22026

POLISHING YOUR APPLE®

HERBERT M. HONIG

POLISHING YOUR APPLE®

The fun and easy-to-follow practical manual packed with useful information for the novice programmer.

- Begins with explaining the most elementary form of computer programs and progresses to more detailed programs by building upon this basic knowledge in a clear, logical way.
- Contains a concise assembly of all the procedures needed for writing, disk filing, and printing programs for an Apple® computer.
- Explains how to create disk files and how to operate various brands of printers.
- Provides a quick guide to getting into useful programming of the Apple® computer for those who want to write their own programs for business or personal applications with a minimum amount of learning time.
- Utilizes the interactive capability of the computer in the example programs.
- Provides a program to list and alphabetize up to 250 names, addresses, and phone numbers.

A valuable reference for every Apple® user, from new microcomputing students to experienced programmers.

HOWARD W. SAMS & CO., INC.
4300 West 62nd Street, Indianapolis, Indiana 46268 USA

$4.95/22026 ISBN: 0-672-22026-1

Polishing Your Apple®

Herb Honig started swimming against the tide when he left college in 1946 to manufacture custom-built high-fidelity systems. During that time, no mass market for what can now be called "one-eared stereo" existed. Several articles published in *Radio News* and *Audio Engineering* magazines (they have other names now) drew Herb into technical writing. After over 15 years in that field he set up his own quick-printing shop that he operated for 12 years. He is now working exclusively in technical writing, managing the publications department for a leading manufacturer of aircraft instruments.

Continuing his early penchant for exploring technical frontiers, Herb bought an Apple II computer and (despite being well past 50) started carving out a whole new career. He earned enough to pay for the computer by supplying price lists to quick printers, and he used the Apple word processor to expand his writing horizons. He also wrote *The Money Tool* program for controlling personal finances that is being nationally distributed by Advanced Operating Systems, a division of Howard W. Sams & Co., Inc.

Polishing Your Apple®

by

Herbert M. Honig

Howard W. Sams & Co., Inc.
4300 WEST 62ND ST. INDIANAPOLIS, INDIANA 46268 USA

FIRST EDITION
FIRST PRINTING—1982

International Standard Book Number: 0-672-22026-1
Library of Congress Catalog Card Number: 82-61967

Edited by: *Arlet Pryor*

Printed in the United States of America.

Contents

Preface

This book is intended as a primer for users of the Apple®* computer who want to write their own programs for business or personal applications with a minimum amount of learning time. It is packed with useful techniques that may be new to even experienced Apple computer users.

My experience learning to program the Apple computer showed me how frustrating it can be to get hung up on some small detail just because the needed answer was not readily available.

Programming is not really difficult for those who are disciplined in any other area of problem solving, provided one can learn how to communicate with the computer in its own language. The Apple IIc ~~Plus~~ computer uses the Applesoft BASIC language, which is a dialect not too different from other BASIC languages. Small differences, however, make it unnecessarily difficult to use texts written for general use.

I have tried to address these differences by putting in this one book the essential knowledge for programming in Applesoft BASIC, using one disk drive and either the Centronix 737 Printer, the IDS 440 Paper Tiger Printer, the Epson MX-80, or other printers in the same families using the same commands. I hope you will agree that this arrangement allows you to cut your initial learning time in half. Once you have read, copied, and used all the programs in this book, you will be much better prepared to expand your knowledge using the other available Apple computer books.

This book begins with a very simple three-line program and progresses to a final complex program with disk data storage, sorting, and printing of either a single data entry or a whole list. The programs are all explained step by step so that your knowledge

*Apple is a registered trademark of Apple Computer, Inc.

builds in a gradual, logical way. The programs have practical utility, and each can be modified to meet your particular needs.

After you finish studying this book, you will be much better prepared to understand the official Apple computer books, and other BASIC language books.

HERB HONIG

Note

Throughout this book, reversed letters (e.g. RETURN) designate keystrokes. That is, when this designation is used, it means the reversed letters are a single keystroke on the keyboard (or keys that are pressed simultaneously to produce a desired result) instead of being input as individual characters.

The Basics

Chapter 1

To start, I am assuming that you know how to set up and turn on your computer and operate the keyboard, either because the salesperson showed you how when you bought it, or you read the first chapter of *The Applesoft Tutorial,* published by Apple Computer, Inc. Now, let us get to some simple programming techniques that you can consider either as a review or as a good starting point.

BASIC programs are constructed from sequential statements (or steps) identified and ordered by numbered lines. Good programming practice uses line numbers in increments of 5, 10, or more to allow room for insertion of additional steps as the program develops or needs revision. Proper usage of line numbers is illustrated in the following section with a short, practical program.

1-1 COUNTING USING VARIABLES

The following program called COUNT 1 has only three steps. Type NEW and hit the RETURN key to clear any previous program from the Random Access Memory (RAM) of the Apple. *Always clear the RAM before starting any new program.* Type line 10 of COUNT 1 and then hit RETURN to enter it in the RAM. Do the same for lines 20 and 30. Type RUN and hit RETURN. This starts the program counting.

COUNT 1:

```
10  I = I + 1
20  PRINT I
30  GOTO 10
```

All references to $\boxed{\text{Reset}}$ should be $\boxed{\text{Ctrl}}$-$\boxed{\text{Reset}}$! When SAVEing programs, Use '.' for spaces or anything but letters & numbers.

Count 1 will count from 1 up, until it reaches the highest number the Apple computer can handle, or until you stop the count by hitting the control CTRL key at the same time as you press the C key or by pressing the RESET key.* Let us take a moment to look at how this program works. In step 10 we introduce the single algebraic variable, I. We use an expression that instructs the computer to add 1 to I every time it passes through step 10. In step 20 the PRINT command is used to report the value of I, by printing it on the screen. Step 30 sends the computer back to step 10, completing a loop that will keep adding 1 to I infinitely.

Before we change the program, to break the infinite loop, let us make sure that you understand what happens inside the computer to make it count. For every variable, such as I, the computer assigns a memory location to store the value of the variable. The initial command RUN clears all memory locations to 0 so that on the first pass step 10 assigns a value of 1 to location I and then adds to the value by an additional 1 for each loop of the program. Thus, the value stored in memory location I advances in increments of 1 until stopped. If you have not done so already, press the CTRL key at the same time as you press the C key. (This time you do not need to press the RETURN.)

We can break the loop by inserting an appropriate conditional statement, as in Count 2. Put COUNT 1 back on the screen by typing LIST and hitting RETURN. Note: Steps changed or added are designated by a bullet, as shown in step 25 of COUNT 2.

COUNT 2:

```
  10  I = I + 1
  20  PRINT I
• 25  IF I = 10 THEN END
  30  GOTO 10
```

Type only step 25, in the same way as the previous steps. When you type LIST and RETURN, you will see the revised program with all the steps in proper order. This is a feature of BASIC that helps make program development so easy. New program statements may be added or moved around at will.

see note above

~~The Apple has a switch hidden under the keyboard that can be set to prevent RESET from working unless the CTRL key is pressed simultaneously. This is to make accidental resets impossible.~~

Note: Until you gain familiarity with your Apple computer you will probably make mistakes. Many of these mistakes will create *error messages* that tell you what to check. *The Applesoft Tutorial* explains the meaning of the error messages in Appendix E, page 143. If all else fails, LIST your program and check it carefully against the ones listed in the book.

Run COUNT 2. Each time you run it, numbers 1 to 10 appear vertically at the edge of the screen. As each new set of numbers appears, it causes the previous set to move up and off the top edge of the screen. In more complex programs the screen can get cluttered with data very quickly, so you will want to know how to clear the screen for new data. Add step 5, as in COUNT 3.

COUNT 3:

```
• 5   HOME
 10   I = I + 1
 20   PRINT I
 25   IF I = 10 THEN END
 30   GOTO 10
```

Run the revised program, and a new set of numbers appears at the top of the cleared screen each time. The HOME command clears the screen and returns the cursor to the top left. Vertical display of numbers is not usually desirable, so let us look at some techniques for other formats. Retype line 20 with a comma after the I. (When you type the new line 20 the old one will be replaced.) Run the program. This technique is called *comma spacing.* The comma cancels the line feed to stop the vertical spacing and places ten blank spaces between each printout. This application is often useful in setting up tables.

Change the comma to a semicolon and run the program again. The semicolon also cancels the line feed, but it does not add any space between the numbers. All the numbers form a tight horizontal group.

1-2 COUNTING USING FOR-NEXT LOOPS

COUNT 4 illustrates another way to count. The FOR-NEXT loop is used to do the same thing as the I + 1 counter but in a different way. Line 10 defines the range of values that the variable I will take. Line 20 prints I, using the double-quote and semicolon to space the numbers along one line, and line 30 keeps looping back to line 10 through all of the defined values of I. After the highest value of I has been reached, the program breaks out of the loop and line 40 ends the program.

COUNT 4:

```
 5   HOME
10   FOR I = 1 TO 10
20   PRINT I" ";
30   NEXT I
40   END
```

The two methods of counting have different uses that often overlap. In many cases either method works equally well. Sample applications are illustrated throughout this book.

1-3 USING THE APPLE FOR DIRECT CALCULATIONS

There are occasions when it is useful to make calculations directly from the keyboard to the screen, without using numbered program statements. To do this, type PRINT and then enter your arithmetic operation. Try this: PRINT 2 + 4 and then hit RETURN. Try a few other calculations. Use the asterisk (*) for multiplication and the slash (/) for division. For complex problems use the format PRINT (2+7) * (6/5). Do not use the letter X, for multiplication; it is used as a variable. When the screen gets cluttered use the HOME command.

Using the screen for direct calculations provides *immediate execution* of the commands, as opposed to *deferred execution* by use of numbered line statements. This technique can be used advantageously in many ways; for example, it is useful to try out commands that you may not be sure will work, it can be used to turn on the printer and make lists, and it can put the Apple computer into special modes of operation, which will be described later.

There are two additional ways to clear the screen, and both do the same thing as the HOME command. With the screen cluttered with data, enter CALL -936. (This command is a carry-over from

Integer BASIC, another BASIC language available on some Apple computers. The HOME command is not available in Integer BASIC, so CALL -936 offers the advantage of providing the identical function for programs written in either language.) Now LIST more data, hit the escape ESC key and then, while pressing the shift key, hit the @ key (capital P). This last method is useful in the immediate execution mode but not easily applied to deferred execution. Note that many operations the Apple computer accomplishes may use different commands to do the same task. The choice of command is yours and should be based on what is easiest to execute, which generally means what requires the fewest number of keystrokes.

There is an alternate way to execute the PRINT command, using only one keystroke. Instead of typing PRINT 2 + 4, type ? 2 + 4. Either command performs the same task, and using the question mark in a numbered line statement whenever PRINT is needed will save you many keystrokes. Try it in immediate execution and again on the next program and note that when you LIST the program, the ? is automatically translated to PRINT instead of ?.

1-4 SAVING PROGRAMS ON DISK

Before we go further, prepare to save your programs on disk. The use of the disk drive is described on page 5 of *The Applesoft Tutorial,* which also references page 5 of the *DOS Manual* (either version 3.2 or 3.3) published by Apple Computer, Inc. Initialize a blank diskette, as described on page 13 of the *DOS Manual* before continuing with the following programs.

1-5 PROGRAMMING WITH VARIABLES

The next program, called MPG 1, illustrates an elementary program with several steps and several variables contributing to a single answer. We will build on this program to eventually produce a professional result. Type NEW, hit the RETURN key, and then type the program exactly as shown:

```
 0    REM MPG 1
10    SM = 15010 : FM = 15315 : G = 12.3
20    MT = FM – SM
```

```
30  MPG = MT / G
40  PRINT MPG
50  END
```

After you have checked your result against the printed sample, RUN the program.

Line 0 introduces the remark (REM) statement, which is a useful way to add reminders in the body of a program. A REM has no effect on the program and only shows up when you LIST the program. The REM statement is used to label the program, MPG 1. Line 10 establishes three variables, SM, FM, and G, and assigns values to each variable. Applesoft BASIC (the language of your Apple computer) can use variables that are either one or two letters in length. The advantage of two letters is having many more than 26 combinations, but also using the variable as a mnemonic (memory prompter) as shown in the MPG 1 program. SM, for example, stands for starting miles, FM is finishing miles, and G is gallons. More than two letters can be used, even making word names, but the Apple recognizes only the first two letters of any variable. Thus AUTO and AUDIO form the same variable (AU). Line 20 provides a formula for computing miles traveled (MT), and line 30 computes miles per gallon. Line 40 tells the Apple computer to print the answer (on the screen) and line 50 ends the program, clearing all variables to 0. Each time this program is run, you will get the same answer from the same variables. LIST the program and retype line 10 with new variables to get new answers as often as you desire.

Another very important new concept has been introduced in line 10. Note that there are actually three program statements in line 10, and they could have been written as lines 10, 20, and 30. Instead, they have been combined into the one line by using colons (:) to make separate statements. The advantage of using the colon is that a line number has been saved (including the two bytes of memory required). This saving can result in a significant increase in operating speed when accumulated in a lengthy program.

