COMMODORE 128™ DATA FILE PROGRAMMING

DAVID MILLER

No. 2805
$21.95

COMMODORE 128™ DATA FILE PROGRAMMING

DAVID MILLER

TAB BOOKS Inc.
Blue Ridge Summit, PA 17214

Blessings crown the head of the
righteous . . . .

Proverbs 10:6

To Mark and Paul
Who have stuck with me
Through it all!

A special message of
appreciation and thanks
to Ron Powers
for his
constant patience,
understanding and confidence.

FIRST EDITION
FIRST PRINTING

Library of Congress Cataloging in Publication Data

Miller, David, 1944-
Commodore 128 data file programming.

Includes index.
1. Commodore 128 (Computer)—Programming.
I. Title.
QA76.8.C645M55 1987 005.265 86-23050
ISBN 0-8306-0205-4
ISBN 0-8306-2805-3 (pbk.)

Contents

Program List

Preface

What exercise is to the body,
Programming is to the mind.

The purpose of this book is to take some of the misery and mystery out of learning to use the Commodore 128's file structure. The book is aimed at people who would like to use the computer to assist them at home or at work by using the file capabilities of the Commodore 128 personal computer with an 80-column monitor. It is designed as a step-by-step tutorial. The book explains some things that, without adequate manuals, take many painful hours of trial and error to learn. Progress has been made in creating better file-handling techniques, and an explanation of some of these techniques is included.

Upon completion of the book, you should fully understand what files are and how to use them. You will be able to create your own sequential or relative access files. Examples of both of these file types are included throughout the book. Program examples include creation programs for the stock market, mailing lists, inventories, drill and practice, and medical records.

There are some very good database programs available commercially. If your needs require an elaborate database structure, you should probably use one of those programs or pay a programmer to create one for you. Reading this book will not make you capable of creating complete commercial database programs, but with practice you will be able to effectively create and use any type of file you want.

I really enjoy programming and creating programs for my own use. I like the freedom programming gives me, because I can easily change or add to what the program does. I hope this book conveys some of that enjoyment and freedom.

Introduction

Books . . . . (should) be read as
deliberately
and
reservedly
as they were written.
H.D. Thoreau

No book is magic. Merely by possessing the book, you cannot possess the knowledge of that book. Nevertheless I have tried to make it relatively easy for anyone to learn to meaningfully use the Commodore 128 personal computer.

No single book will suffice for everyone, and this book makes no claim to being the exception. But I have attempted to make it useful for the beginner, as well as the more experienced Commodore 128 BASIC user. The program examples cover the areas of home, education, business, hobby, and investment.

Computer vocabulary is introduced very gradually. Readers somewhat knowledgeable about the vocabulary might find the process repetitious at first, but I have found this to be the best method for acquiring a working knowledge of the multitude of "jargon."

The system approach has been used so the reader would not be overwhelmed with a large number of different application programs. The programs presented are intended to be useful as well as instructive. They build upon themselves, so that something that might appear awkward to an experienced programmer is used to help explain a concept needed in later chapters.

Information for the more experienced BASIC user includes relative-access files, automatic file access, initial use, file-creation techniques, sorting, and indexing.

You cannot just absorb this information. You must read the book, reread it, and study any text and programs that are, at first, unclear. Invest time in learning how to get the most out of the Commodore 128. Many people spend a great deal of time exercising their bodies. Programming is exercise for the mind. Therefore, the time you spend in learning and using the Commodore 128 might provide benefits that go beyond intended results.

Everyone seems to have difficulty with at least some aspect of using a keyboard to communicate with a computer. It does not seem to matter which computer it is or even how much typing experience a person has. And until better methods of human to machine communications are developed, we are stuck with learning to effectively use the keyboard. This initial learning process is often very important. Some people have become convinced that they cannot work with computers when, in fact, they are simply having difficulty with the keyboard. Therefore, it is imperative for new computer users to become as familiar with the keyboard as possible.

Experienced BASIC users might find that they can either skip parts or proceed quickly through certain sections. I would encourage everyone to finish the book.

Finally, a diskette containing all the programs presented in the book is available. You can make the diskette yourself by typing in all of the programs, but if you just want to see the programs in operation, you might want to purchase the diskette. I sincerely hope you enjoy the book and find it instructive.

Chapter 1

An Introduction to Commodore 128 File Types

There are as many definitions of the word "file" as there are kinds of files. You can quickly become confused if your understanding of the term differs from an author's intended use, and dictionary definitions are of little use in the computer world of today. Before becoming involved with the computer, my understanding of a file was limited to information that was kept in a folder in a file cabinet. I think we often learn best by trying to fit that which is new into something we already understand. Therefore, I will try to explain Commodore 128 file structure in terms of a file cabinet.

In a four-drawer file cabinet, one drawer might be for accounts payable, another could be for accounts receivable, a third for personnel information, and the fourth for inventory information, as shown in Fig. 1-1. These are used only as examples, to show that each drawer might contain different file types. The file cabinet just as easily can contain game instructions in one drawer, receipts in another, name and address information in a third, and medical records in the fourth. The idea is that a file cabinet contains different types of information. The Commodore 128's file cabinet is the disk drive and diskette. One type of file is a BASIC program file. A second file type contains only data or text. Other file types are identified by three-character abbreviations. Each diskette you use is like a file cabinet. It is set up to accept and classify these and other types of files, as illustrated in Fig. 1-2.

How do you know what files are on your diskettes? The tutorial part of this book will begin by going through all of the steps necessary to find out just what files are on your diskettes. (If you are already acquainted with the procedure used to start up the computer and disk drive, you can skip the rest of this paragraph.) First, turn on the 1571 disk drive. (A 1541 disk drive will work, but the book is written specifically for the 1571.) Second, depress the 40/80 DISPLAY key and then turn on the computer. The power light on the top

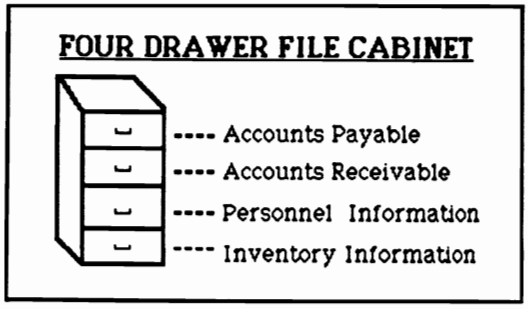


Fig. 1-1. A typical file cabinet.

left of the computer should come on. (This book assumes that you have a monitor capable of displaying 80 column text.)

If you have not yet turned on the monitor or TV, do so now. Take your "model 1571 test/demo" diskette and insert it into the disk drive with the label side face up; the label will be on the last part to go into the drive. (Other diskettes may be used, but the display of file information will be different.) Do not touch the cutout area on the diskette. Make sure the drive door is closed properly with the handle down. At this point, you should see the following message displayed (the message may vary slightly due to different production runs):

```
COMMODORE BASIC V7.0 122365 BYTES FREE
(C)1985 COMMODORE ELECTRONICS, LTD.
(C)1977 MICROSOFT CORP.
ALL RIGHTS RESERVED

READY.
```

It is especially important that you follow these steps in the order given. If you insert the diskette and then turn the drive and computer on, you will get a different message.

Get ready to press the key marked F3 in the upper right-hand part of the computer, but please do not press this key until specifically directed to do so. You do not need to press any other key, but if you want to delay the display, you can press the key labeled NO SCROLL next to the 40/80 DISPLAY key. Pressing the NO SCROLL key a second time causes the display to resume. After you are told to press the F3 key, the green light on the disk

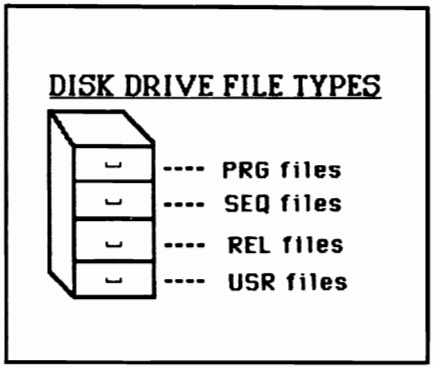


Fig. 1-2. Disk file types.

drive should come on. If you look quickly, you should see the word DIRECTORY below the word READY, which is followed by a list in three columns. Press the F3 key now.

The list should fill the screen, with the first part of the list disappearing off the top of the screen. The list will have three things on each line of text:

- ☐ A number representing the size of the file.
- ☐ The name of the file.
- ☐ A three-character file type abbreviation.

The abbreviation indicates what type of file it is: PRG for a BASIC program file, SEQ for a file containing only data in sequential order, USR for a user file, and REL for a file containing only data in random or relative order. The name, consisting of letters and/or numbers, is the actual name of the specific file. The size of the file is given as the decimal number of blocks or sectors (sections) in that file. In the file cabinet example, that is the information that might appear on each folder. When you create your own files for a file cabinet, you might label each folder with: (1) a name for the information within the folder, (2) how much information the folder contains, and (3) an abbreviation indicating which drawer it goes in.

Most file cabinets do not have a list of all the files stored within them. It would be a time-consuming job to update that list every time you added, changed, or threw away a file, but the Commodore 128's file management system does just that—and does it automatically. The diskette's list of everything in the file cabinet is what you get when you press the F3 key and get a DIRECTORY of a diskette. The DIRECTORY of a diskette shows the list previously created by the file management system.

TYPES OF FILES

How are these files used? What are they? How are they different? Returning to the file cabinet, the person in charge of that cabinet might put some rules or locks on the drawers. In other words, he or she might say that PRG files could be used only in a certain way or only by certain people. The same could be true of the other file types or drawers. This is exactly what Commodore 128's file management system has done. Each file type is used differently and can be used only in specific ways. PRG files are instructions (also called programs) to the computer to do something. Examples of such instructions are:

```
100 SCNCLR
200 IF NUMBER = THEN 6000
300 GOTO 5000
400 PRINT "Hello"
```

Most SEQ and REL files are not computer instructions but contain information of value to people, such as names, addresses, zip codes, payroll deductions, pay rates, or book titles. Often SEQ and REL files are just lists of such information. Such lists, of course, would not make sense as instructions to the computer.

It should be clear that Commodore 128 files are used to store information just as you use a filing cabinet, that there are different types of files, and that they are used for different purposes. The next chapter will look more closely at the file types and how they are used. In subsequent chapters, two of those file types, SEQ and REL data files, will be examined to see how information is kept, how that information is used, and how those files

are created. The creation of files is the main emphasis of this book, and will occupy the remaining chapters.

If you want to know how to create BASIC files (programs), you will need to learn programming. Effectively creating data files requires knowledge of how to program in some computer language. The programs discussed in this book will be BASIC programs, and the discussion will be such that anyone willing to try the examples should learn to program in BASIC, as well as learn to create and use data files. In other words, although the main emphasis of this book is on data files, you will learn a certain amount of programming in order to be able to create, display, and change data files. No matter what your age, background, or experience, if you are willing to try all the examples and read carefully through the discussion of them, you can and will learn to program and thus make effective use of data files. Programming and file manipulation are a matter of learning how to give instructions to the computer in a manner the computer can understand; or, more simply, programming is learning how to talk to the computer and tell it what you want it to do. The review questions at the end of each chapter (except Chapter 12) should help. The answers are found in the appendix.

CHAPTER 1 QUESTIONS

1. How are Commodore 128 file types identified?
2. What abbreviation is used to identify BASIC program files?
3. What abbreviation is used to identify data files?
4. Which file type will this book emphasize?
5. What should you press in order to see a list of the files on a diskette?
6. Which files contain instructions to the computer?
7. Which files contain information of value to people?
8. What information is shown when you ask to see a list of the files on a diskette?

Chapter 2

BASIC Program Files

One of the most common types of files is the BASIC file. Some of you might already be confused because you have always referred to BASIC *programs* rather than files. In reality, they are both.

Suppose one drawer in the file cabinet is used for games. Each folder contains the rules or instructions for playing a different game. Most of the time, you would simply refer to the folders as games, not files, but they are really both games and files. When you have taken one folder out of the file cabinet and are using the instructions to play the game, it is not a file; however, when you are finished with the game and want to put it back in its place, it becomes a file, one of many game files.

The same is true of BASIC program files. One drawer contains only BASIC programs or computer instructions (rules). When the computer is using the instructions in one of those BASIC "folders," the instructions are a program. When the computer is not using the instructions, they are stored as files. The important thing to understand is that BASIC files contain only computer instructions (programs). Some of these files contain larger or longer sets of computer instructions than others, but a BASIC file can only be a set of instructions for the computer, and therefore, a computer program.

This drawer, or BASIC file, can only be a certain kind of computer program—a BASIC computer program, not a Fortran computer program, or a Pascal computer program. (Fortran and Pascal are two other computer languages, just as BASIC is a computer language. There are more than 300 different computer languages, with variations of many of these 300 languages for different types of computers. In other words, BASIC on an Apple II computer is different from BASIC on a Commodore computer, or an IBM computer. Apple II BASIC is very different from the BASIC on the Commodore 128! Commodore has several different versions of BASIC for its machines, and most IBM computers are sold with two

somewhat different versions of IBM BASIC.)

BASIC FILE COMMANDS

Let's look at the rules for using BASIC computer program files. In our mythical office, there are three main secretaries who can use BASIC program files. Secretary number one can only get the file (LOAD). Secretary number two can only put the file away (SAVE). Secretary number three can get the file and immediately begin execution or operation of the program (RUN). *These three secretaries or commands do not have access to other drawers or file types.*

Secretary number three (RUN) does two jobs, loading a program into memory from a diskette and then beginning the operation or execution of that program. In other words, the RUN command gets a copy of the file from the disk, puts it in the computer's memory, and tells the computer to begin operation according to the instructions in the file (now a program).

Secretary number two (SAVE) can only put the file (program) currently in the computer's memory into the file cabinet (disk).

Secretary number one (LOAD) is only able to get the file (program) from the file cabinet (disk) and put it on the boss's desk (in the computer's memory). The LOAD command goes to the diskette and gets a specific file. In order to know which file to get, the LOAD command must be given the specific name of the file: LOAD "MATHDRILL, or LOAD"CHECKER. The quotation mark indicates the beginning of the file name. (MATHDRILL and CHECKER are used here only as examples of file names that could be used for BASIC program files.)

If the file name is not spelled exactly the way it is spelled in the file manager's list of files, the LOAD command will not be able to find the file. It will come back and tell you "?FILE NOT FOUND ERROR." On the other hand, if the LOAD command does have the exact name, it will go to the disk and get a copy of the file. Notice the use of the word *copy*. The LOAD command does not actually remove the file from the diskette, as a secretary would remove a file from the file cabinet. The LOAD command takes only a copy, so that the original always remains on the diskette. The copy of the file is loaded into the computer's memory, similar to a secretary getting a file and putting it on the boss's desk. You, the boss, must then decide what you want to do with the file. If you want to open it and look at it, use some form of the LIST command. (Type LIST.) If you want to see the program in operation, type RUN.

The secretaries, or BASIC file commands, must be given a specific file name. Usually, after using the LOAD command, you will want to look at the instructions (LIST) and perhaps change, add, or remove some instructions. When you have finished, you might want to keep what you have done by giving the file to secretary number two (SAVE) and telling the secretary the exact name you want this file kept under. If you have made changes but still want to keep the original currently on the disk, the secretary must be informed of a new file name.

If you use the same file name as the file currently on the diskette, a green light on the disk drive will begin blinking and the revised file will *not* be saved (unless you have provided additional information, an @ symbol, indicating that you do want the old file replaced). This might or might not be what you want, so be careful what name you use with the SAVE command.

Before you try some of these commands, you will need the test/demo diskette and a new diskette. If you do not have a new diskette, you should either wait until you do have a new diskette before doing the next steps, or use an old diskette that contains infor-

mation you no longer need.

The Commodore 128 uses a type of diskette known as a 5 1/4 inch, single-sided, 350K diskette. The number 5 1/4 refers to the physical size of the diskette, while the number 350K indicates that each diskette is capable of storing about 350,000 bytes, or characters, of information. This is an enormous amount of storage capacity when you think that a single-spaced, typewritten page may contain approximately 3,600 characters—60 characters per line and 60 lines per page. On this basis, each diskette is capable of storing the equivalent of approximately 100 single-spaced typewritten pages!

FORMATTING AND INITIALIZATION

Use the information from Chapter 1 to get the computer working, but this time insert the test/demo diskette into the disk drive before you turn on the computer. If the computer is already on, turn the computer off and then on again, or press the reset button located next to the power switch. After you turn on the computer, the disk drive will come on, make some noises, and soon the screen should display the following:

BOOTING C128 DOS SHELL . . .

After a brief time, the display will read:

D.O.S. SHELL ON F-1 KEY
READY

Get ready to press the F1 key, but do not press it yet. After you press the F1 key, the screen will show:

COMMODORE DISK UTILITY SYSTEM

PRESS SPACE TO SELECT YOUR LANGUAGE

ENGLISH

You will need to immediately press the space bar to select English as the language used in this program. If you do not press the space bar immediately, the language changes. You can then either wait until ENGLISH is displayed again, or start over using the reset switch. The program that produces this message is called the DOS SHELL program. DOS stands for Disk Operating System.

Now press the F1 key. As soon as the message shown above appears on the screen, press the space bar to select English. If you want to use another language, wait until that language name is displayed and then press the space bar.

After you have selected your language, the computer displays a series of phrases with instructions at the top of the screen: MOVE CURSOR THEN PRESS SPACE BAR TO SELECT, PRESS F5 TO UN-SELECT/STOP TO CANCEL. This display is called a *menu* of the possible choices you have. Each of the phrases is the name of a program that performs the function indicated by the name. FORMAT A DISK is the function you want. Move the highlighted, or light-colored, area using the arrow keys (either set) until the highlighting is over FORMAT A DISK, then, press the space bar. The computer instructs you to:

```
INSERT A BLANK DISK THEN PRESS SPACE
```

If you have the test/demo diskette in the drive, remove the test/demo and insert a blank or new diskette. If you do not remove the test/demo diskette, you could destroy the information on it. When you have inserted the blank diskette, press the space bar. After a few noises from the disk drive, the computer requests a name for this diskette: ENTER DISK NAME:. You can enter up to sixteen characters for the disk name. For this example, enter the name as DATA FILE 1, and then press the key marked RETURN. The green light and the disk drive come on for a short period of time. The computer is transferring numerical information onto the diskette to enable the computer to find locations on that diskette. Soon, the green light goes out and the highlighting returns to the first selection on the menu—DISK/PRINTER SETUP. This step is used to format a new diskette so that the diskette can store files.

The FORMAT command is usually used only once on each diskette. A second use erases whatever is currently on the diskette. The DOS SHELL program does not allow you to reformat a diskette, but Commodore has provided another way of formatting diskettes. I am going to explain how this second way, using the HEADER command, works, but you do not need to understand the process as long as you use the DOS SHELL to format all your diskettes. The explanation of the HEADER command includes some background information that is important. I would suggest, therefore, that everyone read over the explanation and not skip the next few paragraphs.

Direct formatting of diskettes is accomplished with the HEADER command. The complete instruction to format a diskette with a name of DATA FILE 1 is:

```
HEADER "DATA FILE 1", I87, D0, U8
```

followed by pressing the RETURN key.

All of the characters in this formatting sequence have a specific meaning. The word HEADER instructs the computer that you want to format a diskette. The quotation marks set off the name assigned to this diskette. Following the name, and a separating comma, the capital letter I is used to indicate that the next two characters are a diskette identifier. This identifier code is a unique feature to the Commodore line of computers. The code is used throughout the formatting of the diskette, so that the computer can always know if diskettes have been switched since the last diskette access. Therefore, no two diskettes should have the same two-digit code.

The characters D0 and U8 are optional. D0 indicates that this is drive number zero. Zero is used for the first drive and one may theoretically be used for a second drive. Use of the one is not possible with two 1571 drives, two 1541 drives, or a combination of both. The number one drive can exist only on a system that has both drives physically joined together—a dual-drive system. An alternative for a second drive on a C-128 is to change the device number of one of the drives. U8 indicates that this drive is device number eight. A single disk drive on a Commodore computer is usually assigned the device number of eight. (The keyboard is device number zero, the cassette recorder is device number one, modems are assigned device number two, the screen is device number three, printers are device numbers four or five, usually four, and disk drives are device numbers eight through 11, usually just number eight.)

THE "HELLO" PROGRAM

Now you're ready to try some of the BASIC commands. You are finished with the

DOS SHELL program, so exit the program by pressing the F1 key. The screen should clear, and the word READY should be in the upper left-hand corner of the screen. If you are unable to get out the DOS SHELL program, you can always turn the computer off, insert the newly formatted diskette, and turn the computer back on again. Resetting the computer should also return you to the BASIC prompt of READY.

Begin entering a BASIC program by typing the lines below. Remember to press the RETURN key after each line—{RETURN} below indicates that the RETURN key is to be pressed, not that those letters are to be typed.

```
NEW {RETURN}
100 PRINT {RETURN}
200 PRINT "HELLO" {RETURN}
300 PRINT "I AM THE COMMODORE 128 COMPUTER" {RETURN}
```

Check your typing to be sure you have typed everything exactly the way it is shown above. After you have finished typing a line, you must press the RETURN key. If you make a mistake in typing, you can use the INST/DEL key to DELete any characters on a line, or the SHIFT key and the INST/DEL key to INSert any characters. Remember to press the RETURN key after making any changes. You can always just retype the entire line, including the line number, because the computer stores only the last version of a designated line number.

The word NEW (in this case) erases any BASIC program that is already in the computer's memory. It does not do anything to the diskette or any information stored on a diskette. The numbers 100, 200, and 300 are line numbers in a BASIC program. Line numbers can be any number from 0 to 65535. Usually, the numbers chosen are not consecutive to allow other lines to be added, if necessary. The numbers indicate the order in which the computer is to execute the instructions, unless specifically told to alter that order.

The word PRINT instructs the computer to display on the screen whatever follows it and is between the quotation marks. Because nothing follows the word PRINT in line 100, a blank line will be displayed. Line 200 will cause the computer to display the word "HELLO." Line 300 tells the computer to display the sentence "I AM THE COMMODORE 128 COMPUTER."

Now, type the word RUN and press the RETURN key. (Instead of typing the word RUN, you can press the F6 key by holding the SHIFT key down and pressing the key labeled F5/F6.) Below the word RUN, you should see a blank line, the words between the quotation marks followed by a blank line, and the READY prompt.

```
HELLO
I AM THE COMMODORE 128 COMPUTER

READY
```

You have just written and executed a BASIC program. The program is still in the computer's memory, but if you were to turn off the computer now, you would lose that program. It would be lost because it exists only in internal temporary memory (often called RAM—Random Access Memory) and has not been permanently saved on a diskette. Information stored in RAM (inside the computer) will stay there only as long as the computer remains on, or until another program or data replaces the original program or data. The temporary and limited capacity of RAM is the main reason for external storage devices such as disk drives. These external storage devices provide "permanent" storage of val-

ued information and programs, in a form that can be brought back into the computer whenever needed.

The next step is to transfer (SAVE) the program in the computer's memory out to a file on the diskette. This is easily accomplished by giving the file to the SAVE secretary and letting the secretary do all the work. Again, the Commodore 128 allows more than one method of accomplishing this task. For those who like to use the function keys, you can press F5 and the screen will display:

DSAVE"

All you need to type then is the name of this program, "HELLO," and press the RETURN key.

If you don't want to remember all the function keys and what they stand for, you can type the entire command:

DSAVE "HELLO {RETURN}

({RETURN} indicates that the RETURN key is to be pressed, *not* that those characters are to be typed!)

The secretary or the SAVE command transfers a copy of the contents of the computer's program memory to the diskette and stores it in the file called HELLO. A message similar to SAVING 0:HELLO should appear on the screen, and the disk drive should come on for a very short time. You do not have to add the file type indicator, because this command knows to add the standard indicator of PRG for BASIC programs. The SAVE command has transferred a copy of the contents of the computer's program memory to the diskette, and stored it in the file called HELLO. If you want to see the names of the files on the diskette, you need to type one of the following commands:

```
   DIRECTORY {RETURN}
```

or

```
   CATALOG {RETURN}
```

or

press the F3 key without pressing {RETURN}

You should see:

```
0 "DATA FILE 1      ",87 2A
1      "HELLO"                  PRG
1327 BLOCKS FREE.
```

The number 87 will probably be different on your diskette. This is a randomly selected number—unless you specifically choose a number using the HEADER command.

To see what is in the computer's memory, choose one of the following: type the command LIST {RETURN}, or press function key seven (F7). Once again, the Commodore 128 provides more than one way of doing the same task. Regardless of the method chosen, the results should be the same. The display should show:

```
100 PRINT
200 PRINT "HELLO"
300 PRINT "I AM THE COMMODORE 128 COMPUTER"

READY.
```

LIST is similar to DIRECTORY in that DIRECTORY shows the contents of a diskette, and LIST shows what is in the computer's program memory. Now, type the word NEW {RETURN} followed by LIST {RETURN}. The program is now gone and there is nothing in the computer's program memory. The next step will bring back the program we have just erased from memory. Type:

```
DLOAD "HELLO {RETURN}
```

or, use function key two (F2—press and hold the SHIFT key and press the key with F1 on the top and F2 on the front) and then type:

```
HELLO {RETURN}
```

after the computer displays

```
DLOAD"
```

Then, when the screen displays READY, type:

```
LIST {RETURN} (or press F7)
```

Regardless of the way you choose to load the program, the program is back! Immediately after the DLOAD "HELLO, the disk drive comes on for a brief time and the computer displays the message:

```
SEARCHING FOR 0:HELLO
LOADING
READY
```

The computer is instructed to go to the diskette, bring in a copy of the file called HELLO, and store that copy in its program memory. When you type LIST, you are telling the computer to show you what it has in its memory. Therefore, the program now actually exists in two places: in the computer's memory, and as a file on the diskette.

