S. program, 227-235  
Product introduction program, 370-376  
Profit/loss analysis program, 376-380  
Prompt, 21  
Protected words, spacing, 37

Quotation marks, 24  
Quotation marks within quotation marks, 27

RAM, 3, 5, 6  
Random-access file programs, 137-140  
Random-access memory, 3  
Rate of exchange, currency conversion program, 346-349  
Rearranging contents of disks, 18, 19  
Revolution, computer, 1  
Read-only memory, 3  
ROM, 3

Savings, loan, investment program, 336-346  
Schedule C Tax program, 194-206  
Scholastic spelling program (commercial), 53, 54  
Scrambled words program, 249-254  
Securities analysis program (commercial), 48  
Semicolon (;), 25  
Sequential file programs, 137-140  
Simple arithmetic program, 215-221  
Size/price comparison program, 133, 134  
Sound, CALL SOUND, 119-122  
Source, The, 6, 421-425  
Speech synthesizer, 6, 148-153  
Speed program for mathematics or grammar, 236-244  
Sprites, 124  
Sprites, LOGO, 409-415  
Standard calculation formulas, 134-137  
Statements and commands, TI BASIC & EXTENDED BASIC, 86-102  
Statements and commands, TI EXTENDED BASIC only, 102-107  
Statements, number per line, 125, 126  
Stopwatch program, 32, 33, 34, 35  
Strings, 24  
String variables, 29  
System, computer, 2

---

Talking dice game (program), 307-312  
Tax program (Schedule C), 194-206  
Telecommunication, 420-425  
Temperature conversion program, 30,  
31, 32  
The Source, 6, 421-425  
TI EXTENDED BASIC, 6  
Time conversion to decimals, 134, 135  
Travel comparison program, 167-177  
Turtle graphics, LOGO, 397-409

Variables, numeric, 28  
Variables, string, 29  
Video graphs program (commercial),  
57, 58

Word game program, 221-227  
Wordprocessing, general, 60  
Wordprocessing, TI WRITER, 66  
Wordprocessing, TI WRITER func-  
tions and commands, 75-82

---

The only **TI 99/4A** book you'll ever have to buy!

# THE LAST WHOLE TI 99/4A BOOK

## PROGRAMS AND POSSIBILITIES

*The Last Whole TI 99/4A Book* is a complete guide to computing on Texas Instruments' popular home computer. Written in simple, non-technical language, it covers every aspect of the TI 99/4A—hardware, software, peripherals, programming, and applications. From unpacking and setting up your system to writing, running, and debugging your own programs, *The Last Whole TI 99/4A Book* is a unique all-in-one source of information and advice—the only book you need.

### Everything you need to know to get the most from your TI 99/4A:

#### □ EQUIPMENT—

how to care for disks and disk drives, word processing without a word processor, plus ... separate chapters that show how to use the TI word processor and line printers, speech synthesizers and modems

#### □ PROGRAMMING—

you'll learn how to write programs in

TI BASIC, Extended BASIC, and the popular teaching language, TI LOGO

#### □ APPLICATIONS—

programs designed to meet your business needs, including loans, mortgages, investments, and business profit/loss determination ... programs that cover household business like budgeting, travel planning, and tax preparation ... and educational programs in arithmetic, history, grammar, spelling and other topics, which the kids can run themselves

#### □ SOFTWARE—

what's available and where to get it

#### □ PLUS...

a comprehensive dictionary of TI BASIC and an up-to-date glossary of computer terms and abbreviations, as well as a list of computer magazines

**PAUL GARRISON** has written more than 30 books, seven of them on using computers.

Wiley press guides have taught more than three million people to use, program, and enjoy microcomputers. Look for them all at your favorite bookshop or computer store.

**WILEY PRESS** a division of JOHN WILEY & SONS, Inc.  
605 Third Avenue, New York, N.Y. 10158

New York • Chichester • Brisbane • Toronto • Singapore

ISBN 0 471 87920-7