1-6 GETTING DETAILED PRINTOUTS AND CLEARING THE SCREEN

The following program, MPG 2, gives a more detailed printout. Line 40 of MPG 2 replaces the one line numerical answer with a complete statement showing both miles traveled and gallons consumed, to make the answer easy to understand. Note the format for the PRINT statement in line 40. Everything included within the quotes prints on the screen. Numeric variables are not enclosed in quotes so that instead of printing the letters, the Apple prints the numeric values represented by the letters. Blank spaces are left between letters and quotes so that proper spacing can result. Type the new line 40 and run the program.

```
•0    REM MPG 2
 10   SM = 15010 : FM = 15315 : G = 12.3
 20   MT = FM – SM
 30   MPG = MT / G
•40   PRINT "YOU HAVE TRAVELED "MT" MILES AT "MPG"
      MILES/GALLON "
 50   END
```

You can improve the presentation of the answer even further by adding line 35, as in MPG 3. This line clears the screen and moves the answer ten spaces down from the top.

```
•0    REM MPG 3
 10   SM = 15010 : FM = 15315 : G = 12.3
 20   MT = FM – SM
 30   MPG = MT / G
•35   HOME : VTAB 10
 40   PRINT "YOU HAVE TRAVELED "MT" MILES AT "MPG"
      MILES/GALLON "
 50   END
```

1-7 CONDENSING VARIABLE DATA ENTRIES

An easier way to enter variable data is to condense the entries using the *DATA statement,* which allows you to enter a series of numbers to be used as data variables. Line 10 of MPG 4 is the DATA statement and is read by line 15, the *READ statement,* which establishes the variables and the order they are to be read. Try out this operation; it gives exactly the same result as MPG 3.

```
•0    REM MPG 4
•10   DATA 22,15075,15317
•15   READ G,SM,FM
 20   MT = FM – SM
 30  MPG = MT / G
 35  HOME : VTAB 10
 40  PRINT "YOU HAVE TRAVELED "MT" MILES AT "MPG"
     MILES/GALLON "
 50  END
```

The advantage of the DATA statement is demonstrated in MPG 5, which provides two sets of data in line 10.

```
• 0   REM MPG 5
• 5   HOME : VTAB 10
• 10  DATA 22,15075,15317,30,15317,15723,1
  15  READ G,SM,FM
  20  MT = FM − SM
  30  MPG = MT / G
• 35  PRINT
  40  PRINT "YOU HAVE TRAVELED "MT" MILES AT "MPG"
      MILES/GALLON "
• 50  GOTO 10
```

Three different variables in line 15 read the six data items of line 10 in sequence, running through the program twice because of the instruction in line 50. Line 35 is changed to a PRINT command instead of HOME to avoid clearing the first answer and to put a blank line between the two answers. The HOME statement is moved to line 5 so that the screen clears only on the first pass of the program. If the HOME statement in line 35 were left, the first set of answers would be cleared before they could be read. Try it both ways.

You will notice that MPG 5 runs as expected, except that it produces an OUT OF DATA message after giving both answers. To provide a clean end, add line 17 and add three more data items to line 10 in MPG 6. The added 999 is an unlikely answer and is used solely to satisfy the conditional statement of line 17. The added 1's and commas (, 1, 1) simply fill out the third set of data to avoid another OUT OF DATA message. Complete MPG 6 and run it.

```
• 0   REM MPG 6
  5   HOME : VTAB 10
• 10  DATA 22,15075,15317,30,15317,15723,999,1, 1
  15  READ G,SM,FM
• 17  IF G = 999 THEN END
  20  MT = FM − SM
```

```
30   MPG = MT / G
35   PRINT
40   PRINT "YOU HAVE TRAVELED "MT" MILES AT "MPG"
     MILES/GALLON "
50   GOTO 10
```

1-8 USING THE INPUT STATEMENT AND SAVING PROGRAMS

The DATA statement is fine for programs that employ the same data many times, but in a program that is to be used to compute new data every time it is run, the INPUT statement works much better to enter keyboard data. Line 10 replaces lines 10 and 17 of MPG 6, and line 35 clears the screen for each fresh answer.

```
•0    REM MPG 7
 5    HOME : VTAB 10
•10   INPUT "ENTER STARTING MILES, FINISHING MILES,
      AND GALLONS (INSERT A COMMA BETWEEN
      ENTRIES) ";SM,FM,G
 20   MT = FM - SM
 30   MPG = MT / G
•35   HOME : VTAB 10
 40   PRINT "YOU HAVE TRAVELED "MT" MILES AT "MPG"
      MILES PER GALLON "
 45   PRINT : PRINT : PRINT
 50   GOTO 10
```

The format for the INPUT statement is very precise. The INPUT statement is a form of PRINT statement and uses quotes to define what is to appear on the screen. The semicolon following the final quote must be used, or an error message will result, and the variables must be assigned in the same order as the answers are to be typed. Answers must be separated by commas, and, therefore, no INPUT statement answer should contain any commas except those used for separation. The program will run continually asking for a new set of data following each answer. To exit the program so that you can move on to something else, type CTRL C or hit RESET. To *save this program,* now that all the steps for smooth running are completed, type SAVE MPG 7 and hit RETURN. The red light on the disk drive will light, the previously initialized disk in the drive will whirr and make noises, the cursor will disappear, and when the data is saved, the cursor will reappear and the light will go out. Type CATALOG, and you will see the MPG 7 program listed as a filed program.

1-9 INTERACTIVE PROGRAMMING WITH STRINGS

As a reward for progressing this far in the book, a game program that will be amusing (hopefully) is next. This program is not all fun and games, however; there are many new concepts introduced, including *string variables,* which you need to learn how to use before going further.

Type out the program called GAME and save it on disk, to guard against accidental loss before you start. It is always a good idea to protect lengthy programs by saving them as soon as they are written, or even in stages as you go. When you save a program under a previous name, the old program is erased and replaced by the new one. You can, of course, file your progressive changes under various names in order to refer back to earlier developments. This is particularly advisable when developing new complex programs, since sometimes the "improved" versions are not as good as the original.

Play the game a few times so that you understand what it does, and then read further to understand the mysteries.

```
100   REM GAME
140   GOSUB 1000
500   INPUT "PLEASE TYPE YOUR OWN NAME "; NM$
590   GOSUB 1000
600   INPUT "PLEASE TYPE A SENTENCE DESCRIBING
      YOUR FONDEST WISH " ; FW$
690   GOSUB 1000
700   INPUT "PLEASE TYPE THE NAME OF YOUR WORST
      ENEMY "; WE$
710   GOSUB 1000
720   PRINT " TAKE A NUMBER FROM 1 TO 9 "
725   GET NU$
730   NU = VAL (NU$)
740   GOSUB 1000
750   IF NU < 1 OR NU > 9 THEN PRINT CHR$ (7) :
      GOTO 710
```

```
800    IF NU / 3 = INT (NU / 3) GOTO 850
810    IF NU = 5 GOTO 860
820    GF$ = "HAS GONE TO FLORIDA "
840    PRINT WE$ + " " + GF$ ", " + NM$" SO YOU CAN
       HAVE YOUR "FW$
845    GOTO 990
850    PRINT WE$" HAS TAKEN YOUR "FW$", TOO BAD,
       "NM$
855    GOTO 990
860    PRINT "THIS IS YOUR LUCKY DAY, "NM$," TAKE
       YOUR "FW$
990    VTAB 20 : PRINT "PRESS SPACE BAR TO
       CONTINUE, ESC TO QUIT "
992    GET S$
995    IF ASC (S$) = 27 THEN END
997    GOTO 710
1000   HOME : VTAB 10 : RETURN
```

A *string variable* is formed by using a dollar sign ($) at the tail end of the root, as in line 500 of GAME where the string is NM$. The contents of a string variable may be any group of alphanumeric characters up to 255. Each time you use the string in a PRINT statement, you get the whole string so that with a two- or three-character variable anything from a letter to a paragraph may appear. In GAME, NM$ becomes your name, and WE$ becomes your worst enemy. Strings may be entered as answers to an INPUT statement, as equations in programs, or as items in a DATA statement. To illustrate this point, GF$ is used in line 820 to make up part of the output statement in line 840. Note the format: Everything to be included in the string is within the quotes, as in line 820, and strings that are added together (concatenated) are joined with a plus (+) sign, as in line 840.

Once you understand how letters, words, and phrases can be inserted by an INPUT statement, you can see how easily words can be manipulated so that they pop up at will in PRINT statements. This technique is the foundation for creating interactive computer programs, such as those used in teaching situations.

A string may be letters, numbers, or mixtures. Sometimes it is better to use a string variable instead of a numeric variable, and then the string variable can be converted to its numeric value. This is done in line 730 where the conversion is made by an equation. The reason for using a string, in this case, is that although a numeric variable will work just as well if correct answers are always given, the technique used here responds equally to letters and numbers and rejects wrong answers by going back to the question. At the same time the Apple computer sounds a beep to alert the user that something has been done wrong. Try picking any keyboard character except the numbers 1 through 9 and see how nicely this works.

The first part of line 750 defines which answers are not acceptable, specifically anything not equal to (< >) 1 through 9. The first reaction to a wrong answer is made by execution of the command PRINT CHR$(7). The CHR$(7) is the program statement equivalent of a CONTROL G, and on the keyboard the bell signal is part of the G key. Try typing CTRL G. Anytime the CHR$ ( ) format with the appropriate ASCII number for a control signal is included, you will make that control signal execute at that point in the program. In case you are mystified by the term ASCII, let me explain. The letters stand for the *American Standard Code* for *Information Interchange,* and as the name implies, signals with this code permit computers to communicate with each other by telephone line or other means. The peripherals used with your Apple, such as the disk drive or the printer, all work with this code. Each character has a numeric value, and upper- and lower-case letters have different values. To help you find any codes you may need and to clarify your understanding, two special programs are provided in Chapter 2.

This discussion was the result of explaining how the bell signal is made to sound on an erroneous input. A second action to the same condition illustrates a special use of the colon (:) in multipart statements such as line 750. Once the conditional statement is made, all other statements in the same line become subject to the stated condition. Thus line 750 not only causes the beep to sound, but also causes a return via line 710 to the original INPUT statement, signaling that the Apple is still waiting for a satisfactory answer.

Depending on which number has been selected, the program will branch to one of three printouts given in lines 840, 850, and 860. Line 800 provides an equation that responds to any number divisible by 3. If NU is 1, 2, 4, 5, 7, or 8, the resulting division by 3 (NU/3) will produce a compound fraction consisting of an integer (whole number) and a decimal remainder. Equality with the right-hand side of the equation INT(NU/3) (or the integer of NU/3) only occurs for 3, 6, or 9. Thus, only those numbers will cause a jump to line 850. For all other numbers the next step is line 810, and this line causes a jump to line 860 only if the selected number is 5. For the remaining numbers, the program goes directly to line 840. If a jump occurred to either of the two previous printouts, then the following printouts are jumped over by the GOTO statements in lines 845 and 855. Now that you know which numbers to choose, you can always get a happy answer. Sorry, if this explanation has taken the fun out of the game, but now you have the key to creating your own simple games.

Line 990 starts a routine to end the program at any point and coincidentally uses another form of INPUT statement. The GET command of line 992, operating with the question asked in the PRINT statement of line 990, functions in the same way as the INPUT statement with one important difference. A GET responds instantly to the next keystroke before the RETURN key can be pressed. Thus, whatever character is entered becomes S$ and is the input. Pressing the ESC key causes the ASCII value (27) to equal S$ (line 992), satisfying the equation in line 995, and making the program end. Pressing any other key (including the space bar) allows the program to continue to line 997, which loops back to line 710.

One last feature has been introduced in the interest of saving steps. It is a technique used in long programs whenever the same routine occurs many times. Line 1000 contains three operations, and whenever these operations are needed a statement such as lines 140 and 590 calling for the subroutine in line 1000 is used. The RETURN command always causes a return to the step following the GOSUB instruction. Subroutines may be any length and may include other subroutines nested within them.

1-10 CONCLUSION

This chapter has introduced most of the beginning concepts needed for programming in BASIC, teaching mainly by example in a concise, fast moving form that many users prefer to the slow piece-by-piece feeding of new facts used by some traditional educators. The remaining chapters move just as quickly.

This pace will keep you from getting bored, but it may also move you along too fast to catch everything on the first pass. Unless you are an unusually quick student, you may benefit from rereading this book at least once.

Editing and Utility Routines

Chapter 2

Now that you have some exposure to programming, you can appreciate the need for quick, easy revision of material entered into the memory of the Apple computer. Unless you are an expert typist, you have probably made some errors in setting up the previous programs and perhaps used more time than you would have liked in making corrections. Before you get into lengthy program writing, it is worthwhile to stop and learn some of the tricks of editing and correcting programs.

2-1 MOVING THE CURSOR TO MAKE CORRECTIONS

To illustrate the correction techniques, make up a print statement such as:

```
100  PRINT "NOW IS THE TIME FOR ALL GOOD MEN TO
     COME TO THE AID OF THE PARTY. "
```

and enter it on the screen as a new one line program. Observe that, as originally typed, both lines in this statement start at the left-hand side of the screen. Now LIST the program and notice how the format changes. The first line remains at the left-hand side of the screen, but the second line is indented. We will soon study a special problem created by this format change and an easy solution to that problem. But first, let us look at a way to make changes in existing statements.

The cursor can be moved to anywhere desired, and then entries can be inserted or corrected. In order to make such changes permanent, a very specific routine called *pure cursor moves* must be followed. To initiate these moves, press the ESC key once. After that, the keys I, J, K, and M will move the cursor up, left,

or ↑ ← → ↓

The retype key is $\boxed{\rightarrow}$

(some keys do not do this because they do special stuff)

right, and down, respectively. To exit this mode, use any other key. The first operation of any other key causes the exit and nothing else. The following operations cause normal typing. If you forget to type one dummy character, you will find that your first expected character is missing. ~~Caution: if you have an older Apple computer without the autostart ROM, you cannot operate in this way.~~

Try moving the cursor by using these keys. Move the cursor to the middle of the test sentence and change a few words. Now, list the program again and notice that your changes did not register!

You must first move the cursor to the very beginning of the numbered statement, that is, to the 1 in 100. Use the retype key (right arrow) to bring the cursor to the beginning of the numbered statement that needs the change and then type over the words or letters that you wish to change. Complete the correction by moving the cursor all the way to the end of the line, with the ~~retype and repeat REPT~~ $\boxed{\rightarrow}$ key~~s~~ and then hit RETURN. Now list the program and see that your change registered. Also, note that the change operation caused the sentence to separate whenever the retype key moved the cursor over blank spaces in the left margin. Take heart, correct this separation by remembering to use a special command before making the change. As previously noted, the originally typed sentence filled the whole line and crowded the left margin. After the program was listed, all the lines except the first were indented. Type line 100 all over again and LIST it. To restore the listed line to the original format, type POKE 33,33 and hit RETURN. All lines listed after that will crowd the left margin, and when you make corrections using the pure cursor moves, no blank spaces will be inserted. Try listing line 100 again.