Type carefully and add a fourth line like this:

```
400 PRINT "I AM A SMART COMPUTER." {RETURN}
```

Save this new program (in either of the ways specified above) as HELLO2

```
DSAVE "HELLO2 {RETURN}
```

Execute the DIRECTORY command and the list should show the addition of HELLO2.

```
0 "DATA FILE 1      ",87 2A
1      "HELLO"                PRG
1      "HELLO2"               PRG
1326 BLOCKS FREE.
```

There are two files on the diskette. Both are BASIC program files, or programs written using the BASIC language. After the SAVE command has been issued (in either method), the disk drive comes on briefly while the computer transfers a copy of the contents of its memory to the diskette. DIRECTORY shows the new list of files on the diskette. Finally, type:

```
NEW {RETURN}
```

```
LIST {RETURN}
```

The program is gone. Type:

```
RUN "HELLO2 {RETURN}
```

and the screen shows:

```
SEARCHING FOR 0:HELLO2
LOADING

HELLO
I AM THE COMMODORE 128 COMPUTER
I AM A SMART COMPUTER

READY.
```

Now type:

```
LIST {RETURN}
```

and the full program is back. Type:

```
RUN {RETURN}
```

Because the instructions are already in memory, a file name is not necessary. You should get the same message.

Let's review what you just did. First, you erased the program in the computer's memory (NEW). Next, RUN "HELLO2 told the computer to access the diskette, load the file called HELLO2 into its memory, and begin operation according to the file's instructions. LIST showed that the program is back in memory. To prove it, RUN told the computer to again

operate according to the program's instructions.

Now, let's review from the viewpoint of the secretaries and file cabinet. Remember, so far we have three main secretaries: number one (LOAD), number two (SAVE), and number three (RUN). For these secretaries to do anything, they must be given a file name:

`DLOAD "Math.Drill`	(Loads a file called math.drill into the computer's memory from diskette)
`RUN "Checkers`	(Loads a file called Checkers into the computer's memory from diskette and immediately begins execution of the program's instructions)
`DSAVE "Anything`	(Saves the contents of the computer's program memory out to the diskette with the file name of Anything)

BASIC file commands must be given a specific file name.

There are other BASIC commands that can be used with files. RENAME will change the name of any file. SCRATCH will remove, or erase, any file. CONCAT will combine two sequential files. DVERIFY makes a comparison between a program in the computer's memory and a program previously saved on diskette. COPY will make a backup copy of a file on the same diskette. COLLECT organizes a diskette and makes available any area possible. These commands are generally less used than LOAD, SAVE, and RUN, but are good to have when you need them.

I have used the concept of secretaries for two reasons. First, I believe it gives the impression that BASIC file commands are there to help you. In the examples, the secretaries are really BASIC file commands. BASIC file commands do certain things for you that a number of personal secretaries might do. The only limitation is that you must be exact and specific with the secretaries (commands). File commands must be used with a specific file name.

Second, the concept of secretaries can help clarify the use of a duplicate term. When the word RUN is used without a file name, it is no longer seen as a secretary, or BASIC file command. Instead, it is simply one of many internal BASIC commands. Typing RUN without a file name instructs the computer to execute the instructions currently in memory. RUN with a file name instructs the computer to access the disk drive, load the specified program into memory, and begin operation according to that program's instructions.

A lot of new information has been covered in this chapter. If something is not clear, you should review it and use the Commodore 128 and disk drive to better understand these concepts.

CHAPTER 2 QUESTIONS

1. True or False: BASIC programs are stored on diskette as files.
2. What three ways can be used to show a list of files on a diskette?
3. What does DOS stand for?
4. How many main file commands are used with BASIC program files?
5. Which file command gets the program from diskette and immediately begins execution or operation of the program?

6. Which file command stores programs on the diskette as files?
7. True or False: The LOAD (or DLOAD") command actually removes the program from the diskette and loads it into the computer's memory.
8. What happens when you save back to diskette a program you have changed, and you save it under the same name?
9. True or False: LIST shows what is on a diskette.
10. What file command is used to prepare a new diskette to receive files?
11. True or False: BASIC programs are never files.
12. Explain what NEW does.

Chapter 3

Data Files

Using the analogy of the file cabinet example in the last two chapters, this chapter is a quick look inside the DATA file drawer and at the two different kinds of file folders in this drawer. We will examine the characteristics that are common to both kinds of DATA files and look at how you can access those files.

BASIC program files contain instructions (called programs) for the computer. One of the other types of files usually contains information for people rather than machines. This does not mean that the computer cannot make use of the information, but that the information usually is not in the form of direct instructions for the computer. An example of an instruction for the computer is:

```
PRINT "Hello, how are you?"
```

An example of information that is not in the form of a computer instruction would be:

```
Title: Commodore 128 Data Files
Author: David Miller
Publisher: TAB Books
Address: Blue Ridge Summit, PA
```

This last example is the kind of information usually kept in a data file. Before you get into the process of actually storing and retrieving data files, you need to understand the main difference between the two kinds of data files.

SEQUENTIAL AND RELATIVE DATA FILES

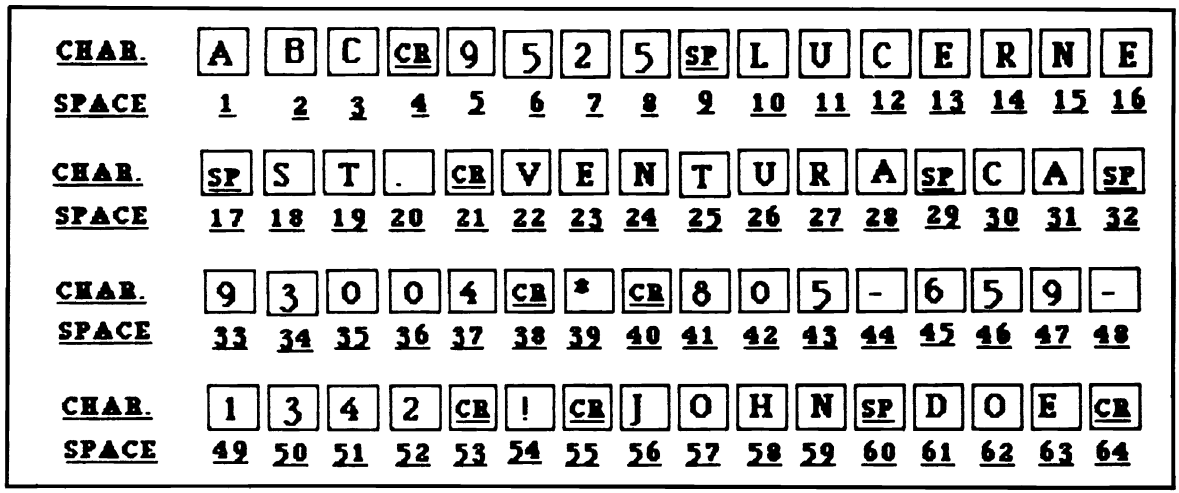
Data files have two ways of storing and retrieving information. (Remember that the information really stays on the diskette, and you are just getting a copy of the information.) These two ways of storing and retrieving information are *sequential access* and *direct access*. A sequential-access data file basically means that the information stored in the file is kept in sequential order, i.e., one right after the other as shown in Fig. 3-1. A direct-access data file usually means that each part of the file is divided equally, and can be reached directly and at random instead of going through all previous records. The process of looking at each record in order (sequence), to decide if it is the record you want, is a characteristic of sequential files. It can require more time than the direct-access method.

The basic difference between sequential data files and direct-access data files is somewhat like the difference between a cassette tape and a phonograph record. If you want to find a specific song on a cassette tape, even using the best available tape deck, you must begin at the current location of the tape and proceed either forward or backward, passing over all intervening songs until you have found the song you want. The process goes in sequence, one song after another. For example, if you want to play only the fourth song on the tape, you would have to advance the tape through the first, second, and third songs until you get to the fourth one.

On the other hand, if the songs are on a phonograph record, all you would have to do to play the fourth song would be to place the phonograph cartridge containing the needle at the start of the fourth division, instead of at the start of the first song. You can do that because you are able to clearly see the divisions between songs, and because those individual songs are directly accessible. You do not have to go through the grooves of the first three songs in order to get to the fourth, and moving the needle by hand only takes seconds.

Using this example, imagine that the DATA drawer contains two basic divisions. The first division contains files that operate in a way similar to cassette tapes, while the second division contains files that operate like phonograph records in the way described. (Commodore refers to direct-access files as *relative* files, but the term "relative" in the computer world often means "in relation to," and can be misleading when used as the name

Fig. 3-1. A record in a sequential-access file.



of the method for directly accessing records within a file. It is important to remember that, from now on, whenever I use the term *relative access*, I am talking about *direct-access* files.)

The lines below show how information is stored in sequential-access and relative-access files.

```
Sequential Access Storage:
1234567890123 123456789012 12345678901234
David Miller, Mark Miller, Paul D. Miller....etc.


Relative Access Storage (Length of 15 Characters per record):
123456789012345 123456789012345 123456789012345
David Miller--- Mark Miller---- Paul D. Miller-....etc.
```

Despite their differences, these two kinds of data files do have things in common, just like tapes have things in common with phonograph records. The most obvious common characteristic is that both files usually contain information that is not in the form of instructions for the computer. In other words, they contain information like lists, addresses, receipts, and inventories. Second, both files make use of some of the same BASIC file commands, but with different *parameters*, which are something like suffixes used with commands.

Because these data files are not computer instructions, they cannot be used in the same manner as BASIC program files. In other words, you cannot RUN, DSAVE, or DLOAD a data file. Those three commands, when combined with a file name, are the computer's means of access to BASIC disk files. The obvious question, then, is that if RUN, DSAVE, or DLOAD cannot be used with data files, how does the computer get the information on the disk in a data file, or back off the diskette from a data file?

In order to gain access to data files, you must use certain BASIC file commands in specific ways, depending on the kind of data file you are accessing. Both sequential and relative-access data files primarily use four types of commands:

(1) OPEN.
(2) CLOSE.
(3) Some way of reading the file (INPUT# or GET#).
(4) Some method of writing to the file (PRINT#).

Future chapters will examine in detail how each of these are to be used. For now, you need only understand the essential task of each command.

Again, the example of the filing cabinet is useful. In much the same way that a secretary must open a file folder removed from the filing cabinet before making use of any information contained in the file, so also must all data files be opened before the information they contain can be put to use. And, just as the secretary should close the file folder before replacing it in the filing cabinet, so should all data files be closed before ending the program or turning off the computer. If a secretary does not close the file folder, some information might drop out and get lost. The same is true if data files are not properly closed. This usually occurs only after new information has been written to the file and the file not closed. Loss of information should not occur if a data file has only been read and not closed.

INPUT#, PRINT#, or GET# are the main processes by which information is either read from or written to the file. If you only want to see information already in a data file, the BASIC file commands INPUT# and GET# are the commands you would use. If you want to

add information to the file, or create a new file, use the file command PRINT#.

SEQUENTIAL-ACCESS EXAMPLE

At this point, let's try out some of this information on the computer. Take the diskette (Data Files 1) that you formatted in the last chapter and place it in the disk drive. Then turn on the computer. If the computer is already on, reset the computer, or turn the computer off, wait a few seconds, and then turn it back on.

After the disk drive stops and the READY prompt appears on the screen, use one of the methods available to see a list of the files on the diskette (press the F3 function key, type DIRECTORY, or type CATALOG).

```
.DIRECTORY {RETURN}
```

or

```
CATALOG {RETURN}
```

or

F3 (No {RETURN} is necessary)

You should see:

```
0 "DATA FILE 1      ",87 2A
1      "HELLO"           PRG
1      "HELLO2"          PRG
1326 BLOCKS FREE.
```

Remember that DIRECTORY shows the names of the files on the diskette. Now type:

```
NEW {RETURN}
```

This clears the computer's memory.

Enter the following program. Check your typing very carefully. If you make a mistake, you can use the edit keys (the CRSR keys, arrow keys, and the INST/DEL key) to correct the mistake, or type the entire line over again. The computer will place the line in the proper sequence. Please notice, I have shifted into the upper/lowercase mode on the computer. I have done so in order to use a mixture of upper and lowercase characters in my screen displays for two reasons: (1) To give the screen displays a professional look, and (2) so that program users will be able to enter lowercase letters. The upper/lowercase mode is obtained by pressing down on the Commodore key, in the extreme lower left-hand corner of the keyboard, and at the same time pressing the SHIFT key. (You may have to try several times before successfully transferring the computer into this upper/lowercase mode.) You can shift back and forth by holding the Commodore key down and alternately pressing and releasing the SHIFT key. From this point on, most of the programs will be presented as written in this upper/lowercase mode. All the listings will also be displayed in lower-

case letters, so that you will be able to easily understand which characters should be entered as uppercase characters.

Type the following, and press RETURN after each line:

```
100  rem ***--write sequential data file example--***
110  :
120  :
130  rem **--output info. to file--**
140  dopen#1,"Seq.Data File",w
150  print#1, "The C-128 is more powerful than the C-64!"
160  dclose#1
```

Most Commodore program listings given in books or magazines display the listings in uppercase and use some procedure to indicate when an actual uppercase character is desired (such as underlining characters or words that actually are uppercase, or placing a special character before those characters that are uppercase). It is not necessary to follow this procedure, and I believe that it is actually much easier to enter programs from a listing that will exactly match the listing once it is entered in the computer. So, in addition to checking the 40/80 DISPLAY key when you turn on the computer, you should shift into the upper/lowercase mode immediately. I know that this might, at first, prove awkward. But the benefits of using lowercase within programs far outweigh the initial awkwardness of shifting from the normal uppercase-only mode into upper/lowercase mode.

Once your program matches the program above, save this program as "Write Seq.File". Type:

```
dsave "Write Seq.File {RETURN}
```

(or press the F5 key and type: **Write Seq.File {RETURN}**).

After you select and execute your method of saving this program, the program is stored on the diskette under the name Write Seq.File. In order to verify that the program is on the diskette, type:

```
directory {RETURN}
```

(Remember {RETURN} means press the RETURN key)

And you now see:

```
0 "data file    1       ",87 2a
1      "hello"                prg
1      "hello2"               prg
1      "Write Seq.File"       prg
1325 blocks free.
```

Now, list the program by typing:

```
list {RETURN}
```

or by pressing F7

just to see that the program is still in the computer's memory. Then type:

```
run {RETURN}
```

The disk drive comes on, but little happens on the screen. Once again, display the directory (type **directory** {RETURN}, type **catalog** {RETURN}, or press the F7 function key).

```
directory {RETURN}
```

just to see that the program is still in the computer's memory. The program now exists in two locations: the computer's memory and the diskette. Now, type:

```
run {RETURN}
```

The disk drive comes on, but little happens on the screen. Once again, type:

```
directory {RETURN}
```

This time you should see an additional file named Seq.Data File.

```
0 "data file    1        ",87 2a
1      "hello"                prg
1      "hello2"               prg
1      "Seq.File Example"     prg
1      "Seq.Data File"        seq
1324 blocks free.
```

You have created a data file! Even though you did not actually see the data file being written to the diskette, that is exactly what happened immediately after you ran the program. The reason you did not see anything on the screen is that your BASIC program told the computer to "print" (PRINT#2) your information to the diskette rather than to the screen.

Let's look at this program to see what each line does and what the correct syntax for each should be. Line 100 provides a name for this program. It is a good practice to make the name of the program similar to the file name under which the program is saved on the diskette. The only problem is that the file name cannot be more than sixteen characters long, while there is no limit to the length of the program name. REM is a BASIC reserved word meaning remark; it indicates that what follows is only a comment by the programmer, and will not be executed by the computer. The characters and words ***—Sequential Data File Example—*** are the actual comment.

Lines 110 and 120 contain colons that can be used to help separate sections of programs. Line 130 names the routine to follow. Again, it is a good idea to identify, with a REM statement, the task that the next lines are designed to accomplish.

The remark, in this case, indicates that you are creating a routine to write information to the diskette. Line 140 tells the computer that you are going to open a file called Seq.Data File on the diskette. The BASIC 7.0 file command **dopen**#1 tells the computer that you

are using the disk drive and that this is your first open file. The 1 represents an arbitrary number between 1 and 127, which is used to identify the file you are going to be using. This number can change from program to program, even though you are accessing the same physical file. In other words, the file number is not written to the diskette with the file name. It is just a number temporarily assigned to the file so that you can refer to that file by number rather than by name.

Notice the quotation marks around Seq.Data File. These quotation marks must be included in the program statements. That is how the computer understands where the specific file name begins and ends.

Next, the computer is told how you are going to be using that file; w is a parameter which indicates that information is to be *written* to the file. If the file is opened without the w parameter, the computer understands that the file is going to be read and not written to. Unfortunately, the format of this parameter is one area where BASIC 7.0 is not as good as BASIC 2.0. BASIC 2.0 allows programmers to spell out the word write so that there isn't any confusion. BASIC 7.0 only allows the single letter w. It is not a very significant point, but it is a step in the wrong direction regarding self-documenting programs.

Next, the computer is told that you are going to be adding information to the file with the PRINT#1 statement. Line 150 also tells the computer what information to put in the data file. Anything between the quotation marks will be written to the data file on the diskette. The PRINT statement in this line does not print the *string* (the information between the quotation marks) on the screen when the program is run, as would normally be expected. This PRINT statement contains the file number, telling the computer to print to the disk rather than to the screen. Line 160 contains the computer instruction to CLOSE the file numbered 1. This time, the file number is optional, because a CLOSE without a file number will close all data files that are open. If there is only one file open, that file will be closed with this simplified CLOSE statement. For clarity's sake, it is a better programming habit to include the specific file number with the CLOSE command.

You have now put information onto the diskette. The next task is to be able to read back from the diskette what was written. There are a number of different ways that you can read back the information, but for now, you will use the separate program approach. The Write Seq.File program is still in the computer's memory. Type:

```
new {RETURN}
```

to clear the computer's program memory. Then begin entering the following program (remember to press the RETURN key after each line):

```
200  rem ***--read sequential data file example--***
210  :
220  :
230  rem **--input info. from file--**
240  dopen#1,"Seq.Data File"
250  input#1,message$
260  dclose#1
270  :
280  :
300  rem **--display info. on screen--**
310  scnclr:rem clear screen/home cursor
```

```
320  char 0,0,9,message$:rem position cursor & print
                                          message
330  print
340  end
```

Check your typing. Then save this program under the name "Read Seq. File"

```
dsave "Read Seq.File {RETURN}
```

or press the F5 key and add the correct name.

Finally, type:

```
run {RETURN}
```

No file name is necessary because the program, besides being on the diskette, is also still in the computer's memory. This time the disk drive will come on for a brief time; then the screen will go blank, and the words "The C-128 is more powerful than the C-64!" will be printed nine lines from the top of the screen.

Let's examine each line of the new program. Line 200 names the program in a REM statement. Remember that the name on the disk should be similar to the name in line 200. Lines 210 and 220 are used to format the listing and, therefore, are not required. Instruction line 230 is another remark line providing a name for the *routine* (set of instructions) that immediately follows.

Instruction 240 in this program is identical to instruction 140 in the Write Seq.File program with one exception—line 140 has an additional parameter, w. The presence of the w indicates that the file will be written to, while the absence of the w indicates that the file will be read. Therefore, instruction line 240 tells the computer to open the file so that the file can be read from, rather than written to. Again this file is opened as the number one file, although you could use any number between 1 and 127. The file number is simply a matter of choice, and has nothing to do with the number of files open at any one time. The "d" preceding the word "open" is the BASIC 7.0 way of indicating that we are using the disk drive rather than a tape recorder or some other device. No space is permitted between the "n" in "dopen" and the "#." A comma must follow the file number. The file name is enclosed in quotation marks, and cannot be more than 16 characters long.

Line 250, **input#1,message$,** informs the computer that it is to bring into the computer's memory, the first (and in this case, only) piece of information in the file assigned the number 1. Again, no space is permitted between the "t" in "input," and the "#," and the file number must be followed by a comma. You can use the number one because you closed the original number one after you finished writing your information to it. Line 12 then tells the computer to bring in from that number one file (INPUT#1), a copy of the first (and, in this case, the only) piece of information in the file. The computer is instructed to store the information in the string variable memory location labeled message$.

A *string variable* is identified in BASIC by a character or characters followed by the dollar sign, $. *Variables* are names assigned to locations in the computer's memory. String variables can contain just about any value (i.e., numbers, letters, punctuation, and so forth). They are referred to as variables because their values may vary within a program; they are not constant. An example of a string variable might be Names$ where the first value

of Names$ is "Andy," the second value is "Mary," the third value is "Paul," and the fourth value is "Jane."

Therefore, you are storing the contents of the first and only piece of information in the Seq.Data File in the computer's memory location labeled message$. If you change the message in line 150 of the Write program, then the contents of message$ would also change when you ran the Read program again. Line 260 is the same as line 160. You close the file before proceeding to the display routine. Lines 270 and 280 again provide the separation between the various parts of the program, while line 300 provides the name for the next portion of the program. Line 310 instructs the computer to clear the screen and position the cursor at the top of the screen in the first column with the use of the command **scnclr**.

Finally, line 320 gives the instructions necessary to display the contents of the string variable message$. Line 320 prints the value contained in memory location message$ nine lines from the top of the screen. The computer prints this information on the screen because you have not told it that you want the information to go anywhere else. The screen is the *default* for such statements. Default, in this case, means that a certain value, the screen, has been predetermined, and unless the value is specifically changed, the predetermined value is taken as the desired value. The **print** in line 330 is used to separate the displayed information from the BASIC prompt of READY that appears immediately following the successful conclusion of a program.

This program is presented merely to give a brief explanation of the main file commands used with sequential-access data files. It is not intended to be a meaningful or useful program in any other sense. Such programs will begin later.

Finally, "clean up" the diskette so that you can put some serious programs on it in the next chapter. By following the instructions given below, you will gain practice in the use of two other BASIC file commands, and learn about the two modes in which the computer can operate. If you do not wish to erase these programs, you can skip what follows and start Chapter 4 with a fresh diskette, provided you remember to first format the new diskette. (Refer to Chapters 1 and 2 if necessary.) Type:

```
directory {RETURN}
```

and the list shows:

```
0 "data file    1       ",87 2a
1      "hello"                prg
1      "hello2"               prg
1      "Write Seq.File"       prg
1      "Seq.Data File"        seq
2      "Read Seq.File"        prg
1322 blocks free.
```

Now, carefully type just the following:

```
scratch"Seq.Data File {RETURN}
```

The computer responds with:

```
are you sure?
```

Type:

```
y {RETURN}
```

(or **yes**) to indicate that you are sure that you want to scratch (delete) this file. The disk drive comes on briefly and the computer informs you that:

```
01, files scratched,01,00
```

Then, type:

```
directory {RETURN}
```

and Seq.Data File is gone! Next, type:

```
scratch "hello {RETURN}

y {RETURN} (to indicate that you are sure)

rename "hello2" to "hello" {RETURN}
```

This will erase (scratch) one more file and change (rename) the name of another. As the last step, type:

```
dload "hello {RETURN}
scratch "hello {RETURN}
dsave "hello {RETURN}

dload "Write Seq.File {RETURN}
scratch "Write Seq.File {RETURN}
dsave "Write Seq.File {RETURN}

dload "Read Seq.File {RETURN}
scratch "Read Seq.File {RETURN}
dsave "Read Seq.File {RETURN}
```

This step puts these three programs back as the first three files on the diskette. Without this step, new files would be listed in the places vacated by the scratched files. To prove that these files are the first three on the diskette, load and run the Write Seq.File program. The sequential file created by this program should now be listed last—as the fourth file on the diskette. If you have followed these instructions carefully, the list of files should show:

```
0 "data file    1        ",87 2a
1      "hello"                 prg
1      "Write Seq.File"        prg
2      "Read Seq.File"         prg
1      "Seq.Data File"         seq
1323 blocks free.
```

All the steps you have just taken have been done in what is called the *immediate mode*; in this mode what you type is acted upon immediately after the RETURN key is pressed. The immediate mode (sometimes called the *direct mode*) can be very helpful in determining what errors exist in your programs.

Most of the work you will be doing will be in what is called the *deferred mode* (sometimes called the *indirect mode*)—deferred because the computer does not follow the program instructions until it is specifically told to act. This is the reason for the line numbers in programs. Line numbers tell the computer the exact sequence in which the computer is to follow the program instructions. In the next chapter, you will create your first useful program.

CHAPTER 3 QUESTIONS

1. Name the type of file that usually contains lists of information, rather than computer instructions.
2. Name the two kinds of data files.
3. Which kind of data file is similar to a cassette tape?
4. How many characters may a file name contain?
5. True or False: You can RUN a data file just as you RUN a BASIC program.
6. Give the number of modes in which the computer can operate and name them.
7. Name the four operations usually used with data files.
8. What does the reserved word REM stand for?
9. What three-character file type designator is used for sequential files?
10. Explain what the BASIC reserved word INPUT#, with a file number, does.
11. What symbol is used to designate string variables?
12. What are variables?