The POKE caused location 33 to change from its normal 40 character per line control mode to 33 characters. Any characters in the seven right most spaces of the cleared screen will remain. This area does not clear during this POKE mode, and other strange and sometimes undesirable effects will also occur. Therefore, after all corrections are made, restore the normal mode by typing POKE 33,40. or TEXT.

An easier method with the same effect is to type TEXT. You can also clear everything on the screen and then POKE, using HOME : POKE 33,33 and RETURN. To LIST line 100 at the same time, type HOME : POKE 33,33 : LIST 100 and RETURN.

'pure cursor' is usually called ESCape mode.

Practice the pure cursor moves, listing the line after each change to make sure you understand how to make the corrections register correctly.

2-2 ADDING WORDS IN THE MIDDLE OF A STATEMENT

The pure-cursor-moves method of correction can work only if you are replacing words with new words of the same or shorter length. You can extend this procedure to add any length correction you wish within the 255 character limit of a single numbered statement. Pure cursor moves do not change the letters that the cursor moves over. The only way to change these letters is to type a different letter or a blank with the space bar. You can, therefore, move the cursor to the place you want to make the insertion using the left and right arrow keys; then move the cursor up or down with pure cursor moves so that the cursor is on a blank space above or below the statement being corrected. Type the information to be added (including all spaces), and then bring the cursor back to the original point of insertion, using only pure cursor moves. Finish going over the line with the retype key; then hit RETURN and list the line. If you moved the cursor properly, the insertion should be in place. Keep practicing making insertions until you have it down pat. Being able to use pure cursor moves to make corrections and insertions will save you much time later.

2-3 CHANGING/DELETING LINE NUMBERS

You can use pure cursor moves to change the line number of a statement. When using this method, not only the statement with a new number will appear, but the same statement with the original number will also appear. *Always delete the old statement after the change by retyping only the line number* unless, of course, you need both statements or one statement with a slight change.

2-4 USING THE DISK CATALOG

Type CATALOG and note that the disk drive goes into operation, causing a catalog display of all the programs stored on the disk. The information has many uses. For the programs generated so far, an A followed by a number will appear along with the name you assigned to each program. The A means that the record stored on the disk is an Applesoft program; the number tells you how much space the program takes up on the disk, and, of course, the name tells you which program it is. The disk can hold up to ~~560 sectors~~ 280 blocks for ~~disk operating system (DOS) 3.3,~~ ProDOS each ~~256~~ 512 bytes long or a total of 143,360 bytes. A *byte* is a single 8-bit character that occupies a single memory location. Your ~~48K~~ 128K Apple Computer Ram, therefore, holds ~~48~~ 128 × 1024, or ~~49,152~~ 131,072 characters. (1024 = 2 to the power of 10, and is 1K of binary storage.) A single disk can, therefore, hold nearly ~~three~~ 1.5 times as much information as the RAM. The number alongside each program is the number of sectors that program occupies. When you try to save more than the ~~492 sectors~~ 273 blocks the disk can handle, you will get a DISK FULL message. ~~(68K of the 560K is used for DOS overhead).~~ Delete the last program you tried to store, insert another disk (previously initialized), and again save your program. You can also get a DISK FULL message if you try to put more than ~~64~~ 52 catalog titles on the disk, even though free sector space is available.

2-5 LOCKING FILES

You can protect finished programs from accidental erasure by locking the program. Type LOCK followed by the file name and watch the disk go into action. When it stops, call for another catalog listing and note that the locked file is preceded by an asterisk. You will not be able to write anything into this file unless you unlock it, using the command UNLOCK and the file name.

2-6 WRITE PROTECTING A DISK

You may have noticed that you cannot write any information onto the system master disk, that is because the disk is *write protected.* Any disk can be write protected by closing off the rectangular notch in the side of the diskette with tape. A write protected disk can be written on if the notch is restored, or a new notch cut into the side of the diskette. You can also write on both sides of a diskette, by cutting an extra notch opposite the manufactured notch and inserting the diskette bottom side up. Such disks are not quite as reliable as disks using one side only, (there is never a free lunch). but for fun and games type storage and especially for infrequently used back up files, the method offers extra economy.

2-7 RECALLING FILES

You may have also noticed that unless you type the file name exactly as stored when loading or running, you will get a FILE NOT FOUND message. You can also file a program under a mis-typed name, leaving the previous title and file untouched, which can lead to later confusion. One way to avoid all these problems is not to retype the title at all, but simply to use the pure cursor moves to position the cursor at the beginning of the title, inserting the desired command, such as SAVE or LOCK, by typing over the left-hand catalog data, and then moving the cursor over the existing title using the ~~RETYPE~~ → key. This eliminates all chance of error and is usually easier than typing.

Several good utility routines are on the market that automatically display a catalog menu when the disk is booted, allowing you to make a selection with a single keystroke.

2-8 DELETING/CHANGING FILE NAMES

You may delete a file by typing DELETE and the file name. Be sure to unlock any locked files first. To change the name of a file, type RENAME *old file name, new file name.* Note that you need to type the comma to separate the two file names.

2-9 FINDING HIDDEN CONTROL CHARACTERS

In ProDOS, you get an error if a CTRL char. is in the name.

If you accidentally hit the CTRL key while typing a file name, you may embed a hidden control character in the name. This happens most often with CONTROL A. ~~The only way you can retrieve the file is to type the correct hidden control character, in the original sequence, when you LOAD or RUN that file. You can find the hidden control characters anywhere in your file name (or in a program if such characters are troublesome) by running~~

~~the special program listed under the heading "File Names" on page 151 of the *DOS Manual*.~~ The most common control character error occurs as a result of hitting the CTRL key while typing the letter A .

2-10 WORKING WITH ASCII CODES

Working with such codes as CHR$(7) shows the need to understand how to translate the code numbers to their corresponding characters and vice versa. Using the following two utility programs will help you to understand this translation. Type and RUN the program called LETTER TO NUMBER. Enter any character on the keyboard, including control characters, and note the corresponding numbers. Make a record of these numbers. Note line 140, which includes two new commands, FLASH and NORMAL. The action of these commands is obvious when you run the program.

```
  0    REM LETTER TO NUMBER
100    HOME : VTAB 10
110    PRINT "ENTER THE CHARACTER FOR WHICH YOU
       WANT THE ASCII VALUE : ";C$
120    GET C$
125    HOME : VTAB 15
130    PRINT "THE ASCII VALUE FOR "C$" IS " ASC(C$)
140    VTAB 18 : PRINT "MORE? "; : FLASH : PRINT "Y/N " :
       NORMAL
150    GET M$
160    IF M$ < > "N" THEN RUN
170    END
```

Save the previous program and then type and RUN the NUMBER TO LETTER program. Convert the numbers you recorded back to the original characters. Note that you can get the letter A by using either 65 or 97. The difference is that 65 represents an upper-case A and that 97 represents a lower-case A. Since the screen will show either command as an upper-case A, you can see the difference only by programming PRINT CHR$(65) and PRINT CHR$(97) into your printer.

```
  0    REM NUMBER TO LETTER
100    HOME : VTAB 5
```

```
110  INPUT "ENTER THE ASCII CODE " ; A
130  HOME : VTAB 10
140  PRINT "THE LETTER CORRESPONDING TO ASCII
     CODE" : VTAB 12 : HTAB 18 : PRINT A : VTAB 14 :
     HTAB 16 : PRINT "IS " CHR$(A)
150  VTAB 18 : PRINT "MORE? (Y/N) "
160  GET M$
170  IF M$ < > "N" GOTO 100
180  END
```

Now that you understand the function of these programs, you can keep both of them on disk and use them anytime you need to analyze codes and numbers.

2-11 USING PRINTER COMMANDS

To turn the printer on, use the command PR#1 (assuming your printer control card is in slot #1), either in a numbered statement, or as a direct (immediate execution) command. From that point on, all PRINT statements will print out both on the screen and on the printer. The command PR#0 turns the printer off. That is all you have to know if you are content to limit your print commands to the basic character size that your printer offers, and if you will accept a 40 character line on your printed sheet. The commands for special type sizes and styles vary with different printers and will be dealt with in other parts of this chapter.

The Apple ~~parallel printer~~ Superserial card requires a special command to change the standard , 40 80 character line width of the ~~screen~~ Printer to another width. Any desired width, up to the limit of the capacity of the printer may be obtained by using PRINT CHR$(9) ''XXN''; where XX is the number of characters you want per line. ~~When using this command *do not use N as a program variable.* If you forget this warning you may find your program variable interacting with the line width command, causing erratic line widths to print out.~~

You may use this line width control as either a direct command or in a numbered program statement. The command is automatically cancelled by a PR #0. An example of the use of this command is given in the program under the main heading 3-7. You may also

PRINT CHR$(9); "XXN"

or TAB

use CONTROL I [the equivalent of CHR$(9)] if you prefer. My own choice is for the visible form, CHR$(9).

~~If you use longer than a 40-character line, without shutting the screen off, you could be inviting trouble. If the screen logic circuits try to operate with anything longer than the standard width, confusion may result, which can cause strange things to happen. The screen circuits could introduce machine language that would appear in the middle of beginning program statements and clobber the program. All this can be avoided by using a command that turns off the screen. That command is POKE 1913,1, assuming your printer is in slot #1, and assuming that your printer and control card require a line feed command with every RETURN. For other slots, use the generalized form: POKE 1912 + s,1 (where s is the slot number). If your printer does not need the line feed command use POKE 1912 + s,0. Like the line width command, this poke is cancelled by a PR#0.~~

Machine language, incidentally, is assembly language programmed permanently into all computers including the Apple computer. Although machine language is more flexible and efficient than Applesoft BASIC or any other high-level language, it is more difficult to learn and use. You will not be concerned with it unless you advance to a very sophisticated level of programming.

PRINT CHR$(4); "PR#s"

Apple provides alternate commands for turning the printer on and off, and these commands may prove useful under those peculiar program conditions where the PR#1 and PR#0 commands do not work as they should. (In a program) ~~These~~ This command~~s are POKE 54,0 : POKE 55,193 for on, and POKE 54,240 : POKE 55,253 for off~~. An example of the use of this and other special print commands is provided in the program at the end of the next section of this chapter, under main heading 2-12.

2-12 SPECIAL COMMANDS FOR THE CENTRONIX 737 PRINTER

The Centronix 737 (and successors) is a very flexible model offering three basic type styles that may be printed in either normal or expanded mode. This printer can also underline as it prints. Specific, tested examples of how to execute the commands for this printer are included in the following paragraphs.

To select the smallest print size available, which is 16.7 characters per inch, the printer manual simply tells you to use the decimal code 27,20. However, to make the printer operate in this size follow this sample:

```
  100   PRINT CHR$(27);CHR$(20);
```

The semicolons assure that there are no *extra line feeds* when you turn on the printer with these commands. The printer will stay at this size until you send another code to the printer or shut the printer off. When you turn the printer back on, it will resume its normal (default) size, which is 10 characters per inch. Or, instead of turning the printer off to get back to the normal size, use the command:

```
  100   PRINT CHR$(27); CHR$(19);
```

To select proportional print which uses letters of differing widths to simulate printers type such as the type in this book, use PRINT CHR$(27);CHR$(17);.

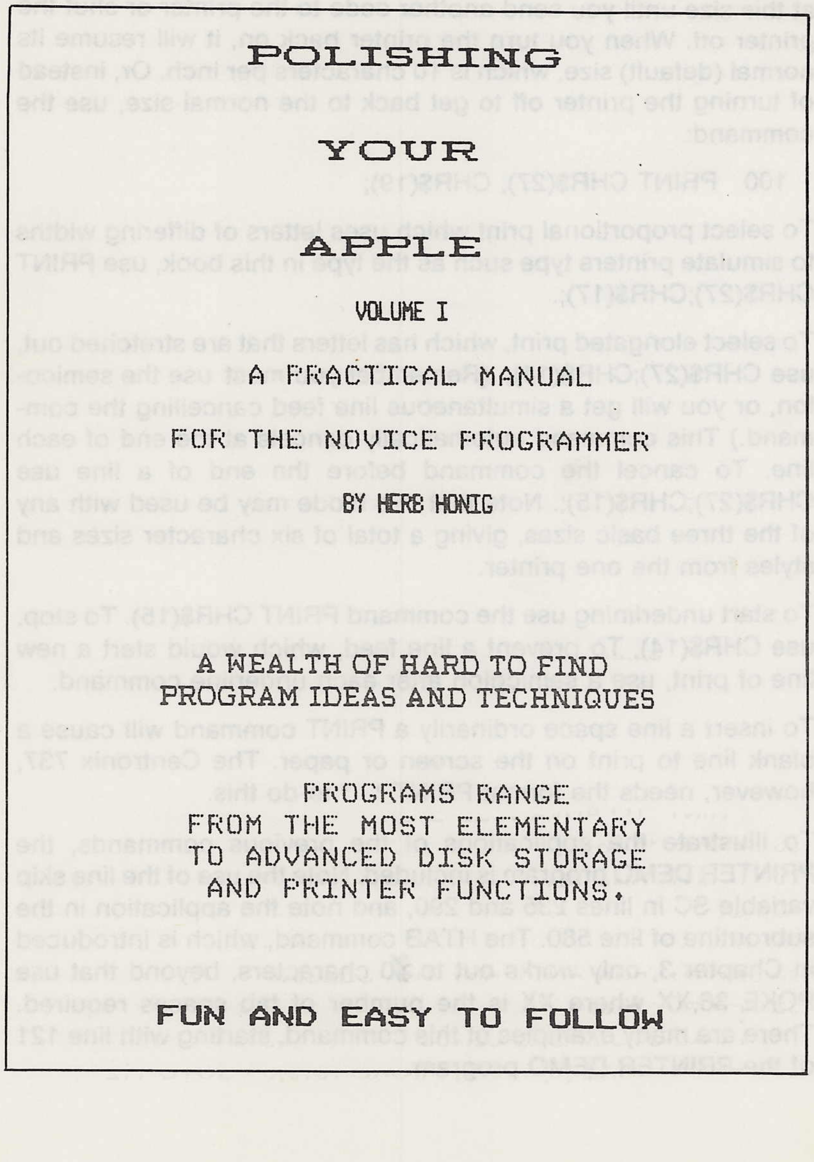
To select elongated print, which has letters that are stretched out, use CHR$(27);CHR$(14);. (Remember you must use the semicolon, or you will get a simultaneous line feed cancelling the command.) This command automatically cancels at the end of each line. To cancel the command before the end of a line use CHR$(27);CHR$(15);. Note that this mode may be used with any of the three basic sizes, giving a total of six character sizes and styles from the one printer.