Chapter 4

Creating Sequential Files

You have seen that there are different types of files: BASIC program files, Fortran or Pascal (or any other language) program files, and data files. You should now understand that program files are files that contain specific instructions for the computer. Data files usually contain only lists or data, not computer instructions. You have also seen that there are two kinds of data files: sequential data files and relative-access (or random) data files. The difference between these two kinds of data files lies in the way the information within them is accessed: sequential data must be accessed one item after the other; relative files allow access to any record directly and immediately. In the preceding chapter, you were introduced to a new set of BASIC file commands common to both sequential and relative data files. Now, you are ready to put some of this knowledge to work and create some useful programs.

THE MAILING LIST SYSTEM

I am not going to get involved in the controversy over flow-charting and structured programming, but, when you build anything, you should have a plan. In this Mailing List System, you will need several programs. The first program should create the file. That program should have at least three different parts: an input routine, a correction routine, and a file creation routine. With this minimum plan, let's begin.

Start with what might be called the initialization statements. Make sure all your equipment is turned on properly, the Data File 1 diskette is in the disk drive, and the computer is in 80 column upper/lowercase mode. Then type:

```
new {RETURN}
```

Remember that **new** clears the computer's memory, making it ready for a new program. From this point on, I will not include the {RETURN} indication after each line, but you must remember to press the RETURN key after you have finished typing each line of instructions. Begin by typing:

```
100 ***--mail.create--***
105 :
110 :
115 rem **--initialization--**
120 :black  =   1
125 :white  =   2
130 :::red  =  11
135 yellow  =   8
140 ::chtr  =   5
145 bckgnd  =   0
150 :
```

Lines 120 through 145 provide a method of using descriptive labels for numeric values used in various BASIC commands. In other words, instead of using the command **color 0,1** you will use **color bckgnd,black**. The use of such descriptive labels makes it easier for anyone to actually "read" a program listing. The colons are used to help format the listing so that the lines are all straight and easy to read. Enter the rest of the initialization routine:

```
200 dim line$(20)
205 print chr$(14):rem upper/lower case mode
210 print chr$(8):rem disable char.set change
215 color bckgnd,black
220 :
225 :
```

Line 200 uses the DIM (DIMension) statement to reserve space in the computer's memory for 20 lines of up to 254 characters per line. This number of lines will be more than enough for a starter program. (In fact, a **dim** statement is not necessary unless you will have more than 10 lines of information for any single set of information lines.)

It is possible that someone using this program may forget to switch the Commodore 128 into the upper/lowercase mode. Line 205 makes certain that the computer is in the correct mode, regardless of whether the user has manually switched into the upper/lowercase mode. Instruction line 205 makes use of a new BASIC reserved word, CHR$. The value of the upper/lowercase mode as set by the code for all characters is 14. The code used in the Commodore 128 and most micros is called ASCII. It sets a decimal and hexadecimal value for all characters used. If you look on page 357 of the Commodore 128 System Guide and go down the page under the column PRINTS until you find the statement "SWITCH TO LOWER CASE," you will find a column to the right with the number 14. Therefore, the value of the SWITCH TO LOWER CASE key (Control N) is 14. The reserved word CHR$ can be used with parentheses and a decimal value to issue commands to the computer.

Commodore also allows you to issue commands to the computer by entering nonalphanumeric characters into a program line. These characters are then displayed as graphic

symbols both on the screen and in a printed listing. I will avoid using these graphic symbols because they are not descriptive of their functions.

One other major problem occurs with the use of Commodore's graphic symbols. Many printers will not actually print these symbols properly. Unless you have the Commodore printer or a printer with the right interface (equipment that translates information from the computer to the printer), you are likely to get a variety of strange characters and/or sounds. Even those users with either the Commodore printer or a printer with the right interface find it difficult to read a printed program listing, and harder still to type such a listing into their computer. Entering such a listing also requires remembering the exact sequence of key strokes needed to obtain the graphic symbol for these keys. Therefore, whenever possible, I will use the ASCII value for various functions that do not have a BASIC reserved word already established to accomplish the purpose of the function.

Line 210 addresses a similar situation. This instruction ensures that the user does not switch out of the lowercase mode. Line 215 has a command new to the Commodore line of computers. With the color command, a program can set the color of a number of different things in the computer. This instruction sets the color of the screen background to black. A little later, you will use the command to set the color of individual characters and/or text. There are sixteen colors available for the background and the text. This feature on the Commodore 128 is one of its best. Such versatility and flexibility are unusual even for more expensive computers. At the same time, misuse of color can be harmful to proper operation of the program. Misuse will not hurt the equipment, but program users might experience eye strain at the very least.

THE KEYBOARD INPUT ROUTINE

The second section of the program involves the instructions that enable a person to type in the information that will eventually be stored in a data file. There are many ways to accomplish this task, some much better than others. For example, probably the simplest method is:

```
INPUT A$,B$,C$,D$
```

But a person using this method would be faced with:

```
?
```

on the screen. In other words, the program user would be given no instructions regarding what to type, how to end the process, or how to correct entries. On the other hand, such a program could be very detailed with extensive programming to catch potential errors entered by any program user. I have tried to take an intermediate approach, so that the method introduced here provides some instructions and error traps, but is still of reasonable length as a teaching program. The keyboard input routine is designed to produce the following screen display (with an example name and address):

```
INSTRUCTIONS
```

Type the name and address as if you are addressing an envelope. Do not use a comma or a colon! Press RETURN after each line. Type the word END on a separate line when you are finished.

Type in line 1: David Miller
Type in line 2: 9999 Any St.
Type in line 3: Somewhere, CA 90000
Type in line 4: end

Press RETURN if there is no phone #.

Type in Phone #: 800-444-1111

The BASIC instructions necessary to achieve such a screen display are not extensive or difficult to understand, but do involve a number of commands and statements that need some explanation. Type the following:

```
300 rem **--keyboard input routine--**
305 k = 1:rem line counter
310 scnclr
315 char 0,22,2
320 color chtr,white
325 print "INSTRUCTIONS"
330 print
335 color chtr,yellow
340 print "Type the name and address as if you are
        addressing an envelope."
345 print
350 color chtr,red
355 print chr$(2);:rem underline
360 print "Do not use a comma or a colon!"
365 color chtr,yellow
370 print chr$(130);" ";:rem underline off
375 print "Press ";
380 color chtr,white:print "RETURN";
385 color chtr,yellow:print " after each line."
390 print
395 print "Type the word ";
400 color chtr,white
405 print "END";
410 color chtr,yellow
415 print " on a separate line when you are
        finished."
420 print:print
```

Line 300 labels the routine and line 305 uses a counter "k" to keep track of the information lines used. Line 310 uses the instruction "scnclr" to clear the screen (SCreeN CLeaR) and place the cursor in the upper left-hand corner. Line 315 positions the cursor 22 spaces from the left side of the screen and two lines from the top of the screen. The "char" instruction is very powerful; it provides parameters to display characters on different screens and at any location on those screens. In addition, char allows programmers to include text. Since I want to set the color of the text (line 320) and be consistent with the other print lines, I will rarely include text with the char instruction.

"Color chtr,white" instructs the computer to change the color of text, or CHaracTeRs, to white. The choice of letters for this variable is important. The normal abbreviation for the word "character" is "char." But BASIC has a reserved word CHAR. One of the rules governing variables is that no variable can match a BASIC reserved word. Therefore, you cannot use "char" as a variable, and must choose other letters that are close to being descriptive. I decided that the letters "chtr" were recognizable as an abbreviation for the word "character," and have used this combination in all color instructions that change the color of text.

The title is given in line 325 and is also displayed in uppercase letters. Line 335 instructs the computer to use yellow for the text that follows. Lines 340-415 provide instructions to the person entering information, in this case, names, addresses, and phone numbers. All of these lines are optional, but I have included them because they help the user enter the information correctly. Of course, the instructions themselves can be reworded to your own preference.

Line 350 sets the color for the text to red, and line 355 instructs the computer to underline the text that follows. The semicolon is necessary to indicate that the text will immediately follow. Line 360 includes the text that is to be displayed in red and underlined. The user is told not to include a comma or a colon in the information. The reason for this prohibition involves the way information is stored on the diskette. The comma and colon are used as *delimiters*, i.e., they signify the end of the information. Information that follows a comma or colon is viewed as belonging to the next variable or set of information. For example, in addressing an envelope, the city and state are usually entered on the same line and are separated by a comma. If the user were to type the city and state on the same line and separate them with a comma, the computer would believe that the user had actually entered two lines. The state would then be stored on the diskette as a separate line. There is a way to get around this problem, but it involves a considerable amount of extra programming in both the storage and retrieval of information. When you learn to use relative files, the comma and colon will be less of a problem.

Line 365 returns the text color to yellow, while line 370 informs the computer that you are finished underlining. The chr$(130) instruction is followed by a pair of quotation marks with a single space in between. The reason for this space is that, without it, the sentence in line 360 and the next sentence would be joined together.

Line 375 first displays the word "Press" and a blank space, using the previously established color for text (yellow), but the computer is instructed to switch to white for the following word. The word "RETURN" in line 380 and the word "END" in line 405 are, therefore, printed in white on a black background. Immediately after the printing of these words, the text color is returned to yellow. Throughout the programs in this book, I will attempt to highlight keys or characters the program user should press or type in a similar manner. Such highlighting helps the program user to specifically follow the instructions given. Again, conservative and appropriate use of color is best.

Line 385 contains a space immediately after the quotation mark, because without that space, the "a" in "after" would be placed right next to the "N" in "RETURN" ("RETURNafter" instead of "RETURN after"). The semicolon instructs the computer to leave no space between the last displayed character and the next displayed character.

Continue entering the keyboard input routine by typing:

```
425 print "Type in line ";k;":";
430 gosub 9000:rem get key subroutine
```

```
435 if t$ = "end" or t$ = "END" or t$ = "End" then 500
440 if t$ = "" then print:print "We need some
          information.":goto 425
445 line$(k) = t$:t$ = ""
450 print
455 k = k + 1
460 goto 425:rem next line
465 :
470 :
```

These lines are the heart of the keyboard input routine. Line 425 tells the user which line is being entered. The phrase "Type in line" is displayed. The semicolon instructs the computer to print the value of the numeric variable k immediately following the quotation mark.

The normal procedure following line 425 is to use the input command. The problem is that the input command displays a question mark. But line 425 does not contain a question. Instead, this line is instructing the user what to do. Consequently, it would be better to not have a question mark displayed.

The GOSUB instruction in conjunction with the RETURN instruction (very different from the RETURN key) allows a programmer to transfer control from one location in a program to another location, process the instructions in that subroutine, and then return control to the instruction that follows the GOSUB statement. This instruction is especially useful for often-used subroutines. These often-used subroutines can be located at the end of a program and then called from various locations within the program. Once again, this flexibility can be misused—and often is. A program that contains many undocumented GOSUBs can be very difficult for even the original programmer to read and decipher later.

The getkey subroutine, which is presented in the section about subroutines, eliminates the question mark. This subroutine accepts whatever is typed in and stores it in t$. T$ is then checked in line 435 to see if it equals "end," "END," or "End." Because the user could type in either uppercase or lowercase, you should check for both, as well as for the capitalization of only the first letter. If t$ does equal some form of the word "end," the computer is instructed to jump ahead to line 500 (the get phone number routine) immediately. If none of the conditions is met, the line is ignored and the computer goes to the next instruction in line 440.

Line 440 checks for a response that contains no information. Since it is possible for a new user to accidentally press the RETURN key more than once, you should make it impossible to do so. Without the check in line 440, such a mistake would result in a number of blank information lines. If the condition is not met, this instruction is ignored and control passes to the next instruction.

Line 445 transfers the contents of the string variable t$ to the reserved memory variable line$(k). The value of k depends on the number of the line. Remember that the computer was told to reserve space for 20 possible lines of line$. Reserving multiple spaces for a variable creates an *array* for that variable. T$ is then immediately reset to equal a nul value (have no value).

Line 455 is a method of increasing the line count. The first time through, k will equal 1, so the formula really is k = 1 + 1, or 2. Once you have increased your line count, you want to go back and get another line. This is exactly what line 460 does. When the user has entered some form of the word "end" on one of the lines, the computer jumps, or branches, to the routine that requests the phone number.

Next, type the following:

```
500 rem **--get phone number--**
505 line$(k) = "*":rem separator for phone number
510 t$ = ""
515 print:print
520 k = k + 1
525 print "Press ";
530 color chtr,white:print "RETURN";
535 color chtr,yellow:print " if there is no phone #."
540 print
545 print "Type in Phone #: ";
550 gosub 9000:rem getkey subroutine
555 line$(k) = t$:t$ = ""
```

These might be the most confusing lines to understand. In order to easily separate the name and address from the phone number, I have included a separator, "*", on a line by itself. The reason for separating the phone number from the rest of the information is that now you can use the first part of your information to produce mailing labels. I have also included a separator, "!", to easily differentiate between the name, address, and phone number of one person and the name, address, and phone number of the next person. Therefore, line 505 sets the kth line of line$ equal to *. At this point, if the first line contains the name, the second line the address, and the third line the city, state, and zip code, then the fourth line will contain the word "END," and k will be equal to 4. By making the fourth line equal to *, you have actually accomplished two tasks: eliminating the word "END," and establishing a one-character separator before the phone number. The user could have been required to type the asterisk when he or she finished entering the name and address, but it is preferable to have the user type something natural within the context. Line 510 again resets t$ to a nul value, and line 520 increases the line count by one for the phone number.

Lines 525-535 give instructions about typing in the phone number. The word "RETURN" is displayed in white. Again, the getkey subroutine accepts whatever format the individual uses to type in the phone number and stores the information in the string reserved memory. If the user just presses the RETURN key, then line$(k) is set to equal the word "none." Line 565 again increases the count by one, this time for the separator between sets of information. Line 570 makes the kth line of line$ equal to !. If the fourth line of line$ is * and the fifth line is the phone number, then k would be 6, and the sixth line would equal !. This concludes the input routine.

THE CORRECTION ROUTINE

The correction routine is next. Once again there are a number of ways to program correction routines, ranging from none at all to very elaborate single-character editing. The system used here does not allow the user to retype just a single letter (instead, the entire line must be retyped), but the user is given ample opportunity to make any necessary corrections or changes. The correction routine presents the following screen display (with example information):

Do not change the line with the *. This symbol is used as a separator.

1 David Miller
2 9999 Any St.
3 Somewhere, CA 90000
4 *
5 800-444-1111

Change any line? Type a Y or N: y

Change which line? 2

Line 2 now is: 9999 Any St.

Line 2 should be: 9999 Any Street

Do not change the line with the *. This symbol is used as a separator.

1 David Miller
2 9999 Any Street
3 Somewhere, CA 90000
4 *
5 800-444-1111

Change any line? Type a Y or N: n

Type the following to begin entering the instructions for the correction routine:

```
700 rem **--correction routine--**
705 scnclr
710 char 0,0,3
715 print "Do not change the line with the ";
720 color chtr,white: print "*";
725 color chtr,yellow:print ". ";
730 print "This symbol is used as a separator."
735 print
```

These lines explain what the routine is, set the format for the correction routine, and give instructions to the user about the separator *. Line 700 labels the routine; lines 700-705 clear the screen and place the cursor in the upper left-hand corner three lines from the top. Lines 715-730 print the instructions for the user. Line 735 prints a blank line after the instructions.

```
740 for i = 1 to k - 1
745 print i;" ";line$(i)
750 next i
```

Lines 740-750 make a loop used to get the information in the string reserved memory and print that information on the screen. Line 740 is the first line of a FOR-NEXT loop. It uses a counter (i) that starts with the value of one and counts to the value of k minus one.

In the example entry, the sixth line was the last line and so was set equal to !. Because that line should not be changed, there is no reason to display the line. Therefore, the counter only goes to k – 1, or 5. Line 745 prints the current value of the counter, a blank space, and then the information contained in line$(i) stored in the computer's memory. Line 750 increases the counter by one until the counter equals the value of k – 1. The instructions between the FOR instruction and the NEXT instruction are executed the number of times specified in the FOR instruction. To conclude the correction routine, type:

```
755 print
760 print "Change any line? ";
765 gosub 8000:rem y/n input subroutine
770 print
775 if yes$ = "y" then 800
780 if yes$ = "n" then 790
785 :
790 goto 3000:rem file creation subroutine
795 :
800 rem *--change line--*
805 print
810 print "Change which line? ";
815 gosub 9000:rem getkey subroutine
820 ln = val(t$):t$ = ""
825 print
830 if ln > k - 1 then print "Number too large!":goto 800
835 if line$(ln) = "*" then print "Line";ln;"is the *":goto 800
840 print "Line;ln;"now is: ";line$(ln)
845 print
850 print "Line";ln;"should be: ";
855 gosub 9000:rem getkey subroutine
860 line$(ln) = t$:t$ = ""
865 goto 700:rem correction routine
870 :
875 :
```

These lines are fairly standard correction routine lines. Line 755 prints a blank line before the question in line 760. Line 760 asks the necessary question and provides instructions for answering it.

Control is transferred to the Y/N (yes/no) subroutine beginning at instruction line 8000. In this subroutine, the user's response is stored in the string variable yes$. When the user has responded with some form of either a "y" or "n," control is returned to the instruction following the GOSUB instruction. Line 775 checks the answer, stored in yes$, to see if it equals y. If it does, the computer is instructed to jump to line 800 and proceed from there. If it does not equal y, the computer goes to the next instruction in line 780. Line 780 checks for the negative response of n. If the value stored in yes$ equals n, control is passed to the instruction in line 790. Because the computer will not reach this point until the value of yes$ is either a y or n, no further checking needs to be done. With a negative response, the computer is instructed to go to line 790. A negative response indicates that the typist believes all the information lines are correct, so the instruction in line 790 directs the computer to the file creation routine. A positive response to the question

in line 760 indicates that at least one of the information lines needs changing. Therefore, the computer is instructed to jump to the instruction in line 800, which asks the user to indicate which line needs changing.

Line 810 requests the number of the line that needs changing and stores that value in the variable "ln." Notice that there is no dollar sign following ln. This indicates that this variable is a numeric variable rather than a string variable. Numeric variables can only contain numbers.

Line 830 checks to see if the user has typed a number larger than the total number of lines displayed. If that is the case, a message is printed and the computer returns to line 800 to ask again for the number of the information line to be changed. Line 835 makes certain that the typist does not change the line with the *. Line 840 prints the line as it was originally entered, and line 850 waits for the user to type in the correct information. Finally, line 865 returns to line 700 to begin the correction process over again.

The correction process will be repeated until the user answers the question in line 760 in the negative, indicating that all information lines are correct. A number of other lines or checks could have been included, but for present needs these lines are sufficient.

THE FILE-CREATION ROUTINE

The third and last major part of the program is the file-creation routine. There is no screen display in connection with this routine, but the disk drive comes on for a brief time while the two files are created and information is written to them. Enter this routine by typing:

```
3000 rem **--file creation subroutine--**
3005 :
3200 rem *--data file--*
3205 dopen#2,"Mail Data",w
3210 for i = 1 to k
3215 print #2,line$(i)
3220 next i
3225 dclose#2
3230 :
3300 rem *--write pointer file--*
3305 dopen#1,"Mail Pointer",w
3310 print#1,k
3315 dclose#1
3320 :
3325 :
```

You are finally into the actual file-handling routine. As you can see, the routine is quite short. The key to filing system programs is often proper planning. If you have tried to anticipate and provide for all possible requirements, present and future, your data files can become very powerful and useful. If you are not careful in your planning, however, you might find that some of the information you thought you had in the file has been overwritten, or lost, or is practically unavailable. This is the reason for including the two single-character separators, and the reason for lines 3300-3325 in this routine.

Line 3305 opens a file called Mail Pointer in the write mode, so that this file can receive information. This file is identified as #1. This first data file will be used to keep track

of the number of information lines in the second data file. This first file provides a pointer value for the Mail Data file. That is the reason for naming this data file Mail Pointer.

Line 3310 *prints* the current value of k, which in the example, should be a six. This is done to keep track of the total number of information lines that will be kept in Mail Data, so that you know how many lines to read back into the computer with other programs. There are other ways to keep track of this number, but with sequential files this is one of the easiest and clearest.

Commodore provides a useful reserved word (ST, for STatus) that actually lets you know when the end of the file has been reached. You will be making use of this handy feature in some later programs. At this point, however, it is important to know the number of information lines and the process for obtaining that number.

Line 3205 opens the second data file, also in the write mode, and prepares it to receive information. Lines 3210-3220 are essentially the same loop as lines 740 to 750, but this time the information is printed to the diskette instead of just the screen. Here you do want to print the separator !, so the counter goes from one to the value of k. Finally, the file is closed in line 3225.

One additional comment needs to be made. Lines 3215 and 3310 print information to the diskette rather than to the screen because the word "print" is immediately followed by the "#" symbol (without an intervening space), a number, a comma, and the information that is to be written to the specific file. This is the method the computer uses to distinguish between information that is to be sent to the disk instead of the screen, and therefore, it is absolutely necessary. If a file name has not previously been identified with the number, an error will occur. If an incorrect number is specified, information can be sent to the wrong file or lost. It is very important to keep the numbers consistent with the file names.

THE SUBROUTINES

The only remaining lines that need be discussed are the two subroutines and a routine to end the program. You will accomplish this with the following lines. Type:

```
5000 rem **--end routine--**
5005 end
5010 :
5015 :
8000 rem **--y/n input subroutine--**
8005 print "Type a ";
8010 color chtr,white:print "Y";
8015 color chtr,yellow:print " or ";
8020 color chtr,white:print "N";
8025 color chtr,yellow:print " :";
8030 gosub 9000:rem getkey subroutine
8035 yes$ = t$:t$ = ""
8040 print
8045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
8050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
8055 print
8060 color chtr,white:print "Incorrect Choice!"
8065 color chtr,yellow
8070 print
8075 goto 8000:rem ask again
```

```
8080 :
8085 :
9000 rem **--getkey subroutine--**
9005 getkey a$
9010 if a$ = chr$(20) then 9040:rem delete char.
9015 if a$ = chr$(13) then return:rem return key
9020 print a$;
9025 t$ = t$ + a$
9030 goto 9000
9035 :
9040 lg = len(t$)
9045 if lg < 1 then 9000
9050 print a$;
9055 t$ = left$(t$,lg - 1)
9060 goto 9000
```

These three short routines (or two subroutines and one routine, if you prefer) actually reduce the amount of program instructions necessary in the entire program. The y/n input subroutine and the getkey subroutine are called by GOSUB statements in previous instructions. Control is transferred to the subroutines until certain conditions are met or certain instructions are followed. Once the task of the subroutine is finished, the RETURN statement transfers, or returns, control back to the instruction that directly follows the GOSUB instruction. The end routine must be placed between the file-creation routine and the two following subroutines, in order to stop the computer from again executing those routines. Entering such a subroutine without the GOSUB statement will cause a "return without gosub error" and cause the program to completely stop. In this situation, such an error would not hurt anything, because you are finished anyway, but it is not good programming to allow such an error to occur.

Lines 8005 to 8025 instruct the user what to type. These lines can be read and understood by someone with a small amount of previous instruction. They will also be understandable to the original programmer six months from now. They say: Print (or display on the screen) the words "Type a " in the previously set color for text (yellow). Immediately follow that with a white capital Y. Switch back to yellow for the word "or" with a space on each side. Still on the same line, use white to display a capital N followed by a final switch back to yellow, a space, and a colon.

The getkey subroutine is called from this subroutine, demonstrating the fact that subroutines can be called from within other subroutines. This process is referred to as *nesting* GOSUBs. The user's response is stored in the string variable t$ (or temporary variable) and transferred to the string variable yes$. Yes$ is then checked to see if it equals y or Y. If it does, the computer is instructed to store a lowercase y in yes$ and then "return" to the line following the GOSUB statement. If it does not equal y or Y, the computer goes to the next instruction in line 8050.

Line 8050 checks for the negative responses of n or N. If the value stored in yes$ matches one of them, the computer is instructed to store a lowercase n in yes$ and then "return" to the line following the GOSUB statement. If it does not match one of the negative responses, the computer ignores this instruction line and proceeds to the following instruction line. If the computer ever reaches these next lines, lines 8055-8075, we know that the individual has typed something other than a y, Y, n, or N. Anything but one of these responses, in this situation, can be viewed as an incorrect response. Therefore, the

instruction in line 8060 tells the user that he or she has typed something wrong. Line 8075 sends the computer back to line 8000 to inform the user to type some form of a y or n. Although, in this program, this subroutine is used only once, when the program is expanded, this subroutine will come in very handy.

The getkey subroutine does just what its name says. It gets one keystroke at a time. The BASIC 7.0 command of GETKEY waits until the program user has pressed some key on the keyboard. This command is very useful and powerful, but requires that the programmer know the values assigned the keys. Once again, the Commodore 128 System Guide comes in handy. Pages 357 and 358 contain the values assigned the different keys. The first key that must be checked is the delete key, or ASCII value 20. Suppose the user has typed in the letters "Davad" instead of "David," and then noticed the error. To correct the error, the INS/DEL key needs to be pressed twice, but the getkey command reads each keypress. Without checking for the delete or backspace key, the errors and backspacing would be stored. When the ASCII value of 20 is identified, control is transferred to the section of the getkey subroutine that eliminates the delete keypress.

ASCII value 13 is the RETURN key and signifies that the user has finished entering a line of information. When this value is identified, the computer is instructed to return to the calling routine.

Line 9020 displays on the screen the character just typed. Then the contents of a$, or the last keypress, is added to the previous keypresses stored in t$. The process is repeated until the user presses the RETURN key.

DEBUGGING THE PROGRAM

If you have been following along and typing in the program, you can edit and make corrections to the program by using the cursor keys on the right of the keyboard. Once a correction has been made, pressing the RETURN key saves the corrected version in the computer's memory. When all corrections have been made, save this program on diskette by giving it the name "mail.create."

```
dsave "mail.create {RETURN}
```

Now, type **directory {RETURN}** to see if the file name is listed. It should be listed like this:

```
0 "data file    1        ",87 2a
1      "hello"                 prg
1      "Write Seq.File"        prg
2      "Read Seq.File"         prg
1      "Seq.Data File"         seq
13     "mail.create"           prg
1310 blocks free.
```

At this point, you can run the program and enter your own name, address, and phone number if you wish. If you do not get the results given below, you will need to check over the program listing to see that what you have typed is entered exactly as given in this chapter. This checking process is often referred to as *debugging* a program (i.e., finding the mistakes that have somehow found their way into the program). Often the mistakes result from unfamiliarity with the particular keyboard. The more you use the keyboard,

the fewer mistakes there will be. Persistence and careful checking will eventually result in a program operating correctly. The exhilaration of finally getting a program to operate correctly is well worth the frustration all programmers go through in creating programs.

But I repeat—type carefully! This program has been thoroughly tested and checked for errors. As listed in the book, the program is fully operational. If you get an error message after you are finished typing in the program and have tried to run it, the most likely problem is with the typing, not the program logic or any misprint of the listings! I cannot emphasize enough the aspect of carefully typing and reading for mistyped instructions. An example is the typing of a 1 (the number one) instead of l (the lowercase letter l). For those who do not want to spend the time keying in all these instructions, a diskette containing all the programs is available. (Information on ordering the diskette is given at the back of the book.)

One last note: I have added asterisks and dashes, as a personal preference, to help set off certain remarks. In addition, I purposely left out line numbers between 875 and 3200 to indicate that the line numbers do not need to have any specific sequence or pattern, and because future programs will add routines within this range of lines.

My preference is to begin with line 100 for the program name. Furthermore, I like to have each part or section of the program begin on some round number, such as 100, 300, 500, or 1000, with each line number being a multiple of five. This numbering system is not always possible, and without a renumbering command or program, it is more trouble than it is worth. I have found, however, that it helps the readability. The main point here is not to introduce the numbering system I feel most comfortable with, but to stress the importance of having a numbering system that you consistently follow whenever possible. Careful attention to small details, like a consistent numbering system and program format, can make life for any programmer much easier. Such attention to detail can also help develop thinking patterns that aid in creativity and logical solutions to difficult problems.

PROGRAM RESULTS

When you type the word "run," you become the user of the program. As the user, you should see the screen clear, and two lines down the screen the word INSTRUCTIONS followed by three lines of information should appear. Type in a name and press the RETURN key. You are then told to type in line number two. If you want to type in a title for the name in the first line, you can. If no title is needed, then type in the street address and press RETURN. This process should continue until you type the word "end." When you do type some form of the word "end" you will be asked for the phone number and told that, if there is no phone number, you should just press the RETURN key.

After the phone number has been typed, you are shown a list of the lines you have typed, and asked if you wish to change any of those lines. If you want to change a line, you must answer the question with a Y or a y. If you do not want to change a line, type either an N or an n. If you need to make changes and have typed a Y or y, you will be asked which line number you want to change. Respond with one of the numbers on the screen. If you type a number larger than the total number of lines on the screen, you will be shown an error message and given another chance to respond correctly. If you do not want to change a line after indicating that you did, respond to the question asking for a line number with the number zero.

If you have answered with a valid line number, you will be shown the line as you typed it originally and asked for the correct information for this line. After typing in the

new line and pressing RETURN, you will be shown the list of lines again with the new line in place of the old line.

You can make as many changes as you wish. When you are satisfied and do not wish to make any more changes, a response of N or n to the question about changes will instruct the computer to write the information out to the diskette. The disk drive will come on for a few seconds, and the ready prompt will reappear. Now, if you type **directory**, in addition to the previously created files, you should see:

```
0 "data file    1       ",87 2a
1      "hello"                 prg
1      "Write Seq.File"        prg
2      "Read Seq.File"         prg
1      "Seq.Data File"         seq
13     "mail.create"           prg
1      "Mail Data"             seq
1      "Mail Pointer"          seq
1308 blocks free.
```

Two additional files have been created with this one program: Mail Pointer has been created to store the number of information lines now in Mail Data. In other words, the first file keeps track of the amount of information in the second file.

Several questions present themselves at this point. How do I add more names to this file? How do I actually see what is in the file? As you may have realized by now, there are a number of possible answers. One answer would be to add more lines to this program so that the program reads back what it just wrote to the diskette. Another answer is to write a separate program, and possibly a program menu, that would enable the user to switch easily between programs that write information and programs that read the information. In the next chapter, you will explore a number of these possibilities and see a little of what can be done with the information once it is safely and correctly on disk. Figure 4-1 shows the entire mail.create, as discussed in this chapter.

CHAPTER 4 QUESTIONS

1. True or False: Information in a sequential access file can be overwritten by additional information.
2. What BASIC reserved word is used to initiate a sequential data file?
3. Name the three main parts (or routines) in the "mail.create" program.
4. What does reserving multiple spaces for a variable create?
5. What symbol did we use to separate sets of information?
6. What does DIM stand for?
7. FOR I = 1 TO K is the first line in what kind of loop?
8. The program user's response is tested by what kind of BASIC statement?

Fig. 4-1. The mail.create program.

The mail.create Program

```
100 rem ***--mail.create--***
105 :
110 :
115 rem **--initialization--**
120 :black =  1
125 :white =  2
130 :::red = 11
135 yellow =  8
140 ::chtr =  5
145 bckgnd =  0
150 :
200 dim line$(20)
205 print chr$(14):rem upper/lower case mode
210 print chr$(8):rem disable char.set change
215 color bckgnd,black
220 :
225 :
300 rem **--keyboard input routine--**
305 k = 1:rem line counter
310 scnclr
315 char 0,22,2
320 color chtr,white
325 print "INSTRUCTIONS"
330 print
335 color chtr,yellow
340 print "Type the name and address as if you are addressing an envelope."
345 print
350 color chtr,red
355 print chr$(2);:rem underline
360 print "Do not use a comma or a colon!";
365 color chtr,yellow
370 print chr$(130);" ";:rem underline off
375 print "Press ";
380 color chtr,white:print "RETURN";
385 color chtr,yellow:print " after each line."
390 print
395 print "Type the word ";
400 color chtr,white
405 print "END";
410 color chtr,yellow
415 print " on a separate line when you are finished."
420 print:print
425 print "Type in line ";k;":";
430 gosub 9000:rem getkey subroutine
435 if t$ = "end" or t$ = "END" or t$ = "End" then 500
440 if t$ = "" then print:print "We need some information.":goto 425
445 line$(k) = t$:t$ = ""
450 print
455 k = k + 1
460 goto 425:rem next line
465 :
470 :
500 rem **--get phone number--**
505 line$(k) = "*":rem separator for phone number
510 t$ = ""
515 print:print
520 k = k + 1
525 print "Press ";
530 color chtr,white:print "RETURN";
535 color chtr,yellow:print " if there is no phone #."
540 print
545 print "Type in Phone #: ";
550 gosub 9000:rem getkey subroutine
555 line$(k) = t$:t$ = ""
560 if line$(k) = "" then line$(k) = "none"
```

```
  565 k = k + 1
  570 line$(k) = "!":rem separator between sets of information
  575 :
  580 :
  700 rem **--correction routine--**
  705 scnclr
  710 char 0,0,3
  715 print "Do not change the line with the ";
  720 color chtr,white:print "*";
  725 color chtr,yellow:print ". ";
  730 print "This symbol is used as a separator."
  735 print
  740 for i = 1 to k - 1
  745 print i;" ";line$(i)
  750 next i
  755 print
  760 print "Change any line? ";
  765 gosub 8000:rem y/n input subroutine
  770 print
  775 if yes$ = "y" then 800
  780 if yes$ = "n" then 790
  785 :
  790 goto 3000:rem file creation subroutine
  795 :
  800 rem *--change line--*
  805 print
  810 print "Change which line? ";
  815 gosub 9000:rem getkey subroutine
  820 ln = val(t$):t$ = ""
  825 print
  830 if ln > k - 1 then print "Number too large!":goto 800
  835 if line$(ln) = "*" then print "Line";ln;"is the *":goto 800
  840 print "Line";ln;"now is: ";line$(ln)
  845 print
  850 print "Line";ln;"should be: ";
  855 gosub 9000:rem getkey subroutine
  860 line$(ln) = t$:t$ = ""
  865 goto 700:rem correction routine
  870 :
  875 :
 3000 rem **--file creation subroutine--**
 3005 :
 3200 rem *--data file--*
 3205 dopen#2,"Mail Data",w
 3210 for i = 1 to k
 3215 print#2,line$(i)
 3220 next i
 3225 dclose#2
 3230 :
 3300 rem *--pointer file--*
 3305 dopen#1,"Mail Pointer",w
 3310 print#1,k
 3315 dclose#1
 3320 :
 3325 :
 5000 rem **--end routine--**
 5005 end
 5010 :
 5015 :
 8000 rem **--y/n input subroutine--**
 8005 print "Type a ";
 8010 color chtr,white:print "Y";
 8015 color chtr,yellow:print " or ";
 8020 color chtr,white:print "N";
 8025 color chtr,yellow:print " :";
 8030 gosub 9000:rem getkey subroutine
 8035 yes$ = t$:t$ = ""
 8040 print
 8045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
```

```
8050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
8055 print
8060 color chtr,white:print "Incorrect Choice!"
8065 color chtr,yellow
8070 print
8075 goto 8000:rem ask again
8080 :
8085 :
9000 rem **--getkey subroutine--**
9005 getkey a$
9010 if a$ = chr$(20) then 9040:rem delete char.
9015 if a$ = chr$(13) then return:rem return key
9020 print a$;
9025 t$ = t$ + a$
9030 goto 9000
9035 :
9040 lg = len(t$)
9045 if lg < 1 then 9000
9050 print a$;
9055 t$ = left$(t$,lg - 1)
9060 goto 9000
```

Chapter 5

Appending Sequential Files

Now the fun begins. You have created a file, but as you will soon see, the creation is one of the easiest parts of sequential file manipulation. There are two things you would immediately like to do with this file: read what is in the file and add to the file. Both tasks are relatively easy to do, but because the job of reading is simpler to explain and more rewarding, a short program to read the file will be discussed first.

Enter the following code either by typing it into the computer again, or by loading in the mail.create program and deleting lines 300 to 9060. If you decide to use the second method, the sequence would be:

```
dload "mail.create {RETURN}

delete 300-9060 {RETURN}

list {RETURN}
```

If you follow the above steps, you should see:

```
100 ***--mail.create--***
105 :
110 :
115 rem **--initialization--**
120 :black =  1
```

```
125 :white =  2
130 :::red = 11
135 yellow =  8
140 ::chtr =  5
145 bckgnd =  0
150 :
200 dim line$(20)
205 print chr$(14):rem upper/lower case mode
210 print chr$(8):rem disable char.set change
215 color bckgnd,black
220 :
225 :
```

Line 100 needs to be changed to reflect the fact that this will be a program that reads the file. Therefore, line 100 should read:

```
100 rem ***--mail.reader1--***
```

It is called "reader1" because this program is a very elementary way of displaying the file information. Eventually you will have a second program that reads and displays the information in the file in a number of different ways.

One other change should be made. Line 200 dimensions the string variable line$ for 20 lines of information. Very soon the file will contain many more than 20 lines of information, making this instruction obsolete. The exact number of lines is stored in the Mail Pointer file, and will be brought into the computer with the next set of instructions. Therefore, you do not need line 200 and line 200 should be deleted:

```
delete 200 {RETURN}
```

or just type the line number and press RETURN. Either method will eliminate line 200 from the listing.

Add the following instruction lines:

```
300 rem **--user message--**
305 scnclr:color chtr,red
310 char 0,32,5
315 print chr$(15);"ONE MOMENT PLEASE!";chr$(143)
320 :
325 :
400 rem **--read file routine--**
405 :
410 rem *--read pointer file--*
415 dopen#1,"Mail Pointer"
420 input#1,rec
425 dclose#1
430 dim line$(rec)
435 :
440 rem *--data file--*
445 dopen#2,"Mail Data"
450 for i = 1 to rec
```

```
455 input#2,line$(i)
460 next i
465 dclose#2
470 :
480 :
```

This section of programming code brings the contents of the data file into the computer's memory. The process of reading a sequential file, in this case, involves accessing the pointer file to find out how many lines of information the file contains, and accessing the data file to read in the designated number of lines of information.

The message in line 315 lets the user know that the computer is working while the contents of the data file are brought into the computer's memory. The message will stay on the screen as long as it takes to read the entire file. Therefore, at first, when the file does not contain many names, the message will not be on the screen very long. The more names the file contains, the longer it will take to read the file and the more the message in Line 315 is necessary.

Line 315 also contains two new chr$ values. Chr$(15) instructs the computer to go into the flash mode, in which text blinks off and on. In this case, the text to flash is the message "ONE MOMENT PLEASE!" Immediately after the text is identified, the computer is instructed to go out of the flash mode [chr$(143)] so that future screen displays are not flashing.

Line 415 tells the computer that you want to use the disk and that you want to open the Mail Pointer file as file number one in order to bring in the number of information lines stored in our main data file. Remember that file numbers can go from 1 to 127. Remember also that the number chosen is simply an arbitrary assignment, and does not indicate the actual number of files in use at any one time.

Line 420 brings in, from file number c ıe, a value for the number of lines written to the diskette in the file creation program. If you are not clear on this, check back to the explanation of lines 3305 to 3315 in the previous chapter. You are simply reading back the number written in those lines. Line 430 reserves space in the computer for the information you will be bringing in from the diskette. Because you are not sure of the exact number of lines, which will change every time you add information, you should use the variable "rec," which will always equal the number of lines, or *records*, that have been written.

Line 445 opens the main data file, called Mail Data, as the second file. This instruction line also opens the file in such a way (without the W, or write, parameter) that information can only be read from the file, not written to the file. Now you can bring in a copy of the information contained in the file. Lines 450 to 460 bring in that data. Line 450 establishes the boundaries for the loop needed to count from the first to the last line of information as represented by the variable "rec." Because the file has been identified by its number, the computer understands that the input statement in line 455 refers to the diskette and not the keyboard. This line tells the computer to go to the diskette and obtain a copy of the information contained in the information line specified by the variable i.

The operation is on the same principle as the logic of lines 3210 to 3220 in our file creation program. This time, however, you are bringing information into the computer from the diskette instead of transferring information from the computer to the diskette. Now the information physically exists in two locations: in the computer's memory, and on the diskette. By bringing the information into the computer, you have not erased that

information on the diskette. Merely reading a file does not disturb the contents of that file. Finally, line 465 closes the file.

You will become deeply involved in different ways of displaying your information a little bit later, but for now the following routine will get the job done:

```
500 rem **--display routine--**
505 scnclr:color chtr,yellow
510 for i = 1 to rec
515 print i;" ";line$(i)
520 next i
525 :
530 :
600 rem **--end routine--**
605 end
```

The read file routine can be used in a number of different programs to bring in all the information from the file, but the display routine above will not be functional in very many situations. You will alter this routine later to make it more usable (see Chapter 6).

Save the program now as "mail.reader1," then press the F3 key (or type DIRECTORY or CATALOG) to see the list of files now on the diskette. Type:

dsave "mail.reader1" {RETURN}
F3 [or DIRECTORY {RETURN} or CATALOG {RETURN}]

and you should see:

```
0 "data file    1       ",87 2a
1      "hello"              prg
1      "Write Seq.File"     prg
2      "Read Seq.File"      prg
1      "Seq.Data File"      seq
13     "mail.create"        prg
1      "Mail Data"          seq
1      "Mail Pointer"       seq
4      "mail.reader1"       prg
1304 blocks free.
```

After you have saved the mail.reader1 program, type:

```
run {RETURN}
```

The information you provided when you used the mail.create program should be displayed on the screen. If it is not, you need to check your programs to see that they are entered exactly as given in this book. The screen display should look similar to the following (with your information in place of this sample information):

1 David H. Miller
2 9999 Any St

3 Somewhere CA 90000
4 *
5 800-444-1111
6 !

ready.

Earlier you created a file, Mail Data, and wrote the first group of lines containing information to that file. Now you have read that information back and displayed it. Next you need to be able to add more information to the file. If you run the file creation program again and use a different set of information lines, what would happen? Would the new information be added to the file? Would the old information be replaced? Every time this program is run, the computer is instructed to open two new files, Mail Data and Mail Pointer, but both of these files already exist, so you will get a "file exists" error. Nothing will happen to the first set of information lines already on the diskette if you try to use this program to add a second set of information lines, but the information you just typed in will be lost.

Thus you need a third program to add more lines of information to the Mail Data file. (For those of you itching to put all these programs into one large program, have patience, I will eventually explain how these programs can be tied together without actually existing as one large program.) This third program is really just a modification of the file creation program. The modification needs to be done, or the results will be worthless. The modification is relatively simple if you closely follow the instructions given below.

MAIL.ADDER1

As far as line 875 of the mail.create program (see the complete listing at the end of Chapter 4), the new program can be the same, with one minor change. First load the mail.create program, and then list it to see a complete listing of the instructions in this program. Type:

```
dload "mail.create {RETURN}

list {RETURN}
```

Remember that pressing the F2 key displays the word DLOAD with the quotation mark, so that the file name is the only thing that needs to be typed. The F7 key produces the word LIST and then displays the listing of the program.

All of the program will not fit on the screen at one time. The first instructions disappear from view off the top of the screen. In BASIC, you can list to a certain line number, or from a certain line number to the end of the program, or from one line number to another, like this:

```
list -200 {RETURN}

list  350- {RETURN}

list 100-800 {RETURN}
```

Line 100 should be changed, so list line 100:

```
list 100 {RETURN}
```

Change line 100 to read:

```
100 rem ***--mail.adder1--***
```

Line 790 also needs changing. Change line 790 so that it reads:

```
790 goto 3000:rem file addition subroutine
```

Except for these changes, the mail.create program works fine as far as line 875 for the new mail.adder1 program, so be certain to make those changes before continuing.

The logic for the placement of the next two routines will become clear a little later. For now, add these lines:

```
2000 rem **--repeat routine--**
2080 scnclr
2085 char 0,0,3
2090 print "Do you want to add more information? ";
2095 gosub 8000:rem y/n input subroutine
2100 if yes$ = "y" then run
2105 if yes$ = "n" then 5000
2110 :
2115 :
```

Line 2000 names the routine. Line 2080 clears the screen and positions the cursor in the upper left-hand corner of the screen. (I have purposely not used the line numbers between 2000 and 2080 because additional code will eventually be included in this routine.) Line 2085 moves the cursor down three lines. Line 2090 prints the question about additional information, and line 2095 sends the computer to the yes/no subroutine to wait for the user to respond with either a y or an n. Line 2100 checks the response. If the response equals a y, then the computer is instructed to run the program again. If the response is negative, control is transferred to the instruction at 5000, the end routine. Line 5005 tells the computer to end this program.

Neither of these choices writes anything to the diskette. That is the reason you need the file addition routine. This file addition routine is very similar to the file creation routine, but the changes are essential. This new routine will do three things: check the Mail Pointer file to see how many information lines are currently in the Mail Data file, append (or add) the new lines to the Mail Data file, and write the new total number of lines back out to the Mail Pointer file.

Enter the following:

```
3000 rem **--file addition subroutine--**
3005 :
3100 rem *--read pointer file--*
```

```
3105 dopen#1,"Mail Pointer"
3110 input#1,rec
3115 dclose#1
3120 rec = rec + k
3125 :
```

This part of the routine should look somewhat familiar. It is almost exactly the same as the read pointer file routine in the mail.reader1 program. Line 3000 names the routine, and line 3100 names the first part of the routine. Line 3105 opens the pointer file as the first file. Because the W parameter is not included, the computer understands that you want to read the file, not write information to it. Line 3110 tells the computer you want to read the number one file (Mail Pointer) and input the same number as line 420 in the mail.reader1 program. You want to store its value in the variable "rec" (for record). You cannot use the variable "K," because you are already using it. Once again, this variable represents the number of lines of information already in the file.

Line 3115 closes the file. Line 3120 adds the number of lines already in the file to the number of new lines you have just typed into the computer, and stores that new total back in variable "rec." The logic for this line is the same as for the standard "k = k + 1" lines. If you are not clear about this logic, it is best to just accept that this is one way the computer totals things.

At this point, you are almost finished with the new file addition routine. Three lines need some changes and one more line needs to be added before you have an operational mail.adder1 program. List the following lines and make the indicated changes, typing carefully. You can either retype the entire line, or use the edit keys to make the necessary changes. The first change is to line 3205:

```
Line 3205 now is:       3205 dopen#2,"Mail Data",w
Line 3205 should be:    3205 append#2,"Mail Data"

Line 3305 now is:       3305 dopen#1,"Mail Pointer",w
Line 3305 should be:    3305 dopen#1,"@Mail Pointer",w

Line 3310 now is:       3310 print#1,k
Line 3310 should be:    3310 print#1,rec
```

After these changes, the full routine should look like the following:

```
3000 rem **--file addition subroutine--**
3005 :
3100 rem *--read pointer file--*
3105 dopen#1,"Mail Pointer"
3110 input#1,rec
3115 dclose#1
3120 rec = rec + k
3125 :
3200 rem *--data file--*
3205 append#2,"Mail Data"
3210 for i = 1 to k
3215 print#2,line$(i)
```

```
3220 next i
3225 dclose#2
3230 :
3300 rem *--write pointer file--*
3305 dopen#1,"@Mail Pointer",w
3310 print#1,rec
3315 dclose#1
3320 :
```

Line 3205 uses a new BASIC file command, APPEND, to do just what it says. It appends, or adds, to the file rather than overwriting any of the information already in the file. This line tells the computer to open the Mail Data file as the number two file, and prepare to add to it. From line 3210 to 3305, the routine is the same as the routine in the mail.create program. Line 3210 sets up the loop; line 3215 prints the information in line i to the diskette, placing it immediately after the information already on the diskette. Line 3220 goes back for another line of information, and line 3225 closes the file when all the new information lines have been added to the file.

The pointer file is then opened in the *replace* mode (with the REPLACE command—@), and the new total number of information lines is written back out to the diskette over the previous total. Line 3315 closes this pointer file again. One more line needs to be added to this routine. When the file handling operations have been completed, the computer must be directed back to the repeat subroutine:

```
3325 goto 2000:rem repeat subroutine
```

Line 3325 directs the computer to jump back to the instruction in line 2000, which is the start of the routine that asks if the user wants to add more information.

You have finished the program that will add more information to the Mail Data file. You should now save this new program to the diskette as mail.adder1:

```
dsave "mail.adder1 {RETURN}
```

You can check your typing by going over the complete listing of the program given at the end of this chapter. Please remember that it is very important to follow along by typing in the necessary lines on your Commodore 128.

Before continuing, contrast the logic in the mail.create program with the logic in the mail.adder1 program. In mail.create, the order of the program was in sequence: (1) enter information through the keyboard (keyboard input routine), (2) check and/or correct the entered information (correction routine), (3) store the entered information out to the diskette (file creation routine), and (4) end the program (end routine).

The logic (or sequence) changes somewhat with the mail.adder1 program. You still begin with: (1) entering information through the keyboard (keyboard input routine), (2) checking and/or correcting the entered information (correction routine), and (3) storing the added information out to the diskette (file addition routine). But instead of ending the program, the computer is directed to the fourth step asking if the user wants to add more information (repeat routine). If the response is positive, the fifth step is to begin the program over again by instructing the computer to run the program. This cycle continues until the user responds in the negative. When the response is negative, the sixth step is to

direct the computer to the routine beginning at line 5000, which ends the program (end routine). In both programs, the flow of logic has directed the computer to store the information on diskette before any additional information can be entered.

You now have three complete programs: mail.create, mail.reader1, and mail.adder1. The combination of these programs will create a file, add information to the file, and read information back from that file. The three programs adequately demonstrate the procedures used to accomplish these tasks, but the programs are not really very useful or practical as they now exist. For instance, every time you run the mail.reader1 program, you will read the entire file and display the entire file. This happens even if you want only one name and address. After just a few names and addresses are added to the file, the list will begin to disappear off the top of the screen during display. More modification needs to be done in order to make these programs useful. If you are a good programmer in BASIC, you probably have some ideas about *features*, or additional options, you would like to see in one or more of these programs. If you have a little experience in programming, you will soon become much more experienced.

A few features will be added to these programs, with a full explanation of each additional step. If you would like to include these options and become more experienced at programming, especially with file information data, closely follow the different programming lines and explanations given. If you don't need these features, or if you want to create your own, you might want to skip ahead to Chapter 8 on advanced sequential data file manipulation. If you have had enough of sequential data files, you might want to jump immediately to Chapter 9 on relative access data files. You will be using some of these same routines in the chapters on relative access, but I will not go into the same explanatory detail then as in these present chapters.

MAIL.ADDER2

You will begin by modifying the mail.adder1 program. If you have used this program to enter a number of names and addresses, you will have noticed that the disk operates every time you have accepted a set of information lines as correct. This disk operation might not bother you if you are in no hurry to enter a large number of names and addresses, but there is no reason why the disk needs to operate after every name. Why not write the information out to the diskette only after you have finished entering all of the information lines? Such a change is clearly a matter of preference that proponents and opponents often argue about. In this situation, I prefer to enter all of my information for the current session before writing any of it to the diskette. In addition, I might want to print a mailing label of the information I have just entered before typing in a second set of information lines. This is obviously a preference feature, and it will do you no good if you do not have a printer. However, even if you do not have a printer, the routine might still be of interest because you will be formatting your display in a new way.