To start underlining use the command PRINT CHR$(15). To stop, use CHR$(14). To prevent a line feed, which would start a new line of print, use a semicolon after each underline command.

To insert a line space ordinarily a PRINT command will cause a blank line to print on the screen or paper. The Centronix 737, however, needs the format PRINT " " to do this.

To illustrate the applications of the previous commands, the PRINTER DEMO program is included. Note the use of the line skip variable SC in lines 235 and 290, and note the application in the subroutine of line 580. The HTAB command, which is introduced in Chapter 3, only works out to ~~4~~80 characters, beyond that use POKE 36,XX where XX is the number of tab spaces required. There are many examples of this command, starting with line 121 of the PRINTER DEMO program.

The entries in the PRINTER DEMO program will produce the cover type for the original version of this book which was privately produced by the author. The illustration below shows the results of this program.



POLISHING

YOUR

APPLE

VOLUME I

A PRACTICAL MANUAL

FOR THE NOVICE PROGRAMMER

BY HERB HONIG

A WEALTH OF HARD TO FIND
PROGRAM IDEAS AND TECHNIQUES

PROGRAMS RANGE
FROM THE MOST ELEMENTARY
TO ADVANCED DISK STORAGE
AND PRINTER FUNCTIONS.

FUN AND EASY TO FOLLOW

I don't know if it works!

10 ESC$=CHR$(27)

```
100   REM PRINTER DEMO (TITLER)
110   GOTO 500
112   GOSUB 530 : POKE 36,53
115   GOSUB 510
120   PRINT "POLISHING"
121   PRINT " " : PRINT " " : POKE 36,61 : GOSUB 510
123   PRINT "YOUR"
124   PRINT " " : PRINT " " : POKE 36,59 : GOSUB 510
125   PRINT "APPLE"
130   PRINT " "
140   GOSUB 550
145   POKE 36,51
150   PRINT "VOLUME I"
160   PRINT " "
170   GOSUB 540
175   POKE 36,25
180   PRINT "A PRACTICAL MANUAL" : PRINT " "
185   POKE 36,21
190   PRINT "FOR THE NOVICE PROGRAMMER"
200   GOSUB 540 : GOSUB 550
210   PRINT " "
225   POKE 36,50
230   PRINT "BY HERB HONIG"
232   SC = 4 : GOSUB 580 : GOSUB 530 : POKE 36,48 :
      PRINT "A WEALTH OF HARD TO FIND" : POKE 36,44 :
      PRINT "PROGRAM IDEAS AND TECHNIQUES"
235   SC = 2 : GOSUB 580
240   GOSUB 540
245   POKE 36,28
250   PRINT "PROGRAMS RANGE"
255   POKE 36,22
260   PRINT "FROM THE MOST ELEMENTARY"
265   POKE 36,22
270   PRINT "TO ADVANCED DISK STORAGE"
280   POKE 36,23
285   PRINT "AND PRINTER FUNCTIONS."
290   SC = 3 : GOSUB 580
295   GOSUB 550 : GOSUB 510 : POKE 36,17
300   PRINT "FUN AND EASY TO FOLLOW"
400   ~~PR#0 : END~~ PRINT CHR$(4)"PR#0"
500   ~~POKE 54,0 : POKE 55,193~~ Print CHR$(4) "PR#1"
505   PRINT CHR$ (9)"120N"; : ~~POKE 1913,0 :~~ GOTO 112
```

See Imagewriter book for codes

```
                         "n";
510  PRINT CHR$ (27); ~~CHR$ (14)~~; : RETURN : REM
     ELONGATED PRINT  "E";
520  PRINT CHR$ (27); ~~CHR$ (15);~~ : RETURN : REM STOP
     ELONGATED PRINT "p";
530  PRINT CHR$ (27); ~~CHR$ (17);~~ : RETURN : REM
     PROPORTIONAL     "E";
540  PRINT CHR$ (27); ~~CHR$ (19);~~ : RETURN : REM
     PRIMARY CHARACTER SIZE
550  PRINT CHR$ (27); ~~CHR$ (20)~~; : RETURN : REM SMALL
     PRINT            "Q";
580  FOR I = 1 TO SC : PRINT " " : NEXT I : RETURN
```

2-13 SPECIAL COMMANDS FOR THE PAPER TIGER

Another popular printer that also offers variable type size is the IDS Paper Tiger, Model 440. The commands for this printer are different from those just described. Unlike the Centronix 737, the Paper Tiger has a switch selector system that permits it to operate on any one of its print styles when it is turned on. The style can be changed by programming the appropriate commands.

To select small print which is 16.5 characters per inch, use

```
100  PRINT CHR$(31);
```

To select 12 characters per inch, use

```
100  PRINT CHR$(30);
```

To select 10 characters per inch, use

```
100  PRINT CHR$(29);
```

To select 8.3 characters per inch, use

```
100  PRINT CHR$(28);
```

To select enhanced print, which is similar to the elongated print on the Centronix 737, use

```
100  PRINT CHR$(1);
```

To select normal mode use

```
100  PRINT CHR$(2);
```

The normal mode command cancels the enhanced mode.

Ignore this program if you don't have an Epson ↓

2-14 SPECIAL COMMANDS FOR THE EPSON MX-80

The Epson MX-80 offers two basic type sizes with three different modes of character emphasis: normal, enhanced, and double strike. It also provides great flexibility in control of the horizontal line feed spacing and the vertical and horizontal tabbing. The following test program will allow Epson users to exercise the printer through its most important modes, using examples of typical commands. Because of the large number of printing possibilities, experiment with combinations of these commands to gain experience in programming the full range of possibilities. This program is intended to show correct format for Apple use.

```
100  REM EPSON MX-80 TEST
105  PR#1
110  PRINT CHR$(15); : PRINT "CONDENSED CHARACTER
     TEST"
120  PRINT CHR$(15);CHR$(27);"G" : PRINT "CONDENSED
     SIZE/DOUBLE STRIKE"
130  PRINT CHR$(18); CHR$ (27)"H"; : PRINT "STANDARD
     SIZE/SINGLE STRIKE"
140  PRINT CHR$(27)"G"; : PRINT "CHANGE TO DOUBLE
     STRIKE"
150  PRINT CHR$(27) "H"; CHR$(27)"E"; : PRINT "CHANGE TO
     SINGLE STRIKE/EMPHASIZED"
160  PRINT CHR$(27) "G"; : PRINT "CHANGE TO DOUBLE
     STRIKE/EMPHASIZED"
170  PRINT CHR$(27)"H"; CHR$(27)"F"; : PRINT "CANCEL
     EMPHASIZED"
180  PRINT CHR$(15);CHR$ (14); : PRINT "DOUBLE
     CONDENSED"
190  PRINT CHR$(14);CHR$(27)"G"; : PRINT "DOUBLE
     CONDENSED/DOUBLE STRIKE"
200  PRINT CHR$(18);CHR$(27)"H"; : PRINT "CANCEL SHIFT
     IN (CONDENSED SIZE)/CANCEL DOUBLE PRINT"
210  PRINT CHR$(14); : PRINT "DOUBLE SIZE"
220  PRINT CHR$(14);CHR$(27)"G"; : PRINT "DOUBLE SIZE/
     DOUBLE STRIKE"
230  PRINT CHR$(14); CHR$ (27)"E"; : PRINT "DOUBLE SIZE/
     DOUBLE STRIKE/EMPHASIZED"
240  PRINT CHR$(27)"O"; : PRINT "1/8-INCH LINE SPACING"
250  PRINT CHR$(27)"1"; : PRINT "7/72-INCH LINE SPACING"
```

Ignore this program if you
don't have an Epson)

```
260  PRINT CHR$(27)"2"; : PRINT "1/6-INCH LINE SPACING"
270  PRINT CHR$(27)"B";   CHR$(N + 128); : REM N = VTAB
     SETTING, SEE LINE 280
280  PRINT CHR$(11); : PRINT "EXECUTES VTAB SETTING"
290  PRINT CHR$(D);CHR$(N1); CHR$ (N2); CHR$ (N3); CHR$
     (0); : REM HTAB SETTING, SEE LINE 300
300  PRINT CHR$ (9); : PRINT "EXECUTES HTAB SETTING"
310  PR#0 : END
```

Some Simple, Practical Programs

Chapter 3

The first step in any programming is to define what the program is to do. Now that many gas stations are measuring their delivery in liters, it would be useful to develop a program that shows you how many miles per gallon your car is getting, using how many liters you bought per tankful. Because this is a learning experience, we will develop this program in small, easy steps.

3-1 PRINTING TABLES

To illustrate the use of tables and add a new dimension to the MPG programs, try the M/L 1 program. The new dimension is conversion from liters to gallons so that you can compute gasoline consumption even when your station measures its sales in liters.

```
0    REM M/L 1
10   HOME : VTAB 10
20   INPUT "ENTER STARTING MILES, FINISHING MILES,
     AND LITERS CONSUMED (INSERT A COMMA
     BETWEEN EACH ENTRY)";SM,FM,L
30   MT = FM - SM : REM MT = MILES TRAVELED
40   G = L * .264200793 : MG = MT / G
45   REM MG = MILES PER GALLON
50   HOME : VTAB 10
60   PRINT "    MILES "
61   PRINT "TRAVELED LITERS          GALS        MPG "
62   PRINT
63   HTAB 3 : PRINT MT; : HTAB 12: PRINT L; : HTAB 18 :
     PRINT G; : HTAB 30 : PRINT MG
65   PRINT : PRINT : PRINT
```

```
70    PRINT " MORE? (Y/N) "
80    GET M$
90    IF M$ = "Y" GOTO 10
95    IF M$ <> "N" GOTO 70
100   END
```

RUN the M/L 1 program before reading the explanation that follows.

Line 60 prints the word MILES as the first line in the table headings. Note that MILES is indented two spaces, by allowing the appropriate number of spaces between the first quote mark and the letter M. Line 61 provides the second line of the heading, with the appropriate spacing. Line 63 introduces a new technique to space out the printout. The HTAB command moves the first letter of the printout the designated number of spaces from the left margin. By using a semicolon after each printout, everything is made to print on one line, spaced under control of the HTABs.

3-2 ROUNDING

The M/L 1 program works well, except that the answers are given to far more decimal places than needed. A more useful result can be obtained by rounding the answer to two places for gallons and liters, and three places for miles per gallon, using the rounding formulas added in steps 52, 53, and 54 of the M/L 2 program. These formulas round with fairly good accuracy, but they show up a characteristic of the Apple computer; which is, when the right most decimal digit is a 0, it is dropped.

For the curious, here is a brief explanation of why the formula works. Assume the unrounded value of G in line 52 is 10.47832. Multiplying by 100 yields 1047.832, and the integer function would truncate this to 1047 if we had not added the 0.~~8~~5. With the addition ~~(1047.832 + 0.8 = 1048.632)~~ we truncate to 1048, and dividing this by 100 yields 10.48, a properly rounded answer.

```
•0    REM M/L 2
10    HOME : VTAB 10
20    INPUT "ENTER STARTING MILES, FINISHING MILES,
      AND LITERS CONSUMED (INSERT A COMMA
      BETWEEN EACH ENTRY) " ; SM, FM, L
```

```
  30   MT = FM − SM : REM MT = MILES TRAVELED
  40   G = L * .264200793 : MG = MT / G
  45   REM MG = MILES PER GALLON
  50   HOME : VTAB 10
• 52   RG = INT (G * 100 + ~~.8~~ .5) / 100
• 53   LR = INT (L * 100 + ~~.8~~ .5) / 100
• 54   MR = INT (MG * 1000 + .5) / 1000
  60   PRINT "      MILES "
  61   PRINT "TRAVELED LITERS          GALS       MPG "
  62   PRINT
• 63   HTAB 3 : PRINT MT; : HTAB 12 : PRINT LR; : HTAB
       22 : PRINT RG; : HTAB 32 : PRINT MR
  65   PRINT : PRINT : PRINT
  70   PRINT " MORE? (Y/N) "
  80   GET M$
  90   IF M$ = "Y" GOTO 10
  95   IF M$ <> "N" GOTO 70
 100   END
```

The introduction of the rounding formulas and headings during the development of this program left some crowding between lines 50 and 70, so before proceeding let us renumber the program to allow room for new steps. The Apple does this with astounding ease and now is a good time to demonstrate the process. Before you proceed, SAVE M/L 2.

Do <u>not</u> do this with ProDOS!

3-3 RENUMBERING AND LISTING

Insert the system master disk and type RUN RENUMBER. After the disk stops and the cursor starts to blink, hit RETURN. A message will appear at the top of the screen, indicating that the renumber program is ready to run. Insert your regular program disk back in the disk drive and type LOAD M/L 2 (or whatever you named the program). When loading is complete, type: & F50, I 20 indicating that the first step should be renumbered as 50, and all other steps should be numbered in increments of 20. Almost immediately the process is complete and can be verified by listing the program. You will have one problem reading this list, however, and that is, because the program now occupies more than one screen of space, the top lines of the program scroll off the top of the screen to make room for the lines that follow.