The first addition will consist of adding lines of computer instructions to allow the user to print out, in a mailing label, the information just entered. The second addition includes the computer instructions necessary so that the information will be written to the diskette after all the information for the current session has been entered and corrected. The additional computer instructions necessary to include both of these features are relatively few.

The Print Label Routine

Begin with the print label subroutine. Add the following lines to the mail.adder1 pro-

gram. (Remember that if the mail.adder1 program is not already in memory, you must load it into the computer's memory from the diskette.)

```
1000 rem **--print label subroutine--**
1005 scnclr
1010 char 0,0,3
1015 print "Would you like to print a mailing label
     now? ";
1020 gosub 8000:rem y/n input subroutine
1025 if yes$ = "y" then 1100
1030 if yes$ = "n" then 2000:rem repeat subroutine
1035 :
```

Line 1000 gives the title of the routine. Line 1005 clears the screen, and line 1010 places the cursor three lines from the top of the screen. Line 1015 prints the question, while line 1020 transfers control to the subroutine that waits for an answer. When either a y or an n is typed, control is returned to this routine and the response is checked. If the response is positive, which indicates that a printed label is desired, the computer is instructed to jump over the next two lines and go on to the rest of the routine, which begins at line 1100. If the response is negative, indicating that the user does not want a printed label, control is transferred to the repeat routine, which begins at line 2000.

```
1100  rem *--mailing label is desired--*
1105 scnclr
1110 char 0,0,3
1115 print "Please make sure the printer is on and ready
     to use.
1120 print
1125 print "Are you ready to begin printing? ";
1130 gosub 8000:rem y/n input subroutine
1135 if yes$ = "n" then 1000:rem ask again
1140 :
```

Lines 1100-1140 provide additional information to the user and check to see that everything is ready to print. If not, or if the user changes his or her mind, control is returned to the original question.

```
1145 rem *--printer channel--*
1150 open 4,4,7
1155 :
1160 rem first 4 = file #
1165 rem second 4 = printer device #
1170 rem 7 = command for upper/lower case
1175 :
1180 for i = 1 to k
1185 if line$(i) = "*" then i = i + 1:goto 1200
1190 if line$(i) = "!" then 1200
1195 print#4,line#(i)
1200 next i
```

```
1205 print#4," "
1210 close 4
1215 :
1220 :
```

Line 1150 opens the printer channel using the standard printer device number of 4 and the command of 7 (indicating the desire to use both uppercase and lowercase letters in the printout). The instruction in line 1180 is the now-familiar beginning of the loop.

Lines 1185 and 1190 are different from anything you have had so far. Line 1185 checks the contents of each line$( ) string for the * symbol. If it locates that symbol, it instructs the computer to add one to the value of the variable i, and then to proceed to the instruction at line 1200. The reason for this is simple, but hard to explain. When the computer comes to an asterisk, you do not want that asterisk printed, nor do you want the telephone number printed in a mailing label. So, you skip printing the asterisk and the telephone number by adding one to the counter (i) and jumping to the end of the loop, line number 1200. Line number 1200 increases the counter by one more line, so you have skipped two lines in the file: the lines that contained the separate symbol, *, and the telephone number. I think this will become clear, if it is not already so, when you type in and try the routine.

Line 1190 does much the same thing. It tells the computer to jump over the print statement and go to the instruction that increases the counter. It has the effect, then, of skipping over the ! separator symbol, and not printing it either. Line 1195 does the printing. It prints the contents of every string in the line$( ) array that is not either an asterisk, exclamation point, or telephone number, unless the telephone number has been typed in before the * symbol.

The information is sent to the printer instead of the screen because you have previously opened output to the printer as device number 4, and used the # symbol with a file number in the print statement. In the next chapter, you will see how to send information to either the printer or the screen with just one line of code, but for this routine, the PRINT#4 statement is sufficient.

Line 1200 increases the counter. When the loop is finished—that is, when all lines of information have either been printed or skipped, and the i counter has reached the value of k—the computer can go on to the next instruction outside the loop. Line 1205 is used to print a blank line, and instruction 1210 closes file number four. It is especially important to close the channel that is used for the printer, because strange things occur with an open channel to device 4. You are now finished with the print label routine, and control can pass to the repeat routine.

The Repeat Routine

The new repeat routine is even shorter and easier than the print label routine. Add the following to line 200:

```
, tlines$(100)
```

Line 200 now reads:

```
200 dim line$(20), tlines$(100)
```

Add the following lines:

```
2005 :
2010 rem *--add new lines to existing file lines--*
2015 for i = 1 to k
2020 tlines$(tk + i) = line$(i):rem tk + i, not tk + 1
2025 next i
2030 tk = tk + k
2035 :
```

Change lines 2100 and 2105 to read:

```
2100 if yes$ = "y" then 300:rem begin again

2105 if yes$ = "n" then 3000:rem file addition routine
```

Change the following lines:

```
Line 3120 now is:       3120 rec = rec + k
Line 3120 should be:    3120 rec = rec + tk

Line 3210 now is:       3210 for i = 1 to k
Line 3210 should be:    3210 for i = t to tk

Line 3215 now is:       3215 print#2,line$(i)
Line 3215 should be:    3215 print#2,tlines$(i)

Line 3325 now is:       3325 goto 2000:rem repeat routine
Line 3325 should be:    3325 :
```

They look like such small changes, but when viewed by the computer, the changed and added instructions make quite a difference in the way the program works. The program has now become more practical. You have added another string variable, and therefore, you must tell the computer to reserve memory for the contents of this new variable. That is what the added code does in line 200. You have DIMensioned the string variable tlines$ so that you can now add 100 lines of information before you need to write the information to the diskette. You can make this number smaller or larger; the number 100 is a completely arbitrary choice. You must be careful about the choice of variable names, though. Variables that contain (or are) reserved words cannot be used.

Next, you have added six lines of code: lines 2005-2030. Line 2010 is the start of a loop. Line 2020 is the real reason for the loop; it is the instruction that enables you to continue entering information without writing each set of information to the diskette separately. Again, the logic is fairly easy. You are going to keep the contents of the string variable line$ in the string variable tlines$ also. Then, because the information is stored in two locations in the computer, you can use the line$ string variable over again. In other words, you have moved the information from one memory location to another memory location—from line$(1) to tlines$(1), from line$(2) to tlines$(2), and so on.

The instruction at line 2030 helps keep track of all the lines that are typed in. TK, which stands for total k, is a cumulative total of all the lines of information typed in during one session. For the first set of information lines, the value of tk is zero, because you have not previously given tk any value. After the first set of information lines, tk becomes the value of k—the number of lines of information in the first set. In the loop, you have moved the contents of the second line$(1) to tlines$(tk + 1), line$(2) to tlines$(tk + 2), and so on.

This process can continue until you have accumulated 100 lines of information (or more if you have dimensioned tlines$( ) to more than 100). If you are adding more lines of information, line 2100 must direct the computer to begin again by transferring control back to the keyboard input routine, resetting the counter, k, to 1, and proceeding with the rest of the routine. Finally, line 3215 is changed to write the information contained in the string variable tlines$, instead of the contents of line$. This last change is a very important one, and, if it is not made, information will be lost.

Let's review the program logic in mail.create. The order of the program was, in sequence: (1) enter information through the keyboard (keyboard input routine), (2) check and/or correct the entered information (correction routine), (3) store the entered information out to the diskette (file creation routine), and (4) end the program (end routine). The logic (or sequence) changed somewhat with the mail.adder1 program: (1) enter information through the keyboard (keyboard input routine), (2) check and/or correct the entered information (correction routine), and (3) store the added information out to the diskette (file addition routine). But, instead of ending the program, the computer was directed to: (4) ask if the user wanted to add more information (repeat routine). If the response was positive the fifth step was to begin the program over again by instructing the computer to run the program. This cycle continued until the user responded in the negative. When the response was negative, the sixth step was to direct the computer to the routine beginning at line 5000 which ended the program (end routine). In both of these programs, the flow of logic directed the computer to store the information on diskette before any additional information could be entered.

In the mail.adder2 program, the logic is different. The computer is directed to accept and temporarily retain all the entered information *before* storing any information on diskette. In other words, the logic in mail.adder2 is (1) enter information through the keyboard (keyboard input routine), (2) check and/or correct the entered information (correction routine), and (3) ask if the user wants to add more information (repeat routine). If the response is positive, the fourth step is to begin the keyboard input routine over again by instructing the computer to go to line number 300. This cycle continues until the user responds in the negative. When the response is negative, the fifth step is to store the entered information out to the diskette (file addition routine), and finally (6) end the program (end routine).

It might not always be possible to have every program proceed in order through the program instructions, but it is a desirable goal. Proceeding in order makes the flow of logic much easier to understand. Proceeding in order is called the *top-down* approach to programming. In the mail.adder2 program, the logic flows from the keyboard input routine, to the correction routine, to the repeat routine, and back again, until the user indicates that he or she is finished entering information. Then, the logic drops from the repeat routine to the file addition routine and, finally, to the end routine.

In the next chapter, some features will be added to the display program and all of the programs will be combined so that they can operate together. Figure 5-1 shows the

programs discussed in this chapter in their entirety. When you have made the necessary changes, save this new version as mail.adder2

```
dsave "mail.adder2 {RETURN}
```

Then type:

```
directory {RETURN}
```

and you should see:

```
0 "data file    1       ",87 2a
1      "hello"                prg
1      "Write Seq.File"       prg
2      "Read Seq.File"        prg
1      "Seq.Data File"        seq
13     "mail.create"          prg
1      "Mail Data"            seq
1      "Mail Pointer"         seq
4      "mail.reader1"         prg
15     "mail.adder1"          prg
18     "mail.adder2"          prg
1271 blocks free.
```

CHAPTER 5 QUESTIONS

1. True or False: Running the mail.create program a second time with new information does no harm to the first information stored in the Mail Data file.
2. Give the name of the BASIC command that erases program lines from the computer's memory.
3. To what does line 3215 in the mail.adder1 program print the information?
4. What is the BASIC word used to tell the computer to jump to a certain line number?
5. Give the purpose of the following function keys: F2, F3, F5, F6, and F7.
6. What BASIC command allows you to add information to an existing sequential data file?

Fig. 5-1. The Mailing Reader programs.

The mail.reader Program

```
100 rem ***--mail.reader1--***
105 :
110 :
115 rem **--initialization--**
120 :black =  1
125 :white =  2
130 :::red = 11
135 yellow =  8
140 ::chtr =  5
145 bckgnd =  0
150 :
205 print chr$(14):rem upper/lower case mode
210 print chr$(8):rem disable char.set change
215 color bckgnd,black
220 :
225 :
300 rem **--user message--**
305 scnclr:color chr,red
310 char 0,32,5
315 print chr$(15);"ONE MOMENT PLEASE!";chr$(143)
320 :
```

```
325 :
400 rem **--read file routine--**
405 :
410 rem *--read pointer file--*
415 dopen#1,"Mail Pointer"
420 input#1,rec
425 dclose#1
430 dim line$(rec)
435 :
440 rem *--data file--*
445 dopen#2,"Mail Data"
450 for i = 1 to rec
455 input#2,line$(i)
460 next i
465 dclose#2
470 :
475 :
500 rem **--display routine--**
505 scnclr:color chr,yellow
510 for i = 1 to rec
515 print i;" ";line$(i)
520 next i
525 :
530 :
600 rem **--end routine--**
605 end
```

The mail.adder1 Program

```
100 rem ***--mail.adder1--***
105 :
110 :
115 rem **--initialization--**
120 :black =  1
125 :white =  2
130 :::red = 11
135 yellow =  8
140 ::chtr =  5
145 bckgnd =  0
150 :
200 dim line$(20)
205 print chr$(14):rem upper/lower case mode
210 print chr$(8):rem disable char.set change
215 color bckgnd,black
220 :
225 :
300 rem **--keyboard input routine--**
305 k = 1:rem line counter
310 scnclr
315 char 0,22,2
320 color chtr,white
325 print "INSTRUCTIONS"
330 print
335 color chtr,yellow
340 print "Type the name and address as if you are addressing an envelope."
345 print
350 color chtr,red
355 print chr$(2);:rem underline
360 print "Do not use a comma or a colon!";
365 color chtr,yellow
370 print chr$(130);" ";:rem underline off
375 print "Press ";
380 color chtr,white:print "RETURN";
385 color chtr,yellow:print " after each line."
390 print
395 print "Type the word ";
400 color chtr,white
405 print "END";
410 color chtr,yellow
415 print " on a separate line when you are finished."
```

```
  420 print:print
  425 print "Type in line ";k;":";
  430 gosub 9000:rem getkey subroutine
  435 if t$ = "end" or t$ = "END" or t$ = "End" then 500
  440 if t$ = "" then print:print "We need some information.":goto 425
  445 line$(k) = t$:t$ = ""
  450 print
  455 k = k + 1
  460 goto 425:rem next line
  465 :
  470 :
  500 rem **--get phone number--**
  505 line$(k) = "*":rem separator for phone number
  510 t$ = ""
  515 print:print
  520 k = k + 1
  525 print "Press ";
  530 color chtr,white:print "RETURN";
  535 color chtr,yellow:print " if there is no phone #."
  540 print
  545 print "Type in Phone #: ";
  550 gosub 9000:rem getkey subroutine
  555 line$(k) = t$:t$ = ""
  560 if line$(k) = "" then line$(k) = "none"
  565 k = k + 1
  570 line$(k) = "!":rem separator between sets of information
  575 :
  580 :
  700 rem **--correction routine--**
  705 scnclr
  710 char 0,0,3
  715 print "Do not change the line with the ";
  720 color chtr,white:print "*";
  725 color chtr,yellow:print ". ";
  730 print "This symbol is used as a separator."
  735 print
  740 for i = 1 to k - 1
  745 print i;" ";line$(i)
  750 next i
  755 print
  760 print "Change any line? ";
  765 gosub 8000:rem y/n input subroutine
  770 print
  775 if yes$ = "y" then 800
  780 if yes$ = "n" then 790
  785 :
  790 goto 3000:rem file addition subroutine
  795 :
  800 rem *--change line--*
  805 print
  810 print "Change which line? ";
  815 gosub 9000:rem getkey subroutine
  820 ln = val(t$):t$ = ""
  825 print
  830 if ln > k - 1 then print "Number too large!":goto 800
  835 if line$(ln) = "*" then print "Line";ln;"is the *":goto 800
  840 print "Line";ln;"now is: ";line$(ln)
  845 print
  850 print "Line";ln;"should be: ";
  855 gosub 9000:rem getkey subroutine
  860 line$(ln) = t$:t$ = ""
  865 goto 700:rem correction routine
  870 :
  875 :
 2000 rem **--repeat routine--**
 2080 scnclr
 2085 char 0,0,3
 2090 print "Do you want to add more information?";
 2095 gosub 8000:rem y/n input routine
```

```
2100 if yes$ = "y" then run
2105 if yes$ = "n" then 5000
2110 :
2115 :
3000 rem **--file addition subroutine--**
3005 :
3100 rem *--read pointer file--*
3105 dopen#1,"Mail Pointer"
3110 input#1,rec
3115 dclose#1
3120 rec = rec + k
3125 :
3200 rem *--data file--*
3205 append#2,"Mail Data"
3210 for i = 1 to k
3215 print#2,line$(i)
3220 next i
3225 dclose#2
3230 :
3300 rem *--write pointer file--*
3305 dopen#1,"@Mail Pointer",w
3310 print#1,rec
3315 dclose#1
3320 :
3325 goto 2000:rem repeat subroutine
3330 :
3335 :
5000 rem **--end routine--**
5005 end
5010 :
5015 :
8000 rem **--y/n input subroutine--**
8005 print "Type a ";
8010 color chtr,white:print "Y";
8015 color chtr,yellow:print " or ";
8020 color chtr,white:print "N";
8025 color chtr,yellow:print " :";
8030 gosub 9000:rem getkey subroutine
8035 yes$ = t$:t$ = ""
8040 print
8045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
8050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
8055 print
8060 color chtr,white:print "Incorrect Choice!"
8065 color chtr,yellow
8070 print
8075 goto 8000:rem ask again
8080 :
8085 :
9000 rem **--getkey subroutine--**
9005 getkey a$
9010 if a$ = chr$(20) then 9040:rem delete char.
9015 if a$ = chr$(13) then return:rem return key
9020 print a$;
9025 t$ = t$ + a$
9030 goto 9000
9035 :
9040 lg = len(t$)
9045 if lg < 1 then 9000
9050 print a$;
9055 t$ = left$(t$,lg - 1)
9060 goto 9000
```

The mail.adder2 Program

```
100 rem ***--mail.adder2--***
105 :
110 :
115 rem **--initialization--**
```

```
120 :black =  1
125 :white =  2
130 :::red = 11
135 yellow =  8
140 ::chtr =  5
145 bckgnd =  0
150 :
200 dim line$(20),tlines$(100)
205 print chr$(14):rem upper/lower case mode
210 print chr$(8):rem disable char.set change
215 color bckgnd,black
220 :
225 :
300 rem **--keyboard input routine--**
305 k = 1:rem line counter
310 scnclr
315 char 0,22,2
320 color chtr,white
325 print "INSTRUCTIONS"
330 print
335 color chtr,yellow
340 print "Type the name and address as if you are addressing an envelope."
345 print
350 color chtr,red
355 print chr$(2);:rem underline
360 print "Do not use a comma or a colon!";
365 color chtr,yellow
370 print chr$(130);" ";:rem underline off
375 print "Press ";
380 color chtr,white:print "RETURN";
385 color chtr,yellow:print " after each line."
390 print
395 print "Type the word ";
400 color chtr,white
405 print "END";
410 color chtr,yellow
415 print " on a separate line when you are finished."
420 print:print
425 print "Type in line ";k;":";
430 gosub 9000:rem getkey subroutine
435 if t$ = "end" or t$ = "END" or t$ = "End" then 500
440 if t$ = "" then print:print "We need some information.":goto 425
445 line$(k) = t$:t$ = ""
450 print
455 k = k + 1
460 goto 425:rem next line
465 :
470 :
500 rem **--get phone number--**
505 line$(k) = "*":rem separator for phone number
510 t$ = ""
515 print:print
520 k = k + 1
525 print "Press ";
530 color chtr,white:print "RETURN";
535 color chtr,yellow:print " if there is no phone #."
540 print
545 print "Type in Phone #: ";
550 gosub 9000:rem getkey subroutine
555 line$(k) = t$:t$ = ""
560 if line$(k) = "" then line$(k) = "none"
565 k = k + 1
570 line$(k) = "!":rem separator between sets of information
575 :
580 :
700 rem **--correction routine--**
705 scnclr
710 char 0,0,3
715 print "Do not change the line with the ";
```

```
  720 color chtr,white:print "*";
  725 color chtr,yellow:print ". ";
  730 print "This symbol is used as a separator."
  735 print
  740 for i = 1 to k - 1
  745 print i;" ";line$(i)
  750 next i
  755 print
  760 print "Change any line? ";
  765 gosub 8000:rem y/n input subroutine
  770 print
  775 if yes$ = "y" then 800
  780 if yes$ = "n" then 790
  785 :
  790 goto 1000:rem print label subroutine
  795 :
  800 rem *--change line--*
  805 print
  810 print "Change which line? ";
  815 gosub 9000:rem getkey subroutine
  820 ln = val(t$):t$ = ""
  825 print
  830 if ln > k - 1 then print "Number too large!":goto 800
  835 if line$(ln) = "*" then print "Line";ln;"is the *":goto 800
  840 print "Line";ln;"now is: ";line$(ln)
  845 print
  850 print "Line";ln;"should be: ";
  855 gosub 9000:rem getkey subroutine
  860 line$(ln) = t$:t$ = ""
  865 goto 700:rem correction routine
  870 :
  875 :
 1000 rem **--print label subroutine--**
 1005 scnclr
 1010 char 0,0,3
 1015 print "Would you like to print a mailing label now? ";
 1020 gosub 8000:rem y/n input subroutine
 1025 if yes$ = "y" then 1100
 1030 if yes$ = "n" then 2000:rem repeat subroutine
 1035 :
 1100 rem *--mailing label is desired--*
 1105 scnclr
 1110 char 0,0,3
 1115 print "Please make sure the printer is on and ready to use."
 1120 print
 1125 print "Are you ready to begin printing? ";
 1130 gosub 8000:rem y/n input subroutine
 1135 if yes$ = "n" then 1000:rem ask again
 1140 :
 1145 rem *--printer channel--*
 1150 open 4,4,7
 1155 :
 1160 rem first 4 = file #
 1165 rem second 4 = printer device #
 1170 rem 7 = command for upper/lower case
 1175 :
 1180 for i = 1 to k
 1185 if line$(i) = "*" then i = i + 1:goto 1200
 1190 if line$(i) = "!" then 1200
 1195 print#4,line$(i)
 1200 next i
 1205 print#4," "
 1210 close 4
 1215 :
 1220 :
 2000 rem **--repeat routine--**
 2005 :
 2010 rem *--add new lines to existing file lines--*
 2015 for i = 1 to k
```

```
2020 tlines$(tk + i) = line$(i):rem tk + i, not tk + 1
2025 next i
2030 tk = tk + k
2035 :
2080 scnclr
2085 char 0,0,3
2090 print "Do you want to add more information? ";
2095 gosub 8000:rem y/n input routine
2100 if yes$ = "y" then 300:rem begin again
2105 if yes$ = "n" then 3000:rem file addition subroutine
2110 :
2115 :
3000 rem **--file addition subroutine--**
3005 :
3100 rem *--read pointer file--*
3105 dopen#1,"Mail Pointer"
3110 input#1,rec
3115 dclose#1
3120 rec = rec + tk
3125 :
3200 rem *--data file--*
3205 append#2,"Mail Data"
3210 for i = 1 to tk
3215 print#2,tline$(i)
3220 next i
3225 dclose#2
3230 :
3300 rem *--write pointer file--*
3305 dopen#1,"@Mail Pointer",w
3310 print#1,rec
3315 dclose#1
3320 :
3325 :
5000 rem **--end routine--**
5005 end
5010 :
5015 :
8000 rem **--y/n input subroutine--**
8005 print "Type a ";
8010 color chtr,white:print "Y";
8015 color chtr,yellow:print " or ";
8020 color chtr,white:print "N";
8025 color chtr,yellow:print " :";
8030 gosub 9000:rem getkey subroutine
8035 yes$ = t$:t$ = ""
8040 print
8045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
8050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
8055 print
8060 color chtr,white:print "Incorrect Choice!"
8065 color chtr,yellow
8070 print
8075 goto 8000:rem ask again
8080 :
8085 :
9000 rem **--getkey subroutine--**
9005 getkey a$
9010 if a$ = chr$(20) then 9040:rem delete char.
9015 if a$ = chr$(13) then return:rem return key
9020 print a$;
9025 t$ = t$ + a$
9030 goto 9000
9035 :
9040 lg = len(t$)
9045 if lg < 1 then 9000
9050 print a$;
9055 t$ = left$(t$,lg - 1)
9060 goto 9000
```

Chapter 6

Displaying Sequential Files

In this chapter, you will begin to put together a *system* of programs and improve the display program. When you want to use the mail.reader1 program, you must type **run "mail.reader1**. When you are ready to add to the file, you need to type: **run "mail.adder2** (or **1** depending upon your preference). For occasional use, that amount of typing is not a problem, but if you are going to use the program quite often, the necessity of typing run and the file name can become bothersome. Besides, the computer can help eliminate the need to type that, so why not let it do so? What you need is another program. When it is properly set up, you will need to do the typing only once, and then you will be able to switch back and forth between programs by typing little more than a number. You will then have a *system* of programs that work together and are controlled by one *master* program.

Let's see how this can work. Make sure any program currently in memory is saved on diskette and then type:

```
new {RETURN}
```

The menu program that follows is a "stripped down" version of the complete mail.menu program listed at the end of this chapter. By now you should understand that all programs have code that is not absolutely necessary to the operation of the program, but does make the program easier or more pleasing, to use. An example of programming code that is not absolutely necessary is an "incorrect-choice routine." If every user always selected valid choices, there would not be any need for an incorrect-choice routine, but, because some user sometime will make a mistake, an incorrect-choice routine should be included

in the program. Such a routine, however, is not necessary to the actual operation of the program. Therefore, with the mail.menu program, I am going to present only the code that is absolutely necessary to the operation of the program, and then list the complete program at the end of the chapter.

Enter the following code as the mail.menu program:

```
500 rem **--mail menu--**
505 tb = 20:rem tab value--20 spaces to the right
510 scnclr
520 chr 0,24,2,"MAIL PROGRAM MENU":print
530 print:print tab(tb) "1.  FILE CREATION PROGRAM"
535 print:print tab(tb) "2.  FILE ADDITION PROGRAM"
540 print:print tab(tb) "3.  FILE DISPLAY PROGRAM"
545 print:print tab(tb) "4.  FILE CORRECTION PROGRAM"
550 print:print tab(tb) "5.  LIST OF FILES"
555 print:print tab(tb) "6.  END"
560 print:print tab(tb)
565 input "Which Program Number";number$
570 number = val(number$)
575 :
580 if number = 1 then run"mail.create
585 if number = 2 then run"mail.adder2
590 if number = 3 then run"mail.reader1
595 if number = 4 then run"mail.correction
600 if number = 5 then directory:goto 500
605 if number = 6 then end
```

As soon as you have finished typing this program into the computer, save it to the diskette. *DO NOT* run the program without saving it first! Type:

```
dsave "mail.menu {RETURN}
```

The directory should now show:

```
0 "data file    1       ",87 2a
1      "hello"                prg
1      "Write Seq.File"       prg
2      "Read Seq.File"        prg
1      "Seq.Data File"        seq
13     "mail.create"          prg
1      "Mail Data"            seq
1      "Mail Pointer"         seq
4      "mail.reader1"         prg
15     "mail.adder1"          prg
18     "mail.adder2"          prg
3      "mail.menu"            prg
1268 blocks free.
```

Note that if you choose to type in the longer version of the mail.menu program listed

at the end of this chapter, your directory will be different from the list provided above and all future directory illustrations.