You may stop the lines from scrolling completely through the program by typing a CTRL S as soon as the screen starts to fill. Typing any character will start scrolling again, and you can stop as often as you wish with the CTRL S. Practice this technique,as you will need it often. You may also list small portions of the program at a time. To do this type LIST 50-290. When you want to view the remainder of the program type LIST 310-.

```
•50   REM M/L 3
 70   HOME : VTAB 10
 90   INPUT "ENTER STARTING MILES, FINISHING MILES,
      AND LITERS CONSUMED (INSERT A COMMA
      BETWEEN EACH ENTRY) ";SM,FM,L
 110  MT = FM - SM : REM MT = MILES TRAVELED
 130  G = L * .264200793 : MG = MT / G
 150  REM MG = MILES PER GALLON
 170  HOME : VTAB 10
 190  RG = INT (G * 430 + .8) / 100
 210  LR = INT (L * 430 + .8) /100
 230  MR = INT (MG * 1000 + .5) / 1000
 250  PRINT "    MILES "
 270  PRINT "TRAVELED LITERS       GALS       MPG "
 290  PRINT
 310  HTAB 3 : PRINT MT; : HTAB 12 : PRINT LR; : HTAB
      22 : PRINT RG; : HTAB 32 : PRINT MR
 330  PRINT : PRINT : PRINT
 350  PRINT " MORE? (Y/N) "
 370  GET M$
 390  IF M$ = "Y" GOTO 70
 410  IF M$ <> "N" GOTO 350
 430  END
```

Another way to view a long program is to slow down the operating speed of the Apple. To do this, type SPEED = 100 before you list the program. The program will then scroll very slowly, and you can follow its action. Try other speeds to become familiar with the available range. The normal maximum speed is 255. Now that you have seen the disadvantages of reading program lists on-screen, use the PRINT commands learned under the main heading 2-11 and PRINT out the program. With your printer set up and ready, type PR#1, RETURN, LIST, RETURN. You should get a complete printed list, just like the one in this book.

The renumbering process is neat, clean, and easy to use. One problem will sometimes show up, as in the example just renumbered, and you must watch for it whenever you renumber. Look at the new steps 190 and 210 and compare them with the old steps 52 and 53 as printed here in the M/L 2 program. Notice how in the rounding formula the 100 inside the parentheses is treated as a renumbered step (old step 100 is now 430). If you run this program without correcting these two steps, you will get answers approximately 4.3 times too high. Make the correction now, so you do not forget, and always check for this condition when using the renumbering program.

One way to avoid this problem completely is to define the rounding factors early in the program as follows for example in step 55:

```
55   R1 = 100 : R2 = 1000
```

If you then use R1 and R2 in place of 100 and 1000 in the rounding formulas, you will avoid trouble.

3-4 STORING DATA IN MEMORY WITH SUBSCRIPTED VARIABLES

To average out the results of several readings in order to get a more accurate measure of the performance of your car and also to pick up some new programming techniques, make the additions in program M/L 4. Add lines 75, 240, 300, 313, 314, 315, 316, and 326. Add the subscript (I) as shown in steps 110, 130 (two places), 190, 210, 230 and 310 (4 places). Run the program to see how it works, and then continue reading.

```
•0     REM M/L 4
•10    FOR I = 1 TO 3
 70    HOME : VTAB 5
•75    PRINT "FOR ENTRY #"I
 80    PRINT : PRINT : PRINT
 90    INPUT "ENTER STARTING MILES, FINISHING MILES,
       AND LITERS CONSUMED (INSERT A COMMA
       BETWEEN EACH ENTRY) ";SM,FM,L
•110   MT(I) = FM − SM: REM MT = MILES TRAVELED
•130   G = L * .264200793 : MG(I) = MT(I) / G
 150   REM MG = MILES PER GALLON
 170   HOME : VTAB 10
```

```
• 190   RG(I) = INT (G * 100 + .8) / 100
• 210   LR(I) = INT (L * 100 + .8) / 100
• 230   MR(I) = INT (MG(I) * 1000 + .5) / 1000
• 240   NEXT I
  250   PRINT "     MILES "
  270   PRINT "TRAVELED LITERS      GALS       MPG"
  290   PRINT
• 300   FOR I = 1 TO 3
• 310   HTAB 3 : PRINT MT(I); : HTAB 12 : PRINT LR(I); :
        HTAB 22 : PRINT RG(I); : HTAB 32 : PRINT MR(I)
• 313   NEXT I
• 314   HTAB 32 : PRINT "- - - - -"
• 315   TL = (MR(1) + MR(2) + MR(3)) / 3
• 316   TR = INT (TL *1000 + .5) / 1000
• 326   HTAB 22 : PRINT "AVERAGE:- "; : HTAB 32 : PRINT
        TR
  330   PRINT : PRINT : PRINT
  350   PRINT " MORE? (Y/N) "
  370   GET M$
  390   IF M$ = "Y" GOTO 70
  410   IF M$ <> "N" GOTO 350
  430   END
```

In step 10 the FOR statement means that each time the step is included in the loop it will advance by one count, until it reaches 3. The loop is closed by the NEXT in step 240, which causes a jump back to step 10 for any value of I between 1 and 3. When I reaches 4, the program continues to step 250. We can understand how the three different values of I create three separate sets of values by analyzing the internal structure of the loop.

For each pass through the loop step 75 will read successively for entry 1, for entry 2 and for entry 3. In step 110, MT(I) will take on values of 1,2, and 3 for I so that the one variable becomes, in effect, three separate variables. Demonstrate this for yourself by running the program and stopping it by answering N to the last question. (This places you in the immediate execution mode, as opposed to the deferred execution mode, which operates with line numbered statements.) You can now print out the value of any variable. Type ? MT(1) and watch the number on the screen. Note that it is the same as answer 1 under MILES TRAVELED. Do the same for MT(2) and MT(3). Try any other variable in the program. Remember this technique for printing on the screen in the immediate execution mode because it is often useful for finding where the bugs are in programs that will not run correctly. It can also be used within programs as a temporary step to print key variables for debugging. A sophicated technique for checking the value of key variables is given in Chapter 5 (5-4).

Subscripted variables such as MT(1), add considerably to the flexibility of the programmer. We will be studying additional variations of these variables later.

As previously described, the program executes three passes through the loop, allowing you to input three separate sets of values and store three separate sets of answers. A second loop, embracing steps 300 through 313 prints out the stored answers. After the third pass the program proceeds through the PRINT and computation statements of steps 314 through 326 and reaches the stopping point.

3-5 USING THE CONDITIONAL STATEMENT FOR GREATER FLEXIBILITY

To modify M/L 4 to work with either liters or gallons as input data, change the steps to create M/L 5 as follows. Replace step 90 as shown. Add steps 115 through 145, replacing step 130 as shown. When you run this program, you may enter either gallons or liters, in any order, and get an answer showing both gallons and liters. You will note how UM$ in line 90 is used to define the input as a unit of measure in liters or gallons. Line 115 directs the program to the appropriate formula, either in lines 120 and 130, or line 140. The computation for miles per gallon takes place in line 145, after the number of gallons has been computed. From line 150 on, the program is the same as M/L 4.

```
•0     REM M/L 5
 10    FOR I = 1 TO 3
 70    HOME : VTAB 5
 75    PRINT "FOR ENTRY #"I
 80    PRINT : PRINT : PRINT
•90    INPUT "ENTER STARTING MILES, FINISHING MILES,
       GALLONS (G) OR LITERS (L), AND QUANTITY
       CONSUMED ";SM,FM,UM$,U
 110   MT(I) = FM - SM : REM MT = MILES TRAVELED
```

```
•115  IF UM$ = "G" THEN GOTO 140
•120  L = U
•130  G = L * .264200793
•135  GOTO 145
•140  L = 3.785 * U : G = U
•145  MG(I) = MT(I) / G
 150  REM LM = MILES PER LITER
 170  HOME : VTAB 10
 190  RG(I) = INT (G * 100 + .8) / 100
 210  LR(I) = INT (L * 100 + .8) / 100
 230  MR(I) = INT (MG(I) * 1000 + .5) / 1000
 240  NEXT
 250  PRINT "  MILES "
 270  PRINT "TRAVELED LITERS       GALS        MPG "
 290  PRINT
 300  FOR I = 1 TO 3
 310  HTAB 3 : PRINT MT(I); : HTAB 12: PRINT LR(I); :
      HTAB 22 : PRINT RG(I); : HTAB 32 : PRINT MR(I)
 313  NEXT I
 314  HTAB 32: PRINT "- - - - - -"
 315  TL = (MR(1) + MR(2) + MR(3)) / 3
 316  TR = INT (TL * 1000 + .5) / 1000
 326  HTAB 22: PRINT "AVERAGE:- "; : HTAB 32 : PRINT TR
 330  PRINT : PRINT : PRINT
 350  PRINT " MORE? (Y/N) "
 370  GET M$
 390  IF M$ = "Y" GOTO 10
 410  IF M$ <> "N" GOTO 350
 430  END
```

3-6 USING MULTIPLE COMPUTATIONS

You are now ready to start developing more complex practical programs. Let us define a manufacturing problem and create a program to simplify its management.

The Wooden Shoe Manufacturing Co. (WSM) wants to know the cost of making a certain shoe in various quantity production lots. They require 8.3 hours to set up the wood carving machine, which then turns out 500 pairs of shoes per hour. Each pair requires $1.275 worth of wood, and direct labor costs are $62.39 per hour.

The cost to the wholesaler will then be 3 times the direct manufacturing cost.

```
0      REM WSM 1
10     SU = 62.39 * 8.3 : PM = 62.39 / 500 : MC = 1.275
20     REM SU=SET UP COST, PM = MACHINE COST PER
       PAIR,MC = MATERIAL COST PER PAIR, UC = UNIT
       COST PER PAIR
90     HOME
100    PRINT "QUANTITY      MFG. COST "
105    PRINT
110    IF Q => 10000 GOTO 140
120    Q = Q + 1000
130    GOTO 150
140    Q = Q + 5000
150    UC = SU / Q + PM + MC
160    PRINT Q; : HTAB 12 : PRINT UC
170    IF Q <>50000 GOTO 110
180    END
```

A beginning solution is provided in program WSM 1. Type out this program, RUN it, and SAVE it. This program contains no new techniques, just some new formulas that are used to do new things. Study the program until you understand exactly why the program works the way it does and then try substituting other numbers in line 10 and see how they affect the manufacturing cost.

It will be useful to save the different runs, so let us use the printer to preserve each run on paper. To do this, change steps 90 and 180, as shown in WSM 2. Now when you run the program, your printer will print it out, at the same time it appears on the screen.

```
0      REM WSM 2
10     SI = 62.39 * 8.3 : PM = 62.39 / 500 : MC = 1.275
20     REM SU=SET UP COST, PM = MACHINE COST PER
       PAIR, MC = MATERIAL COST PER PAIR, UC = UNIT
       COST PER PAIR
90     ~~PR#1~~ PRINT CHR$(4) "PR#1"
100    PRINT "QUANTITY      MFG. COST "
105    PRINT
110    IF Q => 10000 GOTO 140
120    Q = Q + 1000
130    GOTO 150
```

```
  140  Q = Q + 5000
  150  UC = SU / Q + PM + MC
  160  PRINT Q; : HTAB 12 : PRINT UC
  170  IF Q <> 50000 GOTO 110
• 180  ~~PR# 0 : END~~ PRINT CHR$(4)"PR#0" : END
```

3-7 USING THE PRINTER FOR MULTIPLE REPORTS

The printer allows us to break free of the limitations of a 40 character per line screen so that we can now print out several columns of data to show WSM their costs and the selling price for various quantities per lot. Program WSM 3 does just that. There are many new steps in this program so type them carefully to avoid errors. As you may have found out by now, it takes much longer to find your errors than to make them. RUN the program, SAVE it, and then continue reading for the explanation.

Step 90 turns the printer on, using the commands introduced under the main heading 2-11. Step 100 and new step 103 have been expanded to include the extra column headings, and steps 152 to 158 introduce new variables to meet the expanded needs of the program. These variables are rounded in step 160.

Note that all computations are located between step 140, which establishes the changing value of Q, and step 160, which prints each changing line of the output of this program. Obviously, it is necessary to make the computation in that part of the program where all variables have been set to the correct value for the job to be done.

```
• 0    REM WSM 3
  10   SU = 62.39 * 8.3:PM = 62.39 / 500 : MC = 1.275
  20   REM SU=SET UP COST, PM = MACHINE COST PER
       PAIR,MC = MATERIAL COST PER PAIR, UC = UNIT
       COST PER PAIR
90 PRINT CHR$(4)"
• 90   PR#1" : PRINT CHR$ (9)"80N"; ~~: POKE 1913,1~~
• 100  PRINT "QUANTITY     MFG. COST"; : HTAB 26 : PRINT
       "HOURS "; : HTAB 36 : PRINT "MACH $ "; : POKE
       36,48 : PRINT "MAT'L $ "; : POKE 36,60 : PRINT
       "WHLS'L $ "
```

```
•103  HTAB 14 : PRINT "/UNIT"; : HTAB 26 : PRINT "/LOT";
      : HTAB 36 : PRINT "/LOT"; : POKE 36,48 : PRINT "/
      LOT"; : POKE 36,60 : PRINT "/ DOZ"
•105  PRINT " "
 110  IF Q => 10000 GOTO 140
 120  Q = Q + 1000
 130  GOTO 150
 140  Q = Q + 5000
 150  UC = SU / Q + PM + MC
•152  HN = 8.3 + (Q / 500) : REM MACHINE HOURS
      NEEDED PER LOT
•154  MD = HN * 62.39 : REM MACHINE COST PER LOT
•156  SD = Q * 1.275
•158  RS = UC * 3 * 12 : REM WHOLESALE COST PER
      DOZEN
•160  HR = INT (HN * 100 + .8) / 100 : SR = INT (SD *
      100 + .8) / 100 : RR = INT (RS * 100 + .8) / 100 :
      MR = INT (MD * 100 + .8) / 100
•165  PRINT Q; : HTAB 12 : PRINT UC; : HTAB 26 : PRINT
      HR; : HTAB 36 : PRINT MR; : POKE 36,48 : PRINT SR;
      : POKE 36,60 : PRINT RR
 170  IF Q <> 50000 GOTO 110
 180  ~~PR#0 : END~~PRINT CHR$(4)"PR#0":END
```

Data Storage and Retrieval

Chapter 4

Now that you have developed some feel for programming, let us begin to discuss the disk storage system and store data as text files for later retrieval.