The logic involved in this program instructs the computer to go to the diskette, load the chosen program, and then run it. Control is transferred to the new program, and the computer receives and follows the instructions contained in that program. Such transfer of control erases the mail.menu program from the computer's memory, replacing those instructions with the instructions in the selected program. That is why it is very important to save this mail.menu program to diskette before running it. If you run this program and make a selection of choices 1, 2, or 3 before you have saved it, all of your typing will be lost or replaced by the instructions associated with the program you selected. A new choice has been included for a program that you do not yet have—the file correction program. Because there is no program, the choice of 4 will not erase the mail.menu program from the computer's memory. Choices 5 and 6 also do not load in new instructions, and therefore will not erase the mail.menu program either.

Once you have saved the mail.menu program, it is safe to run it. Now, all that is needed to run any of your programs is **run "mail.menu**. Then, choose a number and let the computer do the rest.

I have used a new BASIC statement—TAB. This tells the computer to horizontally move over a certain number of spaces (the number stored in the numeric variable tb) and then display the contents of the next print statement. A variable has been used so that you have to change only one instruction line (line 505) if you don't like the positioning on the screen.

Line 570 uses the first of several instructions available to manipulate string variable information. Line 565 stores the user's response in the string variable number$. Line 570 then takes the numeric value (val) of the string(number$). Lines 580 through 605 check this value, now stored in two different memory locations, number and number$, and direct the computer to run the appropriate program.

You still, however, do not have a system of programs. What will happen, for example, when you are finished adding information and the information has been written out to the diskette? Will the program return you to this new mail.menu program? If you are not sure, check any of the previous programs to see if they contain instructions to load and run this mail.menu program. Instead, each program contains an instruction line that ends the operation of the program. That is the reason you must include instructions to run mail.menu in each of your previous programs. You can add the necessary lines to each program by loading the program into the computer's memory, adding the appropriate lines, and then saving each program back to the diskette under its same name. Type in the following lines:

```
5000 rem **--return to program menu--**
5005 scnclr
5010 char 0,15,9
5015 color chtr,white
5020 print "LOADING THE MAIL MENU PROGRAM"
5025 color chtr,yellow
5030 run "mail.menu
5035 :
5040 :
```

When you have finished typing, save this routine as menu.routine:

```
dsave "menu.routine {RETURN}
```

Then load in the mail.create program:

```
dload "mail.create {RETURN}
```

Then, immediately while the "return to program menu" instructions are still on the screen, use the cursor up key to go to the 5 in line 5000. If you press the RETURN key on each of the line numbers between 5000 and 5040, those lines will be added to the mail.create program just as if you had typed them. When you have finished, remember to save the corrected version. Type:

```
dsave "@mail.create {RETURN}
```

Follow the same procedure for mail.adder2. Type:

```
dload "menu.routine {RETURN}
```

Then list the routine so that the instructions are on the screen:

```
list {RETURN}
```

Load the mail.adder2 program into the computer:

```
dload "mail.adder2 {RETURN}
```

Then, immediately while the "return to program menu" instructions are still on the screen, use the cursor up key to go to the 5 in line 5000. Press the RETURN key on each of the line numbers between 5000 and 5040. These lines will be added to the mail.adder2 program. When you have finished, remember to save the corrected version. Type:

```
dsave "@mail.adder2 {RETURN}
```

With the mail.reader1 program, one additional step is necessary. First, load the menu.routine:

```
dload "menu.routine {RETURN}
```

Again, list the routine so that the instructions are on the screen:

```
list {RETURN}
```

Load the mail.reader1 program into the computer:

```
dload "mail.reader1 {RETURN}
```

while the "return to program menu" instructions are on the screen, use the cursor up key

to go to the 5 in line 5000. Press the RETURN key on each of the line numbers between 5000 and 5040. These lines are added to the mail.reader1 program. Next, delete lines 600 and 605:

```
delete 600-605 {RETURN}
```

Finally, save the new version:

```
dsave "@mail.reader1 {RETURN}
```

You now have a system of programs that work together and are controlled by one master program. There is no need to go through all of the steps that are sometimes involved in building one large program. In addition, it is much easier to make changes to individual programs than to change something in a very large program that might have an unnoticed effect. This borders on programmer preference, but I have found this method to be easy and effective.

The directory should show some changes to existing files as well as the addition of the menu.routine file:

```
0 "data file    1        ",87 2a
1      "hello"                 prg
1      "Write Seq.File"        prg
2      "Read Seq.File"         prg
1      "Seq.Data File"         seq
14     "mail.create"           prg
1      "Mail Data"             seq
1      "Mail Pointer"          seq
4      "mail.reader1"          prg
15     "mail.adder1"           prg
19     "mail.adder2"           prg
3      "mail.menu"             prg
1      "menu.routine"          prg
1265 blocks free.
```

You now have a system that will create a file, add to that file, and, in a primitive way, read the file. Two main tasks are left: improving the display features of the mail.reader1 program, and creating a program that will change and delete information in the file. One other feature that will be added within the display program is the capability of reformatting data, with the option of creating a new file for reformatted data.

MAIL.READER2

The present program displays every line in the Mail Data file, including the two separator symbols. You don't really want to see those symbols, so eliminating them should be one of the first tasks in creating a new display program. What else would be nice or useful to have in this display program? The computer could display a list of just the names of the individuals in the file. How about a list of the names and addresses without the phone numbers? Or a display of a single name, address, and phone number? How about a single name and address, without a phone number? An alphabetical list? Can the com-

puter display a range of names and addresses, rather than just the entire list or a single individual? The answer to all of these questions is yes; you can do these things and others also. With all of these possibilities, the obvious solution would be to have a menu that offers these choices.

There are several ways to go about creating a program that uses the same routines as other programs. The most obvious is to retype all of the instructions. A far better (and faster) method is to use a programmer's utility program that allows the merging of program segments. If you have such a utility, operational routines can be saved separately on a diskette that contains a library of such routines. Then, when you are creating a new program, these routines can be incorporated directly into the new program without the necessity of retyping the instructions and with the certain knowledge that the routines will operate correctly (if you have been consistent and careful in your choice of your variable names, etc.). If the routine needs to be placed at a different point in the new program, renumbering can alter the line numbers of the routine so that the routine will fit within the desired program. The use of such utilities is widespread among professional programmers and helps explain why a programmer's work will look similar from program to program.

Without a utility program, you can approximate the library of routines by loading routines separately, changing instruction lines and line numbers that need changing, deleting lines that will not be used, and adding lines that are needed. Then, the complete routine can be incorporated into the new program in the same manner as was used to add the "return to program menu" routine to the mail.create, mail.adder2, and mail.reader1 programs. As long as the instructions are on the screen, placing the cursor on any part of a line and pressing the RETURN key incorporates that instruction into any program in the computer's memory. In fact, such a procedure can replace existing lines if the instructions on the screen have the same line numbers as instructions in the computer's memory. Therefore, it is very important to keep track of the available line numbers within any program. The easiest way to accomplish this is to be as consistent as possible in numbering your routines.

One other problem can occur. Sometimes the routines are too long to completely fit on the screen at one time. When that happens, I split the routine into parts that will fit on the screen and still allow me to load in the new program. This method is certainly more time consuming than if you had access to a good utility program, but it is also less expensive. The results, if you are careful, should be the same.

I will give the specific instructions necessary to change the mail.adder2 program into the new mail.reader2 program, but if you become confused or lost, you can always turn to the end of this chapter and type in the full listing of this mail.reader2 program directly, or obtain the diskette. The instructions might sound complicated, but I assure you that the instructions will work if followed exactly. I have tested these instructions to make certain that they do not contain errors.

Make certain that you have saved any program in memory and then load the mail.adder2 program. Type:

```
dload "mail.adder2 {RETURN}
```

Change line 100 to read:

```
100 rem ***--mail.reader2--***
```

Change line 200 to read:

```
200 blank$ = chr$(10):rem blank line
```

Delete lines 300 to 999:

```
delete 300-999 {RETURN}
```

Note that you can delete to or from line numbers that do not exist. In such cases, the computer deletes all instructions that fall within the range of line numbers specified.

Delete line numbers 2001 and 5999:

```
delete 2001-5999 {RETURN}
```

Line 2000 has not been deleted because instruction 1030 refers to line 2000, and the Commodore renumber utility will not operate if you delete line 2000—an "unresolved reference error in 1030" is displayed.

Renumber the print label routine so that it now begins at line 10000. Type:

```
renumber 10000,5,1000 {RETURN}
```

This command tells the computer to renumber the instructions that begin at line 1000 so that they now begin at line 10000 and are incremented by 5 for each instruction.

Renumber the y/n input subroutine to begin at 18000:

```
renumber 18000,5,10170 {RETURN}
```

Next, renumber the getkey subroutine so that it begins at line number 19000:

```
renumber 19000,5,18090 {RETURN}
```

Now you have, in effect, added 10000 to the line numbers of the y/n input subroutine and the getkey subroutine. Save this beginning of the mail.reader2 program:

```
dsave "mail.reader2 {RETURN}
```

The Return to Menu Routine

Next, you need to add some routines. Load the menu.routine:

```
dload "menu.routine {RETURN}
```

Renumber the routine so that it begins at line number 8000:

```
renumber 8000,5 {RETURN}
```

Note that the starting line number is not necessary, because you are renumbering the entire contents of the computer's memory—the complete program or routine. Again, list

the routine so that the instructions are on the screen:

```
list {RETURN}
```

Reload the mail.reader2 program into the computer:

```
dload "mail.reader2 {RETURN}
```

While the "return to program menu" instructions are on the screen, use the cursor up key to go to the 8 in line 8000. Press the RETURN key on each of the line numbers between 8000 and 8040. These lines are added to the mail.reader2 program. Although you are not finished, save the new version:

```
dsave "@mail.reader2 {RETURN}
```

It is best to periodically save your work so that if something should happen and the computer loses power, you would lose only the amount of work you had entered since the last time you saved the program.

The Read File Routine

There is a part of one other program that you can make use of in the mail.reader2 program—the file routines in the mail.reader1 program. Load the mail.reader1 program (but make certain that you have saved the work you have done so far first):

```
dload "mail.reader1 {RETURN}
```

Delete lines 100 to 299, 500 to 6000, and line 325:

```
delete 100-299 {RETURN}

delete 500-6000 {RETURN}

delete 325 {RETURN}
```

List the routine so that the instructions are on the screen:

```
list {RETURN}
```

The full listing fits on the screen but does not leave much room for other instructions or commands. Use the cursor up-arrow key to position the cursor on the line immediately below instruction line 465. Then type:

```
dload "mail.reader2 {RETURN}
```

While the instructions are on the screen, use the cursor up key to go to the 3 in line 300.

Press the RETURN key on each of the line numbers between 300 and 465. These lines are added to the mail.reader2 program. Save the additional work:

```
dsave "@mail.reader2 {RETURN}
```

There are some changes that need to be made to the lines you have just added. Change line 430 to:

```
430 dim tlines$(rec), nd$(rec), u(rec), r(rec), ad$(rec),
zi$(rec)
```

Change line 380 so that the information is stored in the variable tlines$(i) not in line$(i):

```
380 input#2, tlines$(i)
```

Change line 475 from the colon to:

```
475 tk = rec:rem total k = # of lines
```

Finally, add lines 480 and 485:

```
480 :
485 :
```

These lines are merely for formatting purposes and are not needed for the operation of the program. When you have finished making all these changes, the added and changed lines should look like the following:

```
300 rem **--user message--**
305 scnclr:color chtr,red
310 char 0,32,5
315 print chr$(15);"ONE MOMENT PLEASE!";chr$(143)
320 :
400 rem **--read file routine--**
405 :
410 rem *--read pointer file--*
415 dopen#1,"Mail Pointer"
420 input#1,rec
425 dclose#1
430 dim tlines$(rec), nd$(rec), u(rec), r(rec),
                ad$(rec), zi$(rec)
435 :
440 rem *--data file--*
445 dopen#2,"Mail Data"
450 for i = 1 to rec
455 input#2,tlines$(i)
460 next i
465 dclose#2
```

```
470 :
475 tk = rec:rem total k = # of lines
480 :
485 :
```

The Display Routine

You can now proceed to make other changes that are necessary before you have an operational mail.reader2 program. The next series of changes involves the subroutine used to display file information.

Change line 10000 to:

```
10000 rem **--display subroutine--**
```

You can accomplish this change either by editing the existing line or typing the line over again. If you want to make the change by editing the existing line, first list line 10000 by typing:

```
list 10000 {RETURN}
```

Second, use the cursor up-arrow key to move the cursor up to the 1 in line 10000. Third, using the cursor right-arrow key, position the cursor on the character "p" in the word "print" and type:

```
display
```

Fourth, use the cursor right-arrow key to place the cursor on the space. Fifth, use the INST/DEL key to delete the letters leba (reading from right to left in the same manner as the DEL key functions). Sixth, press the RETURN key to accept the corrected version of this instruction line. This same procedure can be used to edit, correct, or change any existing instruction line.

To continue with the necessary changes in the new display subroutine, change line 10015 to:

```
10015 print "Would you like a paper print out? ";
```

Change the indicated lines to:

```
10025 if yes$ = "y" then 10040:rem printer
10030 if yes$ = "n" then 10115:rem screen
```

Delete lines 10085 to 10165:

```
delete 10085-10165 {RETURN}
```

Finally, add the following lines:

```
10085 rem *--printer display--*
10090 file = 4
```

```
10095 dvic = 4
10100 cmnd = 7
10105 goto 10500:rem open instruction
10110 :
10115 rem *--screen display--*
10120 file = 3
10125 dvic = 3
10130 cmnd = 1
10135 goto 10500:rem open instruction
10140 :
10145 :
10500 rem *--open instruction--*
10505 open file,dvic,cmnd
10510 return
10515 :
10520 :
```

This new display routine allows you to use one print statement in each of the different display options. The output will go to either the printer (device four) or to the screen (device three), depending on the response given to the question displayed by line 10015.

With this last change, you have concluded the modification of the mail.adder2 program. The program in the computer's memory is the "skeleton" of the soon-to-be mail.reader2 program. You might want to save this portion out to the diskette. Type:

```
dsave "@mail.reader2 {RETURN}
```

The @ symbol (or replace symbol) must be included whenever you are saving a program with a name that is already on the diskette. In other words, because you already have one version of mail.reader2 on the diskette, you cannot use the normal save procedure. Instead of simply saving the new version, you must instruct the computer to replace the old version with the current version. If you do not want to lose the old version, a different name must be given to the new version.

Additional Subroutines

There are two brief subroutines that need to be added. The first of these subroutines is the "line up numbers" subroutine; the second is the "return to menu" subroutine. Type in the following:

```
11000 rem **--line up numbers--**
11005 if i < 10 then ta = 4
11010 if i > 9 and i < 100 then ta = 3
11015 if i > 99 and i < 1000 then ta = 2
11020 if i > 999 then ta = 1
11025 return
11030 :
11035 :
```

This subroutine performs the function for which it is named. It aligns the numbers by check-

ing how many digits there are in the number and then setting the tab value accordingly.

Type in the next subroutine:

```
20000 rem **--menu return subroutine--**
20005 print#file,""
20010 close file
20015 print
20020 print "Press ";
20025 color chtr,white
20030 print "RETURN";
20035 color chtr,yellow
20040 print "to go to the Display Menu: ";
20045 gosub 19000:rem getkey subroutine
20050 print
20055 goto 500:rem display menu
```

This subroutine is used by all the main options in the mail.reader2 program. It provides instructions to the user after closing any file that is open.

The Reader2 Menu Routine

You are now ready to proceed with the main part of the mail.reader2 program. You need to fill in the skeleton with the remaining structure of the program, beginning with a menu of the options available. Follow the instructions carefully. *Make certain* that you have saved what has been done so far, and then load the mail.menu program into the computer. (At this point, I am assuming you have NOT typed in the complete program listed at the end of this chapter. Instead, the program I am referring to is the listing included with the text at the beginning of this chapter.)

```
dload "mail.menu {RETURN}
```

List the program:

```
list {RETURN}
```

The full listing fits on the screen, but it does not leave much room for other instructions or commands. Use the cursor up-arrow key to position the cursor on the line below instruction line 605. Then type:

```
dload "mail.reader2 {RETURN}
```

After the mail.reader2 program has been loaded, line 500 of the mail.menu program should have disappeared off the top of the screen. That line needs to be changed anyway, so the loss is not significant. While the mail.menu instructions are on the screen, go to the 5 in line 505. Press the RETURN key on each of the line numbers between 505 and 605. These lines are added to the mail.reader2 program.

The next step is to add and change lines so that the mail.reader2 menu routine matches the listing given below:

```
500 rem **--mail.reader2 menu--**
505 tb = 20:rem tab value--20 spaces to the right
510 scnclr
515 color chtr,white
520 char 0,24,1,"DISPLAY MENU":print
525 color chtr,yellow
530 print:print tab(tb) "1.  INFORMATION--ORIGINAL ORDER"
535 print:print tab(tb) "2.  NAMES ONLY"
540 print:print tab(tb) "3.  INFORMATION--NO PHONE"
545 print:print tab(tb) "4.  SPECIFIC NAME"
550 print:print tab(tb) "5.  SPECIFIC NAME--NO PHONE"
555 print:print tab(tb) "6.  INFORMATION--RANGE"
557 print:print tab(tb) "7.  INFORMATION--ALPHABETICAL"
559 print:print tab(tb) "8.  RETURN TO PROGRAM MENU"
560 print:print tab(tb)
565 input "Which Number Please";number$
570 number = val(number$)
575 :
580 if number = 1 then 1000
585 if number = 2 then 2000
590 if number = 3 then 3000
595 if number = 4 then 4000
600 if number = 5 then 5000
605 if number = 6 then 6000
610 if number = 7 then 7000
615 if number = 8 then 8000
620 :
625 rem *--incorrect choice message--*
630 print:print
635 color chtr,white:print tab(tb) "Incorrect Choice!"
640 color chtr,yellow:print:print tab(tb) "Press ";
645 color chtr,white:print "RETURN";
650 color chtr,yellow:print " to continue:";
655 gosub 19000:rem getkey subroutine
660 goto 500:rem menu--check again
665 :
670 :
```

You might find it easier to simply type these lines in rather than make all the necessary changes. Lines 557 and 559 are included for those who have modified the mail.menu program. If you have been following along with the programs, these lines of code should now be easy to understand. You are doing the same sequence of programming you did when you created the mail.menu program. You format the menu display and request a number. Now, you have the basic structure for the rest of the program. All that is necessary is to fill in the code for each routine.

What happens if the user enters a character other than a number between one and eight? This possibility is the reason for the incorrect choice message routine. If the number does not contain a value between one and eight, the computer will not branch to any of the major routines. Immediately after the computer has checked the value of the number against the constant value of eight (line 615), the computer will proceed to the next

instruction, the incorrect choice message routine.

This small routine displays a message whenever the user enters a number that is not between one and eight. After the incorrect choice message (line 635) has been displayed, the user is told to press the RETURN key in order to again be given the opportunity to enter a valid number. Once the user has entered a valid number, control passes to the desired routine.

Note that the routines get progressively more difficult to follow, and it really is not the intent of this book to teach the concepts behind routines such as sorting and searching. It is, however, within its scope to present examples of such routines so that readers can make use of these routines in their own file-manipulation programs.

The Original Order Routine

```
1000 rem **--original order routine--**
1005 gosub 10000:rem display subroutine
1010 scnclr
1015 for i = 1 to tk
1020 if tlines$(i) = "*" then 1040
1025 if tlines$(i) = "!" then print#file,blank$:goto 1040
1030 gosub 11000:rem line up numbers
1035 print#file,tab(ta);i;" ";tlines$(i)
1040 next i
1045 goto 20000:rem menu return subroutine
1050 :
1055 :
```

If you look closely, this routine is very similar to the original mail.reader1 display routine. All this routine does is display all the information lines in the file in the order they were entered. With lines 1020 and 1025, you have eliminated the display of the separator symbols * and !. There are two additional lines that are new. Lines 1005 and 1050 direct the computer to separate routines used by each of the main routines. Line 1005, as indicated by the REM statement, is the code that asks if the user wants the information displayed on a printer or on the screen. This instruction uses a GOSUB statement that directs the computer to go to the instructions that begin at line 10000 and follow those instructions until the computer encounters a RETURN statement. At that point, the computer returns to the instruction following the GOSUB instruction. Line 1050 directs the computer to return the user to the display menu routine when the user is ready.

The Name Only Routine

```
2000 rem **--name only routine--**
2005 gosub 10000:rem display subroutine
2010 scnclr
2015 for i = 1 to tk - 1
2020 if tlines$(i) = tlines$(1) then 2040
2025 if tlines$(i) = "!" then 2040
2030 goto 2055
2035 :
2040 gosub 11000:rem line up numbers
```

```
2045 if tlines$(i) = tlines$(1) then print#file,tab(ta)
                                    i;" ";tlines$(1)
2050 if tlines$(i) = "!" then print#file,tab(ta)
                                    i + 1;" ";tlines$(i + 1)
2055 next i
2060 goto 20000:rem menu return subroutine
2065 :
2070 :
```

This routine should not be very difficult to understand. You want to print only those lines that follow the ! separator. Those lines are printed because they should contain the names of the individuals. The instruction at 2020 is necessary because there is no separator for the first name. You use tk – 1 because you do not want to get the last ! separator, which has no name to follow it yet. The instruction in line 2040 directs the computer to the subroutine that provides for the alignment of the number that precedes the name. This number is for the first line of information in each set of information lines.

The No Phone Routine

```
3000 rem **--no phone routine--**
3005 gosub 10000:rem display subroutine
3010 scnclr
3015 for i = 1 to tk
3020 gosub 11000:rem line up numbers
3025 if tlines$(i) = "*" then i = i + 1:goto 3040
3030 if tlines$(i) = "!" then print#file,blank$:goto 3040
3035 print#file,tlines$(i)
3040 next i
3045 goto 20000:rem menu return subroutine
3050 :
3055 :
```

This routine should look completely familiar. It is practically the same routine you used to print a label in mail.adder2. The effect is the same here also. You can print a mailing label for every person in the file with this routine. Because just about every type of printer handles things differently, you will probably need to add some code to this routine to get the labels spaced properly. One method of spacing is to determine the number of lines on the label, and between labels, and then adjust the routine to always space exactly that number of lines, regardless of the number of lines to be printed. That method would always start the printer at the top of the label and not center the material on the label, but is probably the easiest method to develop.

The Search Routine

```
4000 rem **--search routine--**
4005 gosub 10000:rem display subroutine
4010 scnclr
4015 print "Type the word ";
4020 color chtr,white
```

```
4025 print "END";
4030 color chtr,yellow
4035 print " when finished."
4040 print
4045 print "Name to find: ";
4050 gosub 19000:rem getkey subroutine
4055 find$ = t$:t$ = ""
4060 if find$ = "END" or find$ = "end" then 4155
4065 print
4070 for i = 1 to tk
4075 if tlines$(i) = find$ then 4085
4080 goto 4140
4085 if tlines$(i) = "*" then 4140
4090 if tlines$(i) = "!" then 4140
4095 gosub 11000:rem line up numbers
4100 print#file,blank$
4105 print#file,tab(ta) i;" ";tlines$(i)
4110 print#file,tab(ta + (8 - ta)) tlines$(i + 1)
4115 print#file,tab(ta + (8 - ta)) tlines$(i + 2)
4120 if tlines$(i + 3) <> "*" then print#file,tab(ta + (8 - ta))
                    tlines$(i + 3)
4125 if tlines$(i + 4) = "*" then 4135
4130 print#file,tab(ta + (8 - ta)) tlines$(i + 4):goto 4140
4135 print#file,tab(ta + (8 - ta)) tlines$(i + 5)
4140 next i
4145 print
4150 goto 4015:rem repeat until done
4155 goto 20000:rem menu return subroutine
4160 :
4165 :
```

The routines begin to get more difficult now. To this point, you have not really made any assumptions about the number of lines of information in each set. This routine, however, assumes that there are a maximum of five lines containing information in any set. If you want a greater maximum, additional code will have to be added to print out those other lines. The additional code would follow the pattern of lines 4120 to 4135.

You begin in the same way as usual with lines 4000, 4005, and 4010 (the routine name, gosub display routine, and clear screen lines). Lines 4015 to 4035 give instructions to the user to type the word "end" when the user is finished looking for a specific name. Line 4045 requests the name from the user, and line 4050 sends the computer to the getkey subroutine to accept the name entered by the user. That entry is stored in the string variable find$. Line 4060 checks the contents of find$ to see if it contains the word "END" or "end." If it does, the computer is directed to go to line 4155, which further directs the computer to go to the menu return routine.