Since many individuals may need a system to store names, addresses, and phone numbers, we will use an exercise to develop a very useful listing system, which can create a telephone directory and also provide mailing labels for Christmas card lists or other lists that require names, addresses, or phone numbers either for mailing or phoning friends or other selected groups.

4-1 PROGRAM DESCRIPTION

This is the longest, most complex program in this book, and before you go through all the time and trouble of typing it, you will want to know exactly what it will do for you. The program allows you to enter names, addresses, and phone numbers to create long lists. This program has been limited to 250 names per list, as explained later. That number of names is just about the practical limit for RAM storage and sorting with a program of this design and a 48K Apple II computer. Disk space is available for many such lists, filed under different names.

Enter the names in any order, and then, with the SORT option, arrange them alphabetically. This feature also allows you to enter names at the end of the list and later sort them into their alphabetical positions. You may print out this list in two different forms. In one form all data is printed on one line, making it useful as a telephone directory. In the other form only the name and address are printed, permitting you to make address labels. You may add to the list at any time, revise any line or lines of any file, and may even create special lists for everyone in the same city, or on the same street, with a minimum number of entries. This program even makes it easy to enter the name of your own city and other cities based on the assumption that most of your friends live in just these few places. Typing a single selected letter will cause a city name to print, saving you many keystrokes. Similarly, a single letter can generate street, road, place, etc., or area code and telephone exchange prefix, leaving you to type only the last four digits of each phone number.

Most important, however, is that you will learn many new techniques associated with data storage and retrieval so that you are better prepared to create your own programs.

4-2 GENERAL USE PROGRAMMING TOOL

Before starting the full program, let me introduce you to a tool that can be applied to every long program you write, to make editing and listing easier so that you catch and correct bugs as you work. This tool is a subprogram consisting of lines 20 through 45, 2150, and 2160. The first six lines create a menu that allows you to (1) Run the program in normal fashion; (2) Clear the screen and enter the compact editing mode described under the main heading 2-11; (3) Turn on the printer, print a complete list, and then turn off the printer and come to a complete stop, ready for more editing.

The subroutine at line 1330, called for in line 2150, is the printer on routine. You must complete this and lines 2150 and 2160 before you can use these edit menu options. You can then edit and list the incomplete program as often as you wish, just by typing RUN and hitting RETURN. You can revise line 2150 to read "LIST 1000, 1500" if you only need to list lines 1000 through 1500. When the program is complete and all bugs have been eliminated, you can delete this portion and run the main program in a normal way.

One new and very useful routine is used in this subprogram, at line 45. The command ON A GOTO means that for each value of A (1, 2, or 3) the branch will be to line 60, 2160, or 2150. This compact statement functions the same as three separate IF statements of the form: IF A = 1 THEN 60.

4-3 GENERAL INSTRUCTIONS

Type the whole program as carefully as you can. If you have any doubts about the accuracy of your typing, list the program on the printer and compare it with the program printed in the book. When you first run the program, choose option 1. You will be asked to specify a name for the file so that you can recall this particular file later. You can specify names for as many different files as you wish, using either the same name or any different name each time you run the program, thus creating many different lists. Enter only one or two names to check out the program. After the last name is entered, type QUIT and the disk storage system should immediately start filing. Note that you are asked for the first name, and then, in a second entry, the last name. The last name is separated so that all items can be sorted in the alphabetical order of the last name.

```
20    TEXT : HOME : VTAB 10
25    PRINT "1 RUN PROGRAM " : PRINT
30    PRINT "2 CLEAR TO EDIT " : PRINT
35    PRINT "3 PRINT PROGRAM LIST" : PRINT
40    HTAB 15 : INVERSE : PRINT "ONE DIGIT"; : NORMAL
      : GET A$ : A = VAL (A$)
45    ON A GOTO 60,2160,2150
60    REM   <<< LISTER 3.0 >>>
70    TEXT
80    DIM A$(6,250)
100   D$ = CHR$ (4):A = 1 : PRINT ~~: PRINT D$"MON C,I,O"~~
110   HOME : VTAB 5
115   PRINT DT$: PRINT
120   B = 6 : Y = C : X = 1
125   IF DR = 1 THEN FLASH
130   PRINT "A TO ENTER NAMES ON LIST "
135   NORMAL
140   PRINT
145   IF DR <> 1 THEN FLASH
150   PRINT "B TO READ LIST NAMES ON SCREEN "
155   NORMAL
160   PRINT
170   PRINT "C TO PRINT ADDRESSES "
180   PRINT
190   PRINT "D TO PRINT ADDRESS LABEL "
```

```
200  PRINT
210  PRINT "E TO ENTER NAMES FOR ONE STREET AND
     CITY "
220  PRINT "F TO REVISE A FILE " : PRINT
230  PRINT "G TO SAVE ON DISK "
240  PRINT : PRINT "H TO SORT "
250  PRINT
260  PRINT "I TO QUIT (AUTOMATIC SAVE) "
265  PRINT : PRINT "J TO CHANGE DATE "
270  PRINT : HTAB 10 : INVERSE : PRINT "LETTER?"; :
     NORMAL : GET PI$
280  PI = ASC (PI$) - 64
290  IF PI < 1 OR PI > 10 THEN PRINT CHR$ (7) : GOTO
     110
300  IF PI = 7 THEN J = 0 : GOTO 1130
310  IF PI = 9 THEN J = 0 : GOTO 1130
320  IF PI = 10 THEN GOSUB 2050 : GOTO 110
330  IF PI = 8 GOTO 1650
340  HOME : VTAB 10
350  IF DR = 1 GOTO 370 : REM DR= 1 AFTER DISK
     HAS BEEN READ
355  IF DR = 2 THEN DR = 0: GOTO 370
360  INPUT "WHAT IS THE NAME OF THIS LIST? : " ; LN$
370  IF PI = 5 GOTO 1550
380  IF PI => 2 GOTO 750
385  REM <<< DATA ENTRY >>>
390  J = Y
400  J = J + 1
410  IF Y < J THEN Y = J : C = Y
420  FOR I = A TO B
430  HOME : VTAB 5
435  JD = J - 1 : IF I > 1 THEN JD = J
440  PRINT "#"JD" "A$(1,JD) + " "A$(2,JD) : HTAB 6 :
     PRINT A$(3,JD) : HTAB 6 : PRINT A$(4,JD) : HTAB 6 :
     PRINT A$(5,JD) : HTAB 6 : PRINT A$ (6,JD)
450  PRINT : PRINT "FOR "LN$" #"J" : " ;
460  HTAB 20 : PRINT "ENTER "; : INVERSE : PRINT
     "QUIT"; : NORMAL : PRINT " TO EXIT"
470  PRINT
480  PRINT "ENTER "; : INVERSE : PRINT "DROP"; :
     NORMAL : PRINT " TO DROP NAME ABOVE "
483  PRINT
```

```
485    PRINT "ENTER "; : INVERSE : PRINT "LINE"; :
       NORMAL : PRINT " TO GO BACK AND CHANGE " :
       PRINT "PREVIOUS LINE"
490    PRINT : PRINT
500    ON I GOTO 510,530,550,570,620,625
510    PRINT "FIRST NAME? : "
520    GOTO 630
530    PRINT "LAST NAME? : "
540    GOTO 630
550    PRINT "STREET? : "
560    GOTO 650
570    IF PI = 5 GOTO 600
580    PRINT "CITY, STATE, ZIP? : "
590    GOTO 630
600    A$(I,J) = CY$
610    GOTO 720
620    PRINT "PHONE? : "
623    GOTO 630
625    PRINT "NOTES? : "
630    PRINT : PRINT
640    IF PI = 5 AND I = 3 GOTO 710
650    INPUT "ENTER DATA: " ;A$(I,J)
660    IF LEFT$ (A$(I,J),4) = "QUIT" THEN J = 0 : Y = Y -
       1 : C = Y : GOTO 1130
670    IF LEFT$ (A$(I,J),4) = "DROP" THEN Y = Y - 1 : J
       = J - 1 : I = 1 : GOTO 430
675    IF LEFT$ (A$(I,J),4) = "LINE" THEN A$(I,J) = " " :I = I
       - 1 : GOTO 2100
680    IF A$(4,J) = "E" THEN A$(4,J) = "ENGLEWOOD N.J.
       07631 "
682    IF A$(4,J) = "N" THEN A$(4,J) = "NEW YORK CITY
       N.Y. "
684    IF A$(4,J) = "S" THEN A$(4,J) = "ST. LOUIS
       MISSOURI "
686    IF A$(4,J) = "T" THEN A$(4,J) = "TENAFLY N.J.
       07670 "
688    IF RIGHT$ (A$(2,J),1) = "F" THEN A$(2,J) = A$(2,J)
       + "AMILY"
689    IF RIGHT$ (A$(3,J),1) = "S" THEN A$(3,J) = A$(3,J)
       + "TREET"
690    IF RIGHT$ (A$(3,J),1) = "R" THEN A$(3,J) = A$(3,J)
       + "OAD"
```

```
691    IF RIGHT$ (A$(3,J),1) = "A" THEN A$(3,J) = A$(3,J)
       + "VENUE"
692    IF RIGHT$ (A$(3,J),1) = "P" THEN A$(3,J) = A$(3,J)
       + "LACE"
693    IF RIGHT$ (A$(3,J),1) = "C" THEN A$(3,J) = A$(3,J)
       + "OURT"
695    IF LEFT$ (A$(5,J),1) = "A" THEN PH$ = "201-567-" :
       A$(5,J) = PH$ + RIGHT$ (A$(5,J),4)
696    IF LEFT$ (A$(5,J),1) = "B" THEN PH$ = "201-568-" :
       A$(5,J) = PH$ + RIGHT$ (A$(5,J),4)
697    IF LEFT$ (A$(5,J),1) = "C" THEN PH$ = "201-569-" :
       A$(5,J) = PH$ + RIGHT$ (A$(5,J),4)
698    IF LEN (A$(I,J)) = 0 THEN A$(I,J) = ". . . . ."
700    IF PI = 5 AND I = 3 THEN A$(3,J) = A$(3,J) + " " +
       ST$
710    PRINT
720    NEXT I
730    GOTO 400
740    REM <<< READ FROM DISK >>>
750    HOME : IF DR = 1 GOTO 860
760    DR = 1 : VTAB 10 : HTAB 9 : PRINT "READING DATA
       FROM DISK "
770    PRINT D$"OPEN ";LN$
780    PRINT D$"READ ";LN$
790    INPUT Y
795    INPUT DT$
800    B = 6
810    FOR J = X TO Y : FOR I = A TO B
820    INPUT A$(I,J)
830    NEXT I
840    NEXT J
845    C = Y
850    PRINT D$"CLOSE ";NM$
860    IF PI = 2 GOTO 895
870    IF PI = 6 GOTO 1020
880    IF PI > 2 GOTO 1280
890    NEXT J
895    HOME : VTAB 10 : INPUT "ENTER PAUSE LENGTH (0
       TO 2000) ";PL
896    IF PI = 6 THEN 910
897    C = Y : J =0
900    J = J + 1
```

```
910   FOR I = A TO B
915   IF PL = 0 GOTO 1000
920   PRINT A$(I,J);
930   IF I = 1 GOTO 1620
940   PRINT
950   NEXT I
955   IF CF = 1 GOTO 1960
960   IF PI = 6 GOTO 1070
970   FOR PAUSE = 1 TO PL : NEXT PAUSE
980   PRINT : PRINT
990   IF J < Y THEN 900
1000  GOTO 110
1010  REM <<< DATA REVISION >>>
1020  HOME : VTAB 5
1030  INPUT "WHICH FILE DO YOU WANT TO REVISE? "; J
1035  PL = 1
1040  HOME : VTAB 10
1050  GOTO 910
1060  HOME : VTAB 5
1070  PRINT
1080  PRINT "WHICH LINE DO YOU WANT TO CHANGE? " :
      GET I
1090  PRINT
1100  INPUT "TYPE NEW LINE: "; A$(I,J)
1110  CF = 1 : HOME : VTAB 10 : GOTO 910
1120  REM <<< WRITE TO DISK >>>
1130  HOME
1140  DR = 1 : VTAB 10 : HTAB 10 : PRINT "SAVING DATA
      ON DISK "
1150  PRINT D$"OPEN "LN$
1160  PRINT D$"DELETE "LN$
1170  PRINT D$"OPEN "LN$
1180  PRINT D$"WRITE "LN$
1190  PRINT Y
1195  PRINT DT$
1200  J = J + 1
1210  FOR I = A TO B
1220  PRINT A$(I,J)
1230  NEXT I
1240  IF J =< Y GOTO 1200
1250  PRINT D$"CLOSE "LN$
1260  IF PI = 9 THEN END
```

```
1270  GOTO 110
1280  REM <<< TO PRINT ADDRESS OR PHONE LIST >>>
1290  C = Y
1295  IF LEN (DT$) = 0 THEN GOSUB 2050
1300  PRINT "ONE LIST (1) OR ALL (2) ? "
1310  GET HL$
1320  IF HL$ = "1" THEN INPUT "WHICH LIST "; HL : X =
      HL : Y = HL
1325  GOSUB 1330 : GOTO 1360
1330  ~~POKE 54, 0 : POKE 55,198~~ PRINT D$ "PR#1 "
1340  PRINT CHR$ (9)"132N"; ~~: POKE 1913,0~~
1350  ~~PRINT CHR$ (27); CHR$ (20);~~ "Q"
1355  RETURN
1360  B = 6
1370  POKE 36,50 : PRINT LN$" " + DT$ : PRINT " "
1375  J = 0 : IF HL > 0 THEN J = HL - 1 : HL = 0
1380  IF PI = 4 THEN B = 4
1385  IF PI = 4 THEN 1470
1390  J = J + 1
1395  PRINT J; : POKE 36,5
1400  FOR I = A TO B
1410  IF PI = 4 GOTO 1430
1415  JJ = 56
1420  PRINT " "A$(I,J); : GOTO 1460
1430  IF I = 1 THEN PRINT A$(1,J)" " + A$(2,J) : I = 3
1440  PRINT A$(I,J)
1450  IF I = 5 THEN POKE 36,80 : PRINT "# "J" " + A$(6,J)
      : GOTO 1480
1460  NEXT I
1465  PRINT " "
1467  GOTO 1480
1470  FOR J = 1 TO Y
1473  C = Y
1475  PRINT A$(1,J) + " " + A$(2,J); : POKE 36,45 : PRINT
      A$(1,J + 1) + " " + A$(2,J + 1); : POKE 36,90 :
      PRINT A$(1,J + 2) + " " + A$(2,J + 2)
1476  PRINT A$(3,J); : POKE 36,45 : PRINT A$(3,J + 1); :
      POKE 36,90 : PRINT A$(3,J + 2)
1477  PRINT A$(4,J); : POKE 36,45 : PRINT A$(4,J + 1); :
      POKE 36,90 : PRINT A$(4,J +2)
1478  J = J + 2
1479  PRINT " ": PRINT " "
```

```
1480  REM PUT LINE SPACE HERE FOR DOUBLE SPACED
      PRINTOUT
1485  IF PI = 4 THEN JJ = 28
1490  IF INT (J / JJ) = J / JJ THEN GOSUB 1950
1495  IF PI = 3 AND J < Y THEN 1390
1497  IF PI = 3 THEN ~~PR#0 : GOTO 110~~ PRINT D$"PR#0":GOTO 110
1500  NEXT J
1510  X = 1 : Y = C
1520  POKE 54,240 : POKE 55,253
1530  GOTO 110
1540  REM <<< SET UP FIXED DATA ENTRIES >>>
1550  HOME : VTAB 5
1560  INPUT "WHAT STREET IS USED FOR ALL LISTINGS?
            ";ST$
1570  IF CL = 1 GOTO 390
1580  PRINT : PRINT : PRINT
1590  CL = 1
1600  INPUT "WHAT CITY IS USED FOR ALL LISTINGS?
            ";CY$
1610  GOTO 390
1620  HTAB 30 : PRINT "LIST #"J;
1630  GOTO 940
1640  END
1650  REM <<< SORT >>>
1660  Z = 1
1670  HOME : VTAB 10
1680  FOR J = Z TO Y - 1
1720  IF A$(2,J) > A$(2,J + 1) THEN 1820
1800  NEXT J
1810  GOTO 110
1820  HOME : VTAB 10
1830  FOR I = 1 TO B
1840  B$ = A$(I,J)
1850  A$(I,J) = A$(I,J + 1)
1860  A$(I,J + 1) = B$
1880  PRINT A$(I,J); : HTAB 20 : PRINT A$(I,J + 1)
1900  NEXT I
1910  Z = 1
1940  GOTO 1670
1950  FOR JS = 1 TO 10 : PRINT " " : NEXT JS : RETURN
1955  REM <<< REVISION OPTIONS >>>
1960  PRINT " "
```

```
1970  PRINT "1 TO CHANGE ANOTHER LINE : "
1980  PRINT "2 TO CHANGE ANOTHER ITEM: "
1990  PRINT "3 TO RETURN TO MAIN MENU: "
1995  PRINT "4 TO CHANGE LAST ITEM NO "
2000  GET CI
2010  ON CI GOTO 1070,1030,2040,2020
2015  GOTO 1070
2020  HOME : VTAB 10 : INPUT "ENTER NEW LAST ITEM
      NO. ";J
2030  Y = J : GOTO 110
2040  J = Y : GOTO 110
2045  REM <<< MISCELLANEOUS >>>
2050  HOME : VTAB 10
2060  INPUT "ENTER FULL DATE ";DT$
2070  RETURN
2100  IF I = 0 THEN HOME : VTAB 10 : PRINT "THE LINE
      FEATURE CAN BE USED " : PRINT "ONLY FROM
      LINES 2 THROUGH 5 " : PRINT "ENTER D TO
      CHANGE THE PREVIOUS ENTRY " : PRINT "ANY
      OTHER KEY TO CONTINUE." : GET A$
2110  IF A$ = "D" THEN Y = Y - 1 : J = J - 1 : I = 1 :
      GOTO 420
2120  A$ = " "
2130  GOTO 430
2140  REM <<< PRINT THIS LIST >>>
2150  GOSUB 1330 : LIST : END
2160  HOME : POKE 33,33 : END
```

To check whether the file works, select option 9 to end the program. The disk drive operates automatically whenever this program is ended to make sure that all data entered has been stored. To avoid filing incorrect data when you want to abort the program, hit RESET . When the cursor reappears, call for a catalog listing and notice that a new kind of file has been created, as identified by a ~~T~~ TXT ~~preceding~~ after the file name you specified for this list. The ~~T~~ TXT means that the file is a *text file*, which is different from the program file you have been using up to now. You cannot LIST a text file and can only retrieve it in a special way, as done in the READ FROM DISK section of this program. The text file just made should be at least ~~two sectors~~ 1 block long if the program has properly stored even one name.

To check the operation further, run the program again, selecting option 2 and using the same file name as before. All the names you listed should appear on the screen. You may then choose any other options to check out each of the features of the program. Note that you should always recall all data stored on the disk, by starting out with option 2 before adding other names to the file. The singular exception is when first starting a file, and no data is on the disk. Choosing option 2 in this instance will interrupt the program with an OUT OF DATA error message.

4-4 PRELIMINARY PROGRAM FUNCTIONS

Lines 60 through 280 provide the initial data constants and establish the main menu that permits user selection of each of the options of the program. The function of most of the constants will become clear as you study the program details. Line 70 clears the screen to the TEXT mode for the main body of the program, canceling POKE 33,33 and any prior graphics commands that may have been in use. The function of D$ in line 100 needs special explanation. The use of D$ permits insertion of CONTROL D [CHR$(4)] wherever it is needed to initiate a disk function. It is placed at the beginning so that it is available for both the writing to disk and reading from disk routines that follow.

~~The fourth statement in line 100 is a D$ command, which is used to order the display of disk commands, inputs, and outputs on the screen. This command is placed in line 100 so that you will be able to observe all disk functions as you study the operation of LISTER. If you want to avoid this display, replace the command MON with NOMON. Note that this command has been preceded by an extra PRINT command. All D$ commands must be preceded by a RETURN to assure execution, and since PRINT commands include a RETURN, the method used in this program is a simple way to provide this assurance.~~

Note line 80, which introduces the dimension (DIM) statement. The Apple computer normally allots space for ten numbers to each variable. To change from ten to any other number, the variable must be dimensioned with a DIM statement, as done here. The variable A$(I,J) is dimensioned to allow 250 lists of six items each. It is good programming to dimension all variables including those less than ten so that only the required amount of storage is used.

Line 280 converts PI$, which has no numerical significance as a string, into the numerical value of the digits that the string would ordinarily print out. You can observe this in a practical way by temporarily inserting a line 285 STOP. When you run this program with this line, the program will break at line 285, putting you in the immediate execution mode. You can then print PI$ + PI$ and see its value on the screen (1 & 1, or 11), and compare it with the value of PI + PI (1 + 1, or 2).

Line 290 prevents disrupting the program by accidentally entering a wrong number. Anything less than one or more than ten causes the bell to "beep" and sends the program back to the main menu.

4-5 DATA ENTRY

Lines 340 through 730 permit data entry. If option 1 is selected, then PI$ = 1, making the value of PI$ and PI both 1. If no data has been read in from the disk, then DR = 0 and line 350 will be passed, stopping the program at the INPUT statement of line 360. Whatever name is assigned to the program at this point becomes LN$. On PI = 1 the program continues to line 390, where both J and Y have initial values of 0. At line 400 J increases by 1, and at line 410 Y takes on the value of J. As you will see later, Y is used as a pointer, always carrying the highest value of J.

Line 420 allows I to go from 1 to 6, taking the values of A and B assigned in lines 100 and 120. I is used to control the number of data lines in the list (in this case six) and the number can be changed by assigning a different value to B in line 120. You might, for example, want to drop the sixth line (remarks) or add a seventh line for classification codes to permit sorting your lists for different purposes. It is easy to sort outputs this way by using an IF statement to compare your code with an input defined string, IF STRING = STRING THEN PRINT.

Lines 435 and 440 put the last entry, which is J − 1, or the partial current entry J on the screen so that you can check for errors and also keep track of your position. (If you do not see this now, do not worry about it. Your understanding of the many complex interactions of this program will have to grow slowly, and you can follow this concept better on a future reading.) Line 450 prints a heading for each listing to let you keep track of where you are in a long list.

It does this by printing the current value of J. Lines 510 to 620 print only one heading at a time for each value of I to show you which data line you are to enter for each list. Each time a heading is printed, the program is stopped by the INPUT statement of line 650, allowing you to enter the current value for A$(I,J). Study this part of the program carefully, as it illustrates the principle of storage by subscripted variables. Particularly, note how for each value of J, I has six values. Thus the values of A$(I,J) in the first listing will be A$(1,1), A$(2,1), A$(3,1), A$(4,1), A$(5,1) and A$(6,1). The next list runs A$(1,2), A$(2,2) and so on. This assigns a different variable for each line you enter, allowing up to 1500 different lines, which is 250 times 6.

You will understand the value of using subscripted variables better if you recognize that each such variable provides a readily addressable storage location. To address the name in any location you need only order the Apple to print, for example, A$(1,250).

If you enter the word QUIT at any point, then line 660 will set J back to 0, and Y back to the value of the last allowable entry. The program then advances to line 1130, the WRITE TO DISK routine so that all data is stored before any chance of accidental destruction arises. This is because the data stored in the subscripted variables is retained only in the random-access memory (RAM) of the Apple, and that data could be accidentally erased in a number of ways. If you are running a long list, it is a good idea to store groups of names on disk so that in the event of an accident you do not have to retype any more than the group you have been currently processing.

The reason for the LEFT$ (A$(I,J),4) condition in line 660 is that the word QUIT is only four letters long, but you might type an additional space or hit some other character before hitting RETURN. The conditional statement responds to only the first four left-most letters, as defined by the format in line 660, and as long as they equal QUIT the required condition is met.

A similar scheme is used in line 670 to make the program loop back to line 430 and drop the previous entry. If you enter the word DROP, Y and J are each set back by 1 and I is set to 1 so that the next entry will wipe out the last made entry by rewriting it. To delete a name, change line 2 to ZZZZ with the CHANGE routine and then use SORT to place that entry at the end of the list where you can drop it.

If no entry has been made for any given line and only RETURN has been pressed, then A$(I,J) will contain no letters (zero length), meeting the condition of line 698. You might do this if, for example, no phone number were to be listed. This causes the special blank line insertion (. . . . .) to be printed on any blank line. This feature was put in because it is easier to type RETURN than to type five periods and then RETURN .

Similarly, lines 680 through 686 will print the city, state, and zip code for your favorite city whenever you enter the appropriate single character for that city name. My home town and other selected cities are used for this sample, but you can change this line for any places you like. You will find this feature a great time saver if you use the cities most of your friends live in.

Line 688 will make a last name read "JONES FAMILY" if you type in "JONES F". The computer does this by examining the last letter of the last name, and if it is an F, adding the AMILY. In the unlikely event that you have a last name ending in F, you can prevent getting, for example, the BIFFAMILY, by typing BIFF followed by a space.

Lines 689 through 693 allow you to type just the first letter for street, avenue, etc., and let the program fill in the rest.

Lines 695 through 697 insert the three most common telephone exchanges in my area, together with my area code. To use this feature type A, B, or C and then the last four digits of the phone number. The complete number will be generated.

Line 730 loops the program back to the beginning of this routine, at line 400, after the last line of data (I = 6) for each listing has been entered. This program will continue adding list items until stopped by a QUIT entry or until the DIM limit of 250 is exceeded. Do not exceed 250 names per file, or you will interrupt the program with an error message and lose all recently typed data. If you need more than 250 names, use several files. When you QUIT entering data the program executes the WRITE TO DISK mode, which is the next area to study.

4-6 WRITE TO DISK

Lines 1130 to 1270 store all previously entered data in the disk operating system (DOS). The procedure WRITE TO DISK is extremely simple to execute, but you must do each step in exactly the right sequence to make it work. Line 1140 sets the flag DR to 1, indicating that the data now stored in the memory of the Apple computer includes all data stored on the disk. The initial value of DR is 0, and once it has been set to a value of 1, it does not return to 0 unless all data in the memory of the Apple is also set to 0. As you will see in the discussions of other parts of this program, the value of 1 is used to keep the Apple from going into an unneeded READ FROM DISK routine if the disk data is already in memory.

Line 1150 opens the active file (NM$), using the D$ command (CONTROL D), which puts the system in the DOS mode. Line 1160 then deletes everything in the file, to clear the way for fresh data. Line 1170 opens the file again and line 1180 commands DOS to write all succeeding data into storage.