You might logically ask why line 4060 does not instruct the computer to go directly to the menu return routine. The reason lies in the necessity of structuring the various routines in the same way, so that any programmer can locate the exit point of the routine easily. There are a number of GOTO statements in this routine, but all of them direct the computer (and any programmer) to various lines within the routine. In following the logic of this routine (and all the other routines also), you never need to look outside the routine,

except to find the display routine, the line up numbers routine, and the exit routine, which are common to all other routines. The idea is to keep the flow of logic in one place as much as possible. You enter at the top of the routine and exit at the base of the routine. This is the case for all of the routines.

Lines 4070 to 4140 are the heart of the routine. They are also the boundaries of the loop, used to find and print the information associated with a specific name. Line 4075 checks the contents of tlines$(i) to see if it equals the contents of find$. If it does, the computer is instructed to jump over the next instruction. If it does not, the next instruction is executed. Line 4080 is reached only if the contents of tlines$(i) and find$ do not match, and 4085 is reached only if they do match. Lines 4085 and 4090 check for the separators and skip them when they are found. At this point in the routine, you have found the name you are looking for, and now want to print out the information associated with this name. Assume that the first three lines will not contain a separator and, therefore, will automatically be printed. Lines 4105, 4110, and 4115 accomplish this task.

Lines 4120 to 4135 are lines of code that require some thought. If the fourth information line does not contain the separator *, then you want to print this line also (4120); if it does contain the separator, you do not want the fourth information line printed. Instead, you know that the fifth information line contains something to be printed (the line following the asterisk will have the phone number if there is a phone number). Line 4130 prints the fifth information line (i + 4). Line 4125 first checks the fifth information line to see if it contains the asterisk separator. If it does contain the separator, you need to jump over 4130 (the instruction that prints that fifth information line) and, instead, print the sixth information line (4135).

Go back through the explanation if you are not certain you understand. This same routine, combined with the previous one, is used for the next routine. The instructions **tab(ta)** and **tab(ta + (8 − ta))** are formatting instructions designed to make certain that the displayed information is left-justified.

The Search Routine with No Phone Number

```
5000 rem **--search routine/no phone--**
5005 gosub 10000:rem display subroutine
5010 scnclr
5015 print "Type the word ";
5020 color chtr,white
5025 print "END";
5030 color chtr,yellow
5035 print " when finished."
5040 print
5045 print "Name to find: ";
5050 gosub 19000:rem getkey subroutine
5055 find$ = t$:t$ = ""
5060 if find$ = "END" or find$ = "end" then 5160
5065 print
5070 for i = 1 to tk
5075 if tlines$(i) = find$ then 5085
5080 goto 5145
5085 if tlines$(i) = "*" then i = i + 1:goto 5145
5090 if tlines$(i) = "!" then print#file,blank$:goto 5145
```

```
5095 gosub 11000:rem line up numbers
5100 print#file,blank$
5105 print#file,tab(ta) i;" ";tlines$(i)
5110 print#file,tab(ta + (8 - ta)) tlines$(i + 1)
5115 print#file,tab(ta + (8 - ta)) tlines$(i + 2)
5120 if tlines$(i + 3) <> "*" then print#file,tab(ta + (8 - ta))
                    tlines$(i + 3)
5125 if tlines$(i + 3) = "*" then i = i + 1:goto 5145
5130 if tlines$(i + 4) = "*" then i = i + 1:goto 5145
5135 print#file,tab(ta + (8 - ta)) tlines$(i + 4):goto 5145
5140 print#file,tab(ta + (8 - ta)) tlines$(i + 5)
5145 next i
5150 print
5155 goto 5015:rem repeat until done
5160 goto 20000:rem menu return subroutine
5165 :
5170 :
```

This routine is included for a number of reasons. First, it is a very useful routine because it can be used to print out a specific mailing label. Second, and most importantly, it shows how two routines can be combined into a third routine. Very few programs will do everything anyone could ever want of them, but if a person understands these separate routines, combining two or more to form others should be possible. Quite a number of combinations are possible, and might be useful to some people.

As you can see, this routine is exactly the same as the previous one as far as the instruction at 5085. The only difference is that when the * separator is found, one is added to i, thus skipping the phone number. Lines 5100 through 5120 are the same as 4100 through 4120. The instructions at 5125 and 5130 are the only different instructions. Both of those instructions simply check to see which information line contains the separator symbol, and then advance the counter by one. The end of the routine is the same as the end of the previous routine.

The Range Routine

With the routines at 4000 and 5000, you have the ability to search for a specific name and display that name, either with the phone number or without the phone number. Both of these routines require that you know and type in the exact spelling of the name, including spaces. That presents a reason for the next routine, the range routine. With this routine, you will only need to know the starting and ending information lines to be able to display the information you want. You can obtain those numbers from the "names only" routine. I will present the range routine only, but you might want to combine this routine with the names only routine, and possibly some others also.

```
6000 rem **--range routine--**
6005 gosub 10000:rem display subroutine
6010 scnclr
6015 char 0,0,5
6020 input "Type the beginning line number please: ";bg
6025 print:print
```

```
6030 if bg < 1 then print "The number is too small! Please try
                    again.":goto 6020
6035 input "Type the ending line number please: ";ed
6040 print
6045 if ed > tk then print "The number is too large! Please try
                    again.":goto 6025
6050 for i = bg to ed
6055 gosub 11000:rem line up numbers
6060 if tlines$(i) = "*" then i = i + 1:goto 6075
6065 if tlines$(i) = "!" then print#file,blank$:goto 6075
6070 print#file,tab(ta) i;" ";tlines$(i)
6075 next i
6080 goto 20000:rem menu return subroutine
6085 :
6090 :
```

Line 6020 asks for the beginning information line number. Remember that you can check the numbers first by using the names only routine or by actually including that routine at the beginning of this one. Line 6030 checks the number typed to see if it is less than one, the number of the first information line. If it is too small, a message is printed and the user is again asked for a beginning number. Line 6035 requests the ending information line number and goes through the same checking process, this time for a number larger than the maximum number of information lines. Then comes the loop (lines 6050 to 6075). I have included the code for printing the information without the phone number (6060), thus providing a routine that can print out a selected range of mailing labels.

I have tried to show how you can take various routines and combine them in just about any way you might want. With the addition of each new routine, the number of possible combinations of routines increases so much that no single programmer could include all possibilities within one program, but with a minimum of understanding, everyone can create combinations of routines to meet their needs.

The Alphabetical Order Routine

These are the most complex routines. I will not even attempt to explain the logic involved in all parts of this alphabetizing routine, because many books have been written on various sorting techniques. The sort method I am including is sometimes called the *Quicksort* technique. There are a number of other public domain sorting routines that could have been used, such as the bubble sort or the Shell-Metzner sort, but I decided on the Quicksort because it is very efficient and somewhat less publicized. The sort has been modified to enable it to work with string variables. Otherwise, the sort subroutine is a standard routine that can be used in a number of different ways to order lists composed of numbers or letters. For example, if you want to display the information in the Mail Data file in zip code order, you first need to access the zip codes and their associated information lines in either ascending or descending order. The creation of such a routine would require that you completely understand another feature of this routine: the flexibility possible with string variables and the manner of utilizing that flexibility. Again, I will not try to fully explain the logic or programming power behind the BASIC statements of LEFT$, MID$, or RIGHT$. I strongly encourage you to learn as much as possible about these BASIC statements and how they can be used to take string variables apart and put them back together.

The alphabetizing routine will be presented in sections. The first section makes use of string variable flexibility to do the following: access the last part of characters in that first information line; reverse the order of that information line, placing the last group of characters first (i.e., David Miller becomes Miller, David); and combine all other information lines associated with this first line into one long string variable, ad$(i):

Miller, David ** 999 Any St. ** Somewhere CA9000 ** 800-444-1111

```
7000 rem **--alphabetical order--**
7005 scnclr
7010 char 0,15,9
7015 color chtr,white
7020 print "WORKING--PLEASE DON'T TOUCH!!"
7025 color chtr,yellow
7030 :
7035 :
7040 rem *--get first info.line--*
7045 for i = 1 to tk - 1
7050 if tlines$(i) = tlines$(1) then 7075
7055 if tlines$(i) = "!" then i = i + 1:goto 7075
7060 goto 7135:rem next i
7065 :
7070 :
7075 rem *--reverse order--*
7080 na = len(tlines$(i))
7085 for va = 1 to na
7090 if asc(mdi$(tlines$(i),va,1)) = 32 then vz = va
7095 next va
7100 if vz = 0 or vz > na then ad$(i) = tlines$(i):goto 7110
7105 ad$(i) = mid$(tlines$(i),vz + 1,na - vz) + ", " +
                  left$(tlines$(i),vz)
7110 ad$(i) = ad$(i) + "**" + tlines$(i + 1) + "**" + tlines$(i + 2)
7115 if tlines$(i + 3) <> "*" then ad$(i) = ad$(i) + "**" +
                  tlines$(i + 3)
7120 if tlines$(i + 4) = "*" then 7130
7125 ad$(i) = ad$(i) + "**" + tlines$(i + 4):goto 7135
7130 ad$(i) = ad$(i) + "**" + tlines$(i + 5)
7135 next i
7140 :
7145 :
7150 rem *--renumber for sort--*
7155 j = 1
7160 for i = 1 to tk
7165 if len(ad$(i)) > 0 then nd$(j) = ad$(i): j = j + 1
7170 next i
7175 n = j - 1
7180 :
7185 :
```

As you can see, the routines get more complex. If you do not understand the left$, mid$, and right$ statements, the best thing to do is to get a clear definition of them from a book devoted to teaching BASIC, and then practice their uses. Essentially, they perform

the functions for which they are named. The left$ statement will retrieve a specified number of characters beginning at the left side of a string variable. Left$(a$,4) gets the first four characters of the string variable a$; i.e., if a$ equals "John M. Smith" left$(a$,4) is "John." The right$ statement retrieves a specified number of characters starting with the rightmost character of a string variable. Right$(a$,5) gets the last five characters in the string variable a$; i.e., if a$ equals "John M. Smith" right$(a$,5) is "Smith." The mid$ statement retrieves a specified number of characters from a specified position within a string variable. Mid$(a$,6,2) gets the next two characters beginning at the sixth character in the string variable a$; i.e., if a$ equals "John M. Smith," mid$(a$,6,2) is "M."

Therefore, the instructions in 7040 to 7060 identify the first information line in each set of data. Lines 7075-7135 reverse the order of the first information line and then combine all the other information lines associated with it. Finally, 7150 to 7175 are the instructions that renumber the sets of information in such a way that the sort subroutine can function.

```
7190 rem *--quicksort--*
7195 sa = 1
7200 char 0,15,11
7205 print "STILL WORKING--PLEASE DON'T TOUCH!!"
7210 u(1) = 1
7215 r(1) = n
7220 ua = u(sa)
7225 ra = r(sa)
7230 sa = sa - 1
7235 uz = ua
7240 rz = ra
7245 x$ = nd$(int((ua + ra)/2))
7250 c = c + 1
7255 if nd$(uz) = x$ or nd$(uz) > x$ then 7270
7260 uz = uz + 1
7265 goto 7250
7270 c = ca
7275 if x$ = nd$(rz) or x$ > nd$(rz) then 7290
7280 rz = rz - 1
7285 goto 7270
7290 if uz > rz then 7325
7295 s = s + 1
7300 tp$ = nd$(uz)
7305 nd$(uz) = nd$(rz)
7310 nd$(rz) = tp$
7315 uz = uz + 1
7320 rz = rz - 1
7325 if uz = rz or uz < rz then 7250
7330 if uz = ra or uz > ra then 7350
7335 sa = sa + 1
7340 u(sa) = uz
7345 r(sa) = ra
7350 ra = rz
7355 if ua < ra then 7235
7360 if sa > 0 then 7220
```

```
7365 rem sort completed!
7370 :
7375 :
```

Now, you have access to a sorting method. The only code necessary outside this subroutine to transfer the results to another program is to set:

1. the dim of u( ) and r( ) to the number of things to be sorted.
2. the numeric variable n equal to the number of things to be sorted.

If you have a different sort method that you like or understand better and want to include it instead, the code for your sort should replace the code between lines 7190 and 7365.

You still need to display the results after sorting. The code to display the results is presented in an elementary way. This display routine might not work exactly right if your original file is not in the following format:

Information-line # 1: First Last (name)
Information-line # 2: Address
Information-line # 3: City State Zip
Information-line # 4: *
Information-line # 5: Phone number
Information-line # 6: !

After the list has been alphabetized, the display will have the following format:

Line # 1: Last, First (name)
Line # 2: Address
Line # 3: City State Zip
Line # 4: Phone number

I will leave to those of you who want to or are able to use the flexibility in string variables to format the display in any way you desire.

```
7380 rem *--display--*
7385 gosub 10000:rem display subroutine
7390 scnclr
7395 for i = 1 to n
7400 vz = 1:q = 1
7405 na = len(nd$(i))
7410 for va = 1 to na
7415 if mid$(nd$(i),va,2) = "**" then 7425
7420 goto 7440
7425 zi$(q) = mid$(nd$(i),vz,va - vz)
7430 vz = va + 2
7435 q = q + 1
7440 next va
7445 :
7450 zi$(q) = mid$(nd$(i),vz,na - (vz - 1))
```

```
7455 :
7460 for e = 1 to q
7465 print#file,zi$(e)
7470 next e
7475 :
7480 print#file,blank$
7485 next i
7490 goto 20000:rem menu return subroutine
7495 :
7500 :
```

You now have an opportunity to create a sequential access file in a way that might be more powerful than in the mailing list system. The usefulness of the new file-creation method depends on the programmer's knowledge of and willingness to work with string variables (i.e., left$, mid$, right$, len, str$, val, and dim). All of the information associated with tlines$(1), that is, the address, city, state, zip code, and phone number, are stored in the string variable nd$(1). Everything for the next name is stored in nd$(2), and so on. If you want to locate the zip code, all you need to do is use the mid$ function to determine where in the string the zip code is located. For example, the following short program demonstrates one way of extracting the zip code from a similar string:

```
100 a$ = "Miller, David **9999 Any St.**Somewhere CA 90000** 800-444-1111
200 ln = len(a$)
300 for i = 1 to ln
400 if mid$(a$,i,2) = "**" then c = c + 1
500 if c = 3 then print mid$(a$,i-5,5):c = 0
600 next i
```

This program makes two assumptions that might not always be true: (1) zip codes are only five digits long, and (2) there are only four lines of information. If either of these assumptions are ever false, the program will not properly retrieve the zip code. Other programming techniques will need to be utilized then.

Once you have typed in all the programming code for this mail.reader2 program, be sure to save it to the diskette. Remember, you must use the replace (@) symbol because you have previously saved work under the mail.reader2 name:

```
dsave "mail.reader2 {RETURN}
```

Now the list of files should show:

```
0 "data file    1       ",87 2a
1      "hello"                prg
1      "Write Seq.File"       prg
2      "Read Seq.File"        prg
1      "Seq.Data File"        seq
14     "mail.create"          prg
1      "Mail Data"            seq
1      "Mail Pointer"         seq
```

```
4       "mail.reader1"          prg
15      "mail.adder1"           prg
19      "mail.adder2"           prg
3       "mail.menu"             prg
1       "menu.routine"          prg
36     "mail.reader2"           prg
1229 blocks free.
```

The number of free blocks may be different, depending on the number of blocks used in the Mail Data sequential file. If you have already added a few names and addresses, the number of blocks in that file will be larger than one and the number of free blocks will be less than 1229. All the programs and routines discussed in this chapter are shown in Fig. 6-1. The next chapter examines ways of correcting, changing, or deleting information from a file.

CHAPTER 6 QUESTIONS

1. True or False: BASIC 7.0 allows you to run a program from within another program.
2. What BASIC word allows you to horizontally position text on the screen?
3. Which BASIC word is used to instruct the computer to go to a subroutine?
4. Which BASIC word is used to instruct the computer to return from a subroutine?
5. True or False: In programming, it is a good idea to have just one main entrance and exit point in every routine.
6. Name three public domain sorting routines.
7. What are the three main BASIC words that provide a great deal of power in working with strings?
8. What BASIC word retrieves a specified number of characters from a specified position within a string variable?
9. Name four other BASIC words that can be used in some way with string variables.
10. True or False: When you save a file with the same name as a file already on the disk, the first file is replaced by the second file.

Fig. 6-1. The Mailing List System programs.

The short mail-menu Routine

```
500 rem **--mail menu--**
505 tb = 20:rem tab value--20 spaces to the right
510 scnclr
520 char 0,24,2,"MAIL PROGRAM MENU":print
530 print:print tab(tb) "1.  FILE CREATION PROGRAM"
535 print:print tab(tb) "2.  FILE ADDITION PROGRAM"
540 print:print tab(tb) "3.  FILE DISPLAY PROGRAM"
545 print:print tab(tb) "4.  FILE CORRECTION PROGRAM"
550 print:print tab(tb) "5.  LIST OF FILES"
555 print:print tab(tb) "6.  END"
560 print:print tab(tb)
565 input "Which Program Number";number$
570 number = val(number$)
575 :
580 if number = 1 then run "mail.create
585 if number = 2 then run "mail.adder2
590 if number = 3 then run "mail.reader1
595 if number = 4 then run "mail.correction
600 if number = 5 then directory:goto 500
605 if number = 6 then end
```

```
5000 rem **--return to program menu--**
5005 scnclr
5010 char 0,15,9
5015 color chtr,white
5020 print "LOADING THE MAIL MENU PROGRAM"
5025 color chtr,yellow
5030 run "mail.menu"
5035 :
5040 :
```

The mail.menu Program

```
 100 rem ***--mail.menu--***
 105 :
 110 :
 115 rem **--initialization--**
 120 :black =  1
 125 :white =  2
 130 :::red = 11
 135 yellow =  8
 140 ::chtr =  5
 145 bckgnd =  0
 150 :
 205 print chr$(14):rem upper/lower case mode
 210 print chr$(8):rem disable char.set change
 215 color bckgnd,black
 220 :
 225 :
 500 rem **--mail menu--**
 505 tb = 20:rem tab value--20 spaces to the right
 510 scnclr
 515 color chtr,white
 520 char 0,24,2,"MAIL PROGRAM MENU":print
 525 color chtr,yellow
 530 print:print tab(tb) "1.  FILE CREATION PROGRAM"
 535 print:print tab(tb) "2.  FILE ADDITION PROGRAM"
 540 print:print tab(tb) "3.  FILE DISPLAY PROGRAM"
 545 print:print tab(tb) "4.  FILE CORRECTION PROGRAM"
 550 print:print tab(tb) "5.  LIST OF FILES"
 555 print:print tab(tb) "6.  END"
 560 print:print tab(tb)
 565 input "Which Program Number";number$
 570 number = val(number$)
 575 :
 580 if number = 1 then 1000
 585 if number = 2 then 2000
 590 if number = 3 then 3000
 595 if number = 4 then 4000
 600 if number = 5 then 5000
 605 if number = 6 then 6000
 610 :
 615 rem *--incorrect choice message--*
 620 print:print
 625 color chtr,white:print tab(tb) "Incorrect Choice!"
 630 color chtr,yellow:print:print tab(tb) "Press ";
 635 color chtr,white:print "RETURN";
 640 color chtr,yellow:print " to continue:";
 645 gosub 9000:rem getkey subroutine
 650 goto 500:rem menu--check again
 655 :
 660 :
1000 rem **--file creation program--**
1005 scnclr
1010 char 0,32,5
1015 color chtr,white
1020 print "WARNING!"
1025 print:print
1030 color chtr,yellow
```

```
1035 print "If the 'Mail Data' file already exists, do ";
1040 color chtr,red
1045 print "NOT";
1050 color chtr,yellow
1055 print " run this program!"
1060 print
1065 print "Do you want the file creation program? ";
1070 gosub 8000:rem y/n input routine
1080 if yes$ = "n" then 500:rem menu
1085 print:print
1090 print "Are you sure? Type ";
1095 color chtr,white
1100 print "YES";
1105 color chtr,yellow
1110 print " if you are: ";
1115 gosub 9000:rem getkey subroutine
1120 yes$ = t$:t$ = ""
1125 if yes$ = "YES" or yes$ = "yes" then 1135
1130 goto 1000:rem check again
1135 file$ = "MAIL.CREATE"
1140 gosub 7100:rem new program routine after the question
1145 run "mail.create"
1150 :
1155 :
2000 rem **--file addition program--**
2005 file$ = "MAIL.ADDER2"
2010 gosub 7000:rem new program routine
2015 run "mail.adder2"
2020 :
2025 :
3000 rem **--file display program--**
3005 file$ = "MAIL.READER2"
3010 gosub 7000:rem new program routine
3015 run "mail.reader2"
3020 :
3025 :
4000 rem **--file correction program--**
4005 file$ = "MAIL.CORRECTION"
4010 gosub 7000:rem new program routine
4015 run "mail.correction"
4020 :
4025 :
5000 rem **--list of files routine--**
5005 scnclr
5010 directory
5015 print:print
5020 print "Are you ready to return to the menu? ";
5025 gosub 8000:rem y/n input routine
5030 if yes$ = "y" then 500:rem menu
5035 goto 5000:rem check again
5040 :
5045 :
6000 rem **--end routine--**
6005 scnclr
6010 color chtr,white
6015 char 0,18,9,"That's all for this session! See you next time."
6020 color chtr,yellow
6025 char 0,0,22
6030 end
6035 :
6040 :
7000 rem **--new program routine--**
7005 scnclr:char 0,15,5
7010 print "You have selected the ";
7015 color chtr,red
7020 print file$;
7025 color chtr,yellow
7030 print " program."
7035 char 0,15,9
```

```
7040 print "Is this the program you want? ";
7045 gosub 8000:rem y/n input routine
7050 if yes$ = "n" then 500:rem menu
7055 :
7100 rem *--loading message--*
7105 scnclr
7110 char 0,15,5
7115 print "Please wait! I'm loading....";
7120 color chtr,red
7125 print file$
7130 color chtr,yellow
7135 return
7140 :
7145 :
8000 rem **--y/n input subroutine--**
8005 print "Type a ";
8010 color chtr,white:print "Y";
8015 color chtr,yellow:print " or ";
8020 color chtr,white:print "N";
8025 color chtr,yellow:print " :";
8030 gosub 9000:rem getkey subroutine
8035 yes$ = t$:t$ = ""
8040 print
8045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
8050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
8055 print
8060 color chtr,white:print "Incorrect Choice!"
8065 color chtr,yellow
8070 print
8075 goto 8000:rem ask again
8080 :
8085 :
9000 rem **--getkey subroutine--**
9005 getkey a$
9010 if a$ = chr$(20) then 9040:rem delete char.
9015 if a$ = chr$(13) then return:rem return key
9020 print a$;
9025 t$ = t$ + a$
9030 goto 9000
9035 :
9040 lg = len(t$)
9045 if lg < 1 then 9000
9050 print a$;
9055 t$ = left$(t$,lg - 1)
9060 goto 9000
```