Line 1190 is extremely important. The value of pointer Y has been previously set to a number corresponding to the last file number (highest J) in step 410. In order to properly recover the stored data, without getting an OUT OF DATA message, you must define exactly how much data has been stored. That is done by storing the value of Y, and since this value is required only once, step 1190 precedes the loop counting action that takes place between steps 1200 and 1240.

LINE 1210 steps I through six values, exactly as done in statement 420 during DATA ENTRY. Line 1220 writes each value for A$(I,J) into storage, in exactly the same sequence as originally done when the data was entered. So long as J is less than Y the program will keep looping from line 1200 to 1240, with a different value for J each loop. When the last item of data previously entered is stored, J will equal Y, and line 1250 will close the DOS. If you leave this step out, you can invite all kinds of difficulties when you try to run the program. Note that line 1250 includes the file name (LN$). This is optional in a program such as this, where only one file is active at a time. You can leave the file name out (as done in line 850 of the READ FROM DISK routine) saving typing time and storage space.

Line 1260 is included so that if the storage took place following a QUIT command in option 9, the program will now end. If the storage followed a QUIT command during option 1 or SAVE command in option 7, then line 1270 will return the program to the main menu.

4-7 READ FROM DISK

Line 750 will bypass the READ FROM DISK section of the program if data has already been stored in memory, as indicated by DR equaling 1. This bypass will happen when the data is stored, as previously discussed, or the bypass will happen when the data is read from the disk because of step 760. Once the data has been read it would be a waste of time to read the same data again. The only time the stored data must be read is when starting to add names to an existing list, which is normally done in conjunction with option 2.

Line 770 opens the file, and line 780 sets DOS to read data from the disk, just as if the data were coming from the keyboard. Line 790 INPUTS the previously stored value of Y, for reasons already explained. Line 810 will cycle the read mode through all values of J, from 1 through Y, and I from 1 to 6 (or any value of B you wish). The action is similar to that in the WRITE TO DISK section, except that you input all values of A$(I,J) in a read mode and save the data in a write mode. Line 850 closes the DOS file and the next three lines cause the program to branch as required to execute whichever option has been picked. Note the relationship of lines 870 and 880. If PI = 6, the program branches directly to line 1020; otherwise, for any value between 3 and 5, the program jumps to the print mode at line 1280.

If PI equals 2, line 860 will cause a jump to line 895 that starts a FOR-NEXT routine running through line 990 that prints each entry on the screen. Line 970 holds the display on the screen while it loops through 1 to 2000 or more operations, as defined by PL. The value of PL is established by the INPUT statement at line 895. If PL is set to 0, then line 915 will cause the program to skip past the display mode entirely.

4-8 CHANGING EXISTING DATA

Changing data is achieved by inputting fresh data to the desired A$, identified by specific J and I subscripts. Line 1030 INPUTS the value for J, and line 1050 routes the program back to line 910 to print all six lines on the screen. At line 960 option 6 is recognized, causing a jump to line 1070. This jump puts a blank space between the list and the INPUT statement, which asks for a value of I in line

1080. With I and J both designated, the single line A$(I,J) is then replaced by answering the INPUT statement of line 1100.

At this point in the program, let us add a few steps, and instead of inserting them here, by using the renumbering routine to create the space, let us use another technique. Line 1110 creates a jump via the display routine of lines 910 through 955, at line 955 to line 1960, where there is room for the new steps that conclude at line 2010. These steps provide a sub menu permitting additional changes or a return to the main menu. Even though this routine is an afterthought, fit it into the program by jumping from line 1110, adding the new steps, and then going to other program points, as in line 2010. Although this technique is a useful expedient, it is not good programming practice to jump around in illogical order and lose sight of your main flow.

4-9 PRINTING OUT DATA

If PI equals 2, 3, or 4, then the route is from the main menu to step 380, and from there to 750. If no data has been entered, the READ FROM DISK routine will execute, and at line 860 (even if the read mode was bypassed because PR equaled 1) the program will jump to line 1280.

Line 1330 turns on the printer. Line 1340 sets the line length to 132 characters and also turns off the screen to avoid problems. ~~This command is needed only with some printer cards, and (for printer slot # 1) should be POKE 1913,0, or POKE 1913,1, as discussed under main heading 2-11~~. Line 1350 is the small print command for the ~~Centronix 737~~ Apple printer. For other printers, see main heading 2-11. The small print size allows many characters to a line.

Line 1360 resets B to its normal value of 6 if B has been set to a lesser number following the execution of option 5. The value of B determines how many lines will print out and permits elimination of the lines containing the phone number and notes, when the address label format is selected by option 5. In that case, line 1380 sets B to 4. Line 1370 prints the list name and current date at the top of the list and then adds a blank space below the name.

Lines 1390 to 1400 set up the FOR-NEXT loops to print out the listings. If PI equals 4, there is a jump from line 1410 to 1415 where a page length constant (JJ) is set to 56. This allows 56 lines to print out. Then, line 1490 skips ten spaces creating top and bottom page margins.

Lines 1430 through 1450 make the print format consist of three separate lines. If PI equals 3, then line 1420 PRINTS with a semicolon to avoid line feed and a blank space between the quotes to separate the six strings that print consecutively on one line.

After the last J has been printed line 1520 turns the printer off and line 1530 causes a return to the main menu.

4-10 DATA ENTRY FOR ALL ONE STREET

On option 5 line 370 causes a jump to line 1550. Line 1560 establishes a value for ST$, which is the street name, and line 1590 causes a jump to line 390 only if the city has been previously selected, as indicated by a value of 1 for CL. If CL equals 0, CL is now set to 1 at line 1590, and then the INPUT statement of line 1600 establishes CY$, which is the city name.

Line 390 starts the data input routine, which is now modified because of the conditions just established. Line 570 causes a jump past line 580 to line 600, which sets all line 4's to CY$ without any additional keyboard input. At line 500 it is only necessary to supply the house number. The street name (ST$) is added in line 700. Line 570 jumps past the INPUT statement for CY$. Finally, line 700 changes the street line to the house number plus a blank space and ST$.

When no more names are to be added for a given street, execute a listing QUIT and return to the main menu. On the next execution of option 5, line 1600 will be bypassed because CL is still 1 at line 1590. Thus a new street name can be added without reinserting the city name. To change the city, execute an option 9 QUIT, saving all data on disk. When you rerun the program all variables will be set to zero, giving you a fresh start.

4-11 SORT

The sort routine is designed to illustrate the general techniques of sorting, as it provides a tool for processing the lists generated by this program. Because this routine has been designed to illustrate the basic principles of sorting, it is an elementary example and lacks the efficiency and speed of more refined programs. Although this routine may take an hour or more to sort a long scrambled list, it will do so effectively. Once you have learned how sort routines work, you will be able to choose from among the many better routines that are available and adapt them to your own needs.

4-12 GENERAL DISCUSSION

Programmers use many different routines, but all programmers follow the same general principles. Before analyzing the program studying the fundamentals will be useful. The sort process compares two items at a time and places one of them ahead of the other, based on some rules established for the list. In this example, the comparison is made on the last name of each person in the list, and the names are then placed in alphabetical order. The sort is done by a comparison such as line 1720.

If the comparison results in a decision to interchange the two names, then one of them (called B for purposes of this illustration) is transferred to a holding memory (B$ in this program) and name A is then transferred to the former location of B. To complete the transfer, name B then goes to the former location of A.

If that shell game was too much to follow, look at it this way. In step 1 of the sample below, name 1 is Ball, name 2 is Adams. A comparison shows these names are not in correct alphabetical order, so in step 2 Ball is entered into B$. In step 3 Adams replaces Ball as name 1. At this point, Adams is carried as both name 1 and name 2, and Ball is B$. B$ replaces Adams as name 2,

STEP	NAME 1	NAME 2	B$
1	BALL	ADAMS	----
2	BALL	ADAMS	BALL
3	ADAMS	ADAMS	BALL
4	ADAMS	BALL	BALL

in step 4, completing the transfer. Name 1 is Adams and name 2 is Ball. By moving names one pair at a time, an entire list can be jockeyed around until all names are in proper order. You will actually see something similar to the example in the box on the previous page flash by on the screen when you run this program:

4-13 TESTING LAST NAMES FOR PRECEDENCE

Selecting option 8 causes line 330 to direct the program to line 1650, starting the sort routine. Line 1680 starts the routine counting from the first name (Z) to the last (Y). Line 1720 tests each successive pair of names. If the first name in the pair A$(2,J) is greater than the second A$(2, J + 1), then the two names are not in correct sequence, and the program jumps to the sequence inversion routine at line 1820. (You could, of course, use A$(1,J) to sort on the first name.) If the sequence is correct, then line 1800 completes the loop back to 1830 and the next pair is tested. After the last pair has been tested and there are no more next J's for line 1800, line 1810 branches back to the main menu.

4-14 PUTTING NAMES IN ORDER

If sorting starts with all names in order, then no sorting takes place and the return to the main menu is almost immediate. If the names are completely scrambled, requiring a large number of sort operations, many minutes could go by before finishing.

Processing a long routine will give you an opportunity to observe how the Apple computer purges its string memory from time to time when that memory is filled to capacity. When a new name is assigned to a given string variable such as A$(2,8), the Apple does not clear the memory location but instead assigns the old name to another location. When the space reserved for string storage is filled, the Apple halts and takes time out to clear the memory locations of those strings no longer needed. As you run a long sort you will see the process stop after about 200 screen displays flash by. The display then freezes for about a minute while the garbage collection takes place. The actual sorting operation goes quickly, but the pauses for clearing occur frequently and use up much time. For this reason LISTER has been limited to only 250 names; more names than this would cause even more frequent pauses of even longer length.

To return to our discussion of line 1820 for the case where a sort is required, line 1820 clears the screen and starts a display ten lines down to show what is taking place. (On a long sort, an empty screen can be very boring; the screen message is good therapy for we insecure humans.) Line 1830 starts a FOR-NEXT loop to process all values of I for two names A$(I,J) and A$(I,J + 1) that are to be swapped. Line 1840 holds an A$(I,J) in B$, for each pass through the loop. Lines 1850 and 1860 complete the interchange sequence, and line 1880 displays the names just sorted on the screen.

Line 1900 steps the interchange through all six lines of the listing and then moves the program to line 1910. This resets starting number Z back to 1 so that the search can start from the beginning of the list.

Line 1940 causes a loop back to the start of the sort routine to examine the next item on the list. Note that line 1680 stops at one less than Y. This is necessary because the last two names on the list include both the next to the last name (Y − 1, or J) and the last name (J + 1, or Y).

Writing Your Own Programs

Chapter 5

5-1 GETTING STARTED

The programs you write will do the jobs you already know how to do. Start with simple problems, comparable to those in Chapter 1, and work your way up as you gain familiarity with the techniques and gain confidence in your skills. As you grow, you will learn that you have something unique; that is, the ability to generate programs that fit your needs better than anyone else can. Only you really understand those needs.

However, before I leave you completely on your own, there are a few general tips that should prove useful.

5-2 DEBUGGING

It is unusual to write a new program that works perfectly. Most programs exhibit bugs, when first run, and debugging is an essential skill for any programmer. Bugs will either cause an unexpected result, which will be obvious, or make the program break, printing an error message on the screen to tell you what caused the break.

Unless the program is very short, it is a good idea to print out a fresh LIST frequently as the program grows or changes are made so that you can see all the steps at once. You can usually trace the program route and discover what is wrong. As you gain experience, you will be able to locate bugs with increasing ease. There are a number of special features built into the Apple computer to make debugging easier; for example, the error message previously mentioned, and some of the other techniques discussed as you worked through the programs earlier in this book.

5-3 TRACE

When you RUN a program in the TRACE mode, the Apple computer will print out the line numbers of all program statements, in the sequence the program runs, either on the screen or on paper. Tracing the program lets you see whether the program is following your expected sequence so that you can make any needed corrections.

To initiate the trace mode, type TRACE before you RUN the program or insert a numbered line with a TRACE command at the program point you want the trace to start. When you are ready to shut off the trace, type NOTRACE.

5-4 PRINT/PAUSE

Even more useful than trace is the addition of a line such as

```
1275 PRINT X" "Y" "A$(X); : GET Q$: IF Q$="Q" THEN STOP
```

Insert this line at any point in your program that you wish to check. The program will stop at this point, with the value of all designated variables printed on the screen or printer if you are in that mode. When you press any key to GET Q$, the program will continue to run. This is especially useful to prove that the program passed through a particular set of program lines and is better than TRACE because it does not slow or interfere with normal operation.

If you want to stop the program to make changes or print out other variables, using the immediate execution mode, use "Q" RESET . Using any other key would continue running the program. Revising a program line will clear your variables, and you will not be able to continue running from the point you stopped. Use RUN and RETURN to restart the program from a full stop; use GOTO LINE XXX to pick up at an appropriate point (line XXX), or CONT to continue where you left off.

5-5 CONCLUSION

The LISTER program in Chapter 4 contains most of the elements needed to develop programs for any application you may have.

Once you have mastered that program, understanding what was done and why, you should be able to take off on your own, limited only by your imagination.

Experiment with variations of the programs in this book, adapting them freely to your own special needs. Particularly, adapt portions of the LISTER program to your own applications, developing as many practical variations as you can use.

Analyze the operations you have been doing with traditional methods and then create Apple programs to do them faster, better, and easier. As you do so, you will develop your programming skills and, as with any other skills you have mastered, you will be able to program with increasing ease and broader scope.

You will find many good books on the general techniques of programming, both in BASIC and more sophisticated languages. Now that you know how to converse in the dialect of the Apple computer, you will be more at ease with other dialects and can broaden your knowledge. Your first step from here will be to read through the books furnished by Apple Computer, Inc. so that you understand the finer points, which I have avoided repeating in this book. If you have already read these books, reread them now. Many things you may have missed before will now be clear.

Index