The mail.reader2 Program

```
100 rem ***--mail.reader2--***
105 :
110 :
115 rem **--initialization--**
120 :black =  1
125 :white =  2
130 :::red = 11
135 yellow =  8
140 ::chtr =  5
145 bckgnd =  0
150 :
200 blank$ = chr$(10):rem blank line
205 print chr$(14):rem upper/lower case mode
210 print chr$(8):rem disable char.set change
215 color bckgnd,black
220 :
225 :
300 rem **--user message--**
305 scnclr:color chr,red
310 char 0,32,5
315 print chr$(15);"ONE MOMENT PLEASE!";chr$(143)
320 :
```

```
400 rem **--read file routine--**
405 :
410 rem *--read pointer file--*
415 dopen#1,"Mail Pointer"
420 input#1,rec
425 dclose#1
430 dim tlines$(rec),nd$(rec),u(rec),r(rec),ad$(rec),zi$(rec)
435 :
440 rem *--data file--*
445 dopen#2,"Mail Data"
450 for i = 1 to rec
455 input#2,tlines$(i)
460 next i
465 dclose#2
470 :
475 tk = rec:rem total k = # of lines
480 :
485 :
500 rem **--reader2 menu--**
505 tb = 20:rem tab value--20 spaces to the right
510 scnclr
515 color chtr,white
520 char 0,24,1,"DISPLAY MENU":print
525 color chtr,yellow
530 print:print tab(tb) "1.  INFORMATION--ORIGINAL ORDER"
535 print:print tab(tb) "2.  NAMES ONLY"
540 print:print tab(tb) "3.  INFORMATION--NO PHONE"
545 print:print tab(tb) "4.  SPECIFIC NAME"
550 print:print tab(tb) "5.  SPECIFIC NAME--NO PHONE"
555 print:print tab(tb) "6.  INFORMATION--RANGE"
557 print:print tab(tb) "7.  INFORMATION--ALPHABETICAL"
559 print:print tab(tb) "8.  RETURN TO PROGRAM MENU"
560 print:print tab(tb)
565 input "Which Number Please ";number$
570 number = val(number$)
575 :
580 if number = 1 then 1000
585 if number = 2 then 2000
590 if number = 3 then 3000
595 if number = 4 then 4000
600 if number = 5 then 5000
605 if number = 6 then 6000
610 if number = 7 then 7000
615 if number = 8 then 8000
620 :
625 rem *--incorrect choice message--*
630 print:print
635 color chtr,white:print tab(tb) "Incorrect Choice!"
640 color chtr,yellow:print:print tab(tb) "Press ";
645 color chtr,white:print "RETURN";
650 color chtr,yellow:print " to continue:";
655 gosub 19000:rem getkey subroutine
660 goto 500:rem menu--check again
665 :
670 :
1000 rem **--original order routine--**
1005 gosub 10000:rem display subroutine
1010 scnclr
1015 for i = 1 to tk
1020 if tlines$(i) = "*" then 1040
1025 if tlines$(i) = "!" then print#file,blank$:goto 1040
1030 gosub 11000:rem line up numbers
1035 print#file,tab(ta);i;" ";tlines$(i)
1040 next i
1045 goto 20000:rem menu return subroutine
1050 :
1055 :
2000 rem **--name only routine--**
2005 gosub 10000:rem display subroutine
```

```
2010 scnclr
2015 for i = 1 to tk - 1
2020 if tlines$(i) = tlines$(1) then 2040
2025 if tlines$(i) = "!" then 2040
2030 goto 2055
2035 :
2040 gosub 11000:rem line up numbers
2045 if tlines$(i) = tlines$(1) then print#file,tab(ta) i;" ";tlines$(1)
2050 if tlines$(i) = "!" then print#file,tab(ta) i + 1;" ";tlines$(i + 1)
2055 next i
2060 goto 20000:rem menu return subroutine
2065 :
2070 :
3000 rem **--no phone routine--**
3005 gosub 10000:rem display subroutine
3010 scnclr
3015 for i = 1 to tk
3020 gosub 11000:rem line up numbers
3025 if tlines$(i) = "*" then i = i + 1:goto 3040
3030 if tlines$(i) = "!" then print#file,blank$:goto 3040
3035 print#file,tlines$(i)
3040 next i
3045 goto 20000:rem menu return subroutine
3050 :
3055 :
4000 rem **--search routine--**
4005 gosub 10000:rem display subroutine
4010 scnclr
4015 print "Type the word ";
4020 color chtr,white
4025 print "END";
4030 color chtr,yellow
4035 print " when finished."
4040 print
4045 print "Name to find: ";
4050 gosub 19000:rem getkey subroutine
4055 find$ = t$:t$ = ""
4060 if find$ = "END" or find$ = "end" then 4155
4065 print
4070 for i = 1 to tk
4075 if tlines$(i) = find$ then 4085
4080 goto 4140
4085 if tlines$(i) = "*" then 4140
4090 if tlines$(i) = "!" then 4140
4095 gosub 11000:rem line up numbers
4100 print#file,blank$
4105 print#file,tab(ta) i;" ";tlines$(i)
4110 print#file,tab(ta + (8 - ta)) tlines$(i + 1)
4115 print#file,tab(ta + (8 - ta)) tlines$(i + 2)
4120 if tlines$(i + 3) <> "*" then print#file,tab(ta + (8 - ta)) tlines$(i + 3)
4125 if tlines$(i + 4) = "*" then 4135
4130 print#file,tab(ta + (8 - ta)) tlines$(i + 4):goto 4140
4135 print#file,tab(ta + (8 - ta)) tlines$(i + 5)
4140 next i
4145 print
4150 goto 4015:rem repeat until done
4155 goto 20000:rem menu return subroutine
4160 :
4165 :
5000 rem **--search routine/no phone--**
5005 gosub 10000:rem display subroutine
5010 scnclr
5015 print "Type the word ";
5020 color chtr,white
5025 print "END";
5030 color chtr,yellow
5035 print " when finished."
5040 print
5045 print "Name to find: ";
```

```
5050 gosub 19000:rem getkey subroutine
5055 find$ = t$:t$ = ""
5060 if find$ = "END" or find$ = "end" then 5160
5065 print
5070 for i = 1 to tk
5075 if tlines$(i) = find$ then 5085
5080 goto 5145
5085 if tlines$(i) = "*" then i = i + 1:goto 5145
5090 if tlines$(i) = "!" then print#file,blank$:goto 5145
5095 gosub 11000:rem line up numbers
5100 print#file,blank$
5105 print#file,tab(ta) i;" ";tlines$(i)
5110 print#file,tab(ta + (8 - ta)) tlines$(i + 1)
5115 print#file,tab(ta + (8 - ta)) tlines$(i + 2)
5120 if tlines$(i + 3) <> "*" then print#file,tab(ta + (8 - ta)) tlines$(i + 3)
5125 if tlines$(i + 3) = "*" then i = i + 1:goto 5145
5130 if tlines$(i + 4) = "*" then i = i + 1:goto 5145
5135 print#file,tab(ta + (8 - ta)) tlines$(i + 4):goto 5145
5140 print#file,tab(ta + (8 - ta)) tlines$(i + 5)
5145 next i
5150 print
5155 goto 5015:rem repeat until done
5160 goto 20000:rem menu return subroutine
5165 :
5170 :
6000 rem **--range routine--**
6005 gosub 10000:rem display subroutine
6010 scnclr
6015 char 0,0,5
6020 input "Type the beginning line number please: ";bg
6025 print:print
6030 if bg < 1 then print "The number is too small! Please try again.":goto 6020
6035 input "Type the ending line number please: ";ed
6040 print
6045 if ed > tk then print "The number is too large! Please try again.":goto 6025
6050 for i = bg to ed
6055 gosub 11000:rem line up numbers
6060 if tlines$(i) = "*" then i = i + 1: goto 6075
6065 if tlines$(i) = "!" then print#file,blank$:goto 6075
6070 print#file,tab(ta) i;" ";tlines$(i)
6075 next i
6080 goto 20000:rem menu return subroutine
6085 :
6090 :
7000 rem **--alphabetical order--**
7005 scnclr
7010 char 0,15,9
7015 color chtr,white
7020 print "WORKING--PLEASE DON'T TOUCH!!"
7025 color chtr,yellow
7030 :
7035 :
7040 rem *--get first info.line--*
7045 for i = 1 to tk - 1
7050 if tlines$(i) = tlines$(1) then 7075
7055 if tlines$(i) = "!" then i = i + 1:goto 7075
7060 goto 7135:rem next i
7065 :
7070 :
7075 rem *--reverse order--*
7080 na = len(tlines$(i))
7085 for va = 1 to na
7090 if asc(mid$(tlines$(i),va,1)) = 32 then vz = va
7095 next va
7100 if vz = 0 or vz > na then ad$(i) = tlines$(i):goto 7110
7105 ad$(i) = mid$(tlines$(i),vz + 1,na - vz) + ", " + left$(tlines$(i),vz)
7110 ad$(i) = ad$(i) + "**" + tlines$(i + 1) + "**" + tlines$(i + 2)
7115 if tlines$(i + 3) <> "*" then ad$(i) = ad$(i) + "**" + tlines$(i + 3)
7120 if tlines$(i + 4) = "*" then 7130
```

```
7125 ad$(i) = ad$(i) + "**" + tlines$(i + 4):goto 7135
7130 ad$(i) = ad$(i) + "**" + tlines$(i + 5)
7135 next i
7140 :
7145 :
7150 rem *--renumber for sort--*
7155 j = 1
7160 for i = 1 to tk
7165 if len(ad$(i)) > 0 then nd$(j) = ad$(i): j = j + 1
7170 next i
7175 n = j - 1
7180 :
7185 :
7190 rem *--quicksort--*
7195 sa = 1
7200 char 0,15,11
7205 print "STILL WORKING--PLEASE DON'T TOUCH!!"
7210 u(1) = 1
7215 r(1) = n
7220 ua = u(sa)
7225 ra = r(sa)
7230 sa = sa - 1
7235 uz = ua
7240 rz = ra
7245 x$ = nd$(int((ua + ra)/2))
7250 c = c + 1
7255 if nd$(uz) = x$ or nd$(uz) > x$ then 7270
7260 uz = uz + 1
7265 goto 7250
7270 c = ca
7275 if x$ = nd$(rz) or x$ > nd$(rz) then 7290
7280 rz = rz - 1
7285 goto 7270
7290 if uz > rz then 7325
7295 s = s + 1
7300 tp$ = nd$(uz)
7305 nd$(uz) = nd$(rz)
7310 nd$(rz) = tp$
7315 uz = uz + 1
7320 rz = rz - 1
7325 if uz = rz or uz < rz then 7250
7330 if uz = ra or uz > ra then 7350
7335 sa = sa + 1
7340 u(sa) = uz
7345 r(sa) = ra
7350 ra = rz
7355 if ua < ra then 7235
7360 if sa > 0 then 7220
7365 rem sort completed!
7370 :
7375 :
7380 rem *--display--*
7385 gosub 10000:rem display subroutine
7390 scnclr
7395 for i = 1 to n
7400 vz = 1:q = 1
7405 na = len(nd$(i))
7410 for va = 1 to na
7415 if mid$(nd$(i),va,2) = "**" then 7425
7420 goto 7440
7425 zi$(q) = mid$(nd$(i),vz,va - vz)
7430 vz = va + 2
7435 q = q + 1
7440 next va
7445 :
7450 zi$(q) = mid$(nd$(i),vz,na - (vz - 1))
7455 :
7460 for e = 1 to q
```

```
7465 print#file,zi$(e)
7470 next e
7475 :
7480 print#file,blank$
7485 next i
7490 goto 20000:rem menu return subroutine
7495 :
7500 :
8000 rem **--return to program menu--**
8005 scnclr
8010 char 0,15,9
8015 color chtr,white
8020 print "LOADING THE MAIL MENU PROGRAM"
8025 color chtr,yellow
8030 run "mail.menu"
8035 :
8040 :
10000 rem **--display subroutine--**
10005 scnclr
10010 char 0,0,3
10015 print "Would you like a paper print out? ";
10020 gosub 18000:rem y/n input subroutine
10025 if yes$ = "y" then 10040:rem printer
10030 if yes$ = "n" then 10115:rem screen
10035 :
10040 rem *--print out is desired--*
10045 scnclr
10050 char 0,0,3
10055 print "Please make sure the printer is on and ready to use."
10060 print
10065 print "Are you ready to begin printing? ";
10070 gosub 18000:rem y/n input subroutine
10075 if yes$ = "n" then 10000:rem ask again
10080 :
10085 rem *--printer display--*
10090 file = 4
10095 dvic = 4
10100 cmnd = 7
10105 goto 10500:rem open instruction
10110 :
10115 rem *--screen display--*
10120 file = 3
10125 dvic = 3
10130 cmnd = 1
10135 goto 10500:rem open instruction
10140 :
10145 :
10500 rem *--open instruction--*
10505 open file,dvic,cmnd
10510 return
10515 :
10520 :
11000 rem **--line up numbers--**
11005 if i < 10 then ta = 4
11010 if i > 9 and i < 100 then ta = 3
11015 if i > 99 and i < 1000 then ta = 2
11020 if i > 999 then ta = 1
11025 return
11030 :
11035 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
```

```
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 print
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print a$;
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
19070 :
20000 rem **--menu return subroutine--**
20005 print#file,""
20010 close file
20015 print
20020 print "Press ";
20025 color chtr,white
20030 print "RETURN";
20035 color chtr,yellow
20040 print " to go to the Display Menu: ";
20045 gosub 19000:rem getkey subroutine
20050 print
20055 goto 500:rem display menu
```

Chapter 7

Correcting Sequential Files

You can use the same beginning for the mail.correction program as you have for both of the mail.reader programs. Once again, the process of creating a new program involves using routines from other programs, modifying those routines, and creating additional routines. The mail.reader2 and mail.adder2 programs will be used to help create the mail.correction program. Remember that if you do not want to follow the steps presented below, you can turn to the end of this chapter and type in the entire program.

I have found that incorporating routines from other programs is a fast way of creating new programs. If you have been careful in the selection of variable names, routines transferred from one program to another do not need to be tested—you already know that they work. You do not need to be concerned about typing mistakes again because those mistakes should have been found already.

Load the mail.reader2 program:

```
dload "mail.reader2 {RETURN}
```

The first thing that needs to be done is to delete unnecessary routines or parts of the program that will not be used in the mail.correction program. Therefore, delete lines 550 to 559, 600 to 615, 1000 to 7999, 10000 to 10999, and 20005 to 20010:

```
delete 550-559 {RETURN}

delete 600-615 {RETURN}

delete 1000-7999 {RETURN}
```

```
delete 10000-10999 {RETURN}
delete 20005-20010 {RETURN}
```

The next step is to change lines. Change the following lines so that they now read:

```
100 rem ***--mail.correction--***
430 dim tlines$(rec)
500 rem **--correction menu--**
520  char 0,24,5, "CORRECTION MENU":print
530 print:print tab(tb)  "1.  CHANGE OR CORRECT INFORMATION"
535 print:print tab(tb)  "2.  DELETE INFORMATION"
540 print:print tab(tb)  "3.  WRITE REVISED FILE"
545 print:print tab(tb)  "4.  RETURN TO PROGRAM MENU"
20040 print " to go to the Correction Menu: ";
```

Third, you need to move two routines to new locations. List lines 625 to 670:

```
list 625-670 {RETURN}
```

Use the up-arrow key to position the cursor on the 6 in line 625. Then change that line number to line 7000 by typing just the number 7000 over the 625 number and the space that comes after it. Do not be concerned that no space exists after the last zero and the letter "r" in "rem." After you have finished, you can list this routine and you will see that the computer automatically places a space after the line number. Once you have entered the number 7000, press the RETURN or ENTER key, and the instruction 625 will be duplicated at line 7000. Change all the other line numbers between 625 and 670 so that they now read 7000 to 7045. Then list the line numbers just to make certain that they are there:

```
list 7000-7045 {RETURN}
```

You should see the following:

```
7000 rem **--incorrect choice message--**
7005 print:print
7010 color chtr,white:print tab(tb) "Incorrect Choice!"
7015 color chtr,yellow:print:print tab(tb) "Press ";
7020 color chtr,white:print "RETURN";
7025 color chtr,yellow:print " to continue:";
7030 gosub 109000:rem getkey subroutine
```

```
7035 goto 500:rem menu--check again
7040 :
7045 :
```

One line needs to be changed. Line 7035 should be **7035 return**. Now the lines from 625 to 650 can be deleted:

```
delete 625-650 {RETURN}
```

Lines 655 needs changing also. It should read:

```
655 gosub 7000:rem incorrect choice message
```

The second routine that needs to be moved is the return-to-program menu routine. List lines 8000-8040:

```
list 8000-8040 {RETURN}
```

Use the up-arrow key to position the cursor on the 8 in line 8000. Then change that line number to line 4000 by typing just the number 4 in place of the 8. Remember to press the RETURN or ENTER key after the change has been made. Change all the line numbers from 8000 to 8040 so that they now go from 4000 to 4040. Then, list the routine:

```
list 4000-4040 {RETURN}
```

and you should see the following listing:

```
4000 rem **--return to program menu--**
4005 scnclr
4010 char 0,15,9
4015 color chtr,white
4020 print "LOADING THE MAIL MENU PROGRAM"
4025 color chtr,yellow
4030 run "mail.menu
4035 :
4040 :
```

Again, this routine now exists in two areas of the program: lines 8000-8040 and 4000 to 4040. Therefore, you need to delete lines 8000 to 8040:

```
delete 8000-8040 {RETURN}
```

All of the deletions and corrections have now been made. Save the work you have done so far:

```
dsave "mail.correction {RETURN}
```

All the subroutines and one of the main menu selections are already included in this

partially completed mail.correction program. The only tasks left are the routines for changing or correcting information, deleting information, and writing the revised file to the diskette. All the instructions for these three routines fall between lines 1000 and 3999.

THE CORRECTION ROUTINE

You can get most of the correction routine from the mail.adder2 program. Make certain that you have saved all that has been done to this point and then load the mail.adder2 program into the computer:

```
dload "mail.adder2 {RETURN}
```

You only need the instructions that exist between lines 700 and 875. Therefore, delete lines 100 to 699 and 1000 to 9999:

```
delete 100-699 {RETURN}
delete 1000-9999 {RETURN}
```

Next, renumber the remaining instruction lines so that they begin at line 1200 and increase by 20. You are incrementing by 20 because there are a number of instructions that must be added between lines in this routine. If you try to immediately renumber, the computer will come back with an error message indicating that you have eliminated three lines that these instructions direct the computer to go to. The easiest way of getting around this problem is to simply include three rem statements and then delete them after the routine has been renumbered. Add these three lines:

```
1000 rem {RETURN}
8000 rem {RETURN}
9000 rem {RETURN}
```

Then renumber the routine so that it begins with line 1200 and each line increases by a count of 20:

```
renumber 1200,20 {RETURN}
```

Next, delete lines 1920 to 1960:

```
delete 1920-1960 {RETURN}
```

Save this routine to the diskette as correct.routine:

```
dsave "correct.routine {RETURN}
```

List the routine up to line 1520:

```
list -1520 {RETURN}
```

Reload the mail.correction program into the computer by using the up-arrow key to position the cursor immediately below instruction line 1520 and then typing:

```
dload "mail.correction {RETURN}
```

Using this procedure, all the instructions remain on the screen. While the instructions are on the screen, use the cursor up-arrow key to go to the 1 in 1200. Press the RETURN key on each of the line numbers between 1200 and 1520. These lines are added to the mail.correction program. Make certain that you do not press the RETURN key when the cursor finally reaches the dload instruction. Pressing RETURN on this instruction would erase the instructions you had just added.

Because the correct.routine is too long to fit on the screen all at one time, the instructions have been split in two parts. You have now added the first half of the correct.routine to the mail.correction program. The new version of this program must be saved before the second part of the correct.routine can be added. Type:

```
dsave "@mail.correction {RETURN}
```

If you check the directory, the mail.correction program should now be up to 13 blocks.

The next step includes the rest of the correct.routine. Load the correct.routine back into the computer:

```
dload "correct.routine {RETURN}
```

List the rest of the routine:

```
list 1540- {RETURN}
```

Reload the mail.correction program into the computer by: using the cursor up-arrow key to position the cursor immediately below instruction line 1900 and then typing:

```
dload "mail.correction {RETURN}
```

While the instructions are on the screen, use the up-arrow key to go to the 1 in 1540. Press the RETURN key on each of the line numbers between 1540 and 1900. Make certain that you do not press the RETURN key when the cursor finally reaches the dload instruction. Then save the mail.correction program once again:

```
dsave "@mail.correction {RETURN}
```

Change the following lines so that they read:

```
1260 print "Do not change the lines with the ";
1320 print "These symbols are used as separators."
1360 for i = nb to j
```

```
1380 print tab(ta) i;" ";tlines$(i)

1460 gosub 18000:rem y/n input subroutine

1560 goto 1000:rem start again

1660 gosub 19000:rem getkey subroutine

1720 if ln > j  then print "Number too large!":goto 1600

1740 if tlines$(ln) = "*" then print "Line";ln;"is the *":goto 1600

1760 print "Line";ln;"now is: "tlines$(ln)

1820 gosub 19000:rem getkey subroutine

1840 tlines$(ln) = t$:t$ = ""
```

Although there are a number of lines that need changing, the changes in most cases are small and certainly easier than retyping the entire line. Before this correction routine is finished, you need to add some lines. Enter the following instructions:

```
1000 rem **--correction--**
1005 scnclr
1010 char 0,5,5
1015 print "Type a ";
1020 color chtr,white
1025 print "0";
1030 color chtr,yellow
1035 print " (zero) when you are finished."
1040 char 0,5,7
1045 print "Display which line? ";
1050 gosub 19000:rem getkey subroutine
1055 nb$ = t$:t$ = ""
1060 nb = val(nb$)

1065 if nb = 0 then print:goto 500 rem menu
1070 if nb > tk then gosub 7000:goto 1000:rem incorrect number
1075 :

1285 color chtr,yellow:print " and ";
1295 color chtr,white:print "!";

1345 j = nb
1350 if tlines$(j) = "!" then 1360
1355 j = j + 1:goto 1350

1365 gosub 11000:rem line up numbers
1375 if i > tk then print tab(ta) "End of File":goto 1400

1710 if ln < nb then print "Number too small!":goto 1600
1730 if ln > tk then print "Number too large!":goto 1600
1750 if tlines$(ln) = "!" then print "Line";ln;"is the !":goto 1600
```

The correction routine should now be operational. The logic is essentially the same

as the logic in the correction routine in the mail.adder2 program. Instructions are displayed to the user, followed by a request for the line number of the information that is to be corrected or changed. The line and its associated information is displayed. If any of the information is not correct, the user is given an opportunity to type in the correct information. The changes made to the mail.adder2 correction routine mainly involve additional error checks, for example, checking for a number smaller than one or checking for the ! separator. Otherwise, the explanation for this correction routine is the same as the explanation for the correction routine given in Chapter 4.

The main difference between the two routines is that when the user indicates that the information is correct, no information is written to the diskette. When the information is correct, the user is taken back to the original request concerning the line number to be displayed. Line 1065 checks for a zero, which indicates that the user wishes to return to the menu. Option three, "Write Revised File," must then be chosen if the changed information is to be permanently added to the file. This procedure gives the user additional time to make certain that the information should be permanently changed, but this procedure might cause problems for some individuals.

Under this system, it is possible to make a number of changes and then forget to save the revised file. If the revised file is not written to the diskette, the changes will not be permanently recorded and subsequent file access will display uncorrected information. Therefore, it is imperative that the revised file be saved if the corrections are to become permanent.

THE DELETION ROUTINE

The deletion routine is more complicated than the correction routine. Enter the following instruction lines:

```
2000 rem **--deletion--**
2005 scnclr
2010 char 0,5,5
2015 print "Type a ";
2020 color chtr,white
2025 print "0";
2030 color chtr,yellow
2035 print " (zero) when you are finished."
2040 char 0,5,7
2045 print "Delete which line? ";
2050 gosub 19000:rem getkey subroutine
2055 nb$ = t$:t$ = ""
2060 nb = val(nb$)
2065 if nb = 0 then print:goto 500 rem menu
2070 if nb > tk then gosub 7000:goto 2000:rem incorrect number
2075 :
2200 rem **--deletion routine--**
2220 scnclr
2240 char 0,0,3
2345 j = nb
2350 if tlines$j) = "!" then 2360
2355 j = j + 1:goto 2350
2360 for i = nb to j
```

```
2365 gosub 11000:rem line up numbers
2375 if i > tk then print tab(ta) "End of File":goto 2400
2380 print tab(ta) i;" ";tlines$(i)
2400 next i
2420 print
2440 print "Are you sure? ";
2460 gosub 18000:rem y/n input subroutine
2480 print
2500 if yes$ = "y" then 2600
2520 if yes$ = "n" then 2560
2540 :
2560 goto 2000:rem start again
2580 :
2600 rem *--delete lines--*
2605 print
2610 print "DELETING THIS INFORMATION!"
2615 for i = nb to j
2620 if i > tk then 2630
2625 tlines$(i) = "DELETED":d = d + 1
2630 next i
2635 :
2640 rem *--renumber without deleted information--*
2645 q = 1
2650 for i 1 to tk
2655 if tlines$(i) = "DELETED" then 2670
2660 tlines$(q) = tlines$(i)
2665 q = q + 1
2670 next i
2675 :
2680 rem *--subtract # of lines deleted for new total--*
2685 tk = tk - d
2690 d = 0:j = 0
2695 print:print
2700 goto 20000:rem menu
2705 :
2710 :
```

There are other ways of doing the same thing that has been done in this routine. Some of the other ways might be shorter, but this way is more understandable. Several things need to be done in this deletion routine. First, the information to be deleted must be identified, displayed, and then deleted (lines 2000 to 2630). Second, the information following the deleted material must be renumbered (instruction lines 2640 to 2670) so that there are no empty information lines, otherwise, an error will occur when these information lines are encountered. Finally, the number of deleted information lines must be subtracted (line 2685) from the original total number of lines.

You have essentially the same beginning as in the correction routine. At line 2345, set a counter (j) equal to the line number of the name to be deleted. Next, increase the counter by one until you have found the information line for the end of the information associated with the individual to be deleted (i.e., the separator symbol !). You know which information lines to delete: the lines beginning with nb and going through j, so now you

can use a loop (lines 2615 to 2630) to delete that information. Use another loop (lines 2650 to 2670) to do the resequencing of the remaining information. Two additional counters are used: q, to keep track of the number of the new information-line-numbers, and d, to keep track of the number of deleted lines. The q is set to one for the beginning of the file, but it could be set to nb, the start of the deleted material. Line 2655 is the key to the resequencing. If tlines$(i) equals the word "DELETED," the counter q is not increased, while the counter i is increased. Remember that q is keeping track of the new line numbers, while i is the old line number. Line 2660 resequences the tlines$ string array. Line 2685 subtracts the number of deleted lines from the original number of lines (tk). Line 2700 is necessary in case more information is to be deleted during this session.

THE WRITE REVISED FILE ROUTINE

```
3000 rem **--write revised file--**
3005 :
3010 rem *--user message--*
3015 scnclr
3020 color chtr,white
3025 char 0,32,9,"UPDATING FILES. PLEASE WAIT!"
3030 color chtr,yellow
3035 :
3040 rem *--delete and rename backup--*
3045 dopen#3,"@Mail Data Backup",w
3050 dclose#3
3055 scratch "Mail Data Backup"
3060 rename  "Mail Data" to "Mail Data Backup"
3065 :
3070 rem *--pointer file--*
3075 dopen#1,"@Mail Pointer",w
3080 print#1,tk
3085 dclose#1
3090 :
3095 rem *--data file--*
3100 dopen#2,"Mail Data",w
3105 for i = 1 to tk
3110 print#2,tlines$(i)
3115 next i
3120 dclose#2
3125 :
3130 rem *--user message & return to menu--*
3135 scnclr
3140 char 0,32,9,"ALL FINISHED!"
3145 char 0,0,22
3150 goto 20000:rem menu subroutine
3155 :
3160 :
```

There is something different about this file routine. Where did Mail Data Backup come from? What is *rename*? The *dopen* command will create a file named Mail Data Backup if the file does not already exist. Therefore, if the file, Mail Data Backup does not already

exist, the command **dopen#3, "@Mail Data Backup",w** will create a file by that name. Next delete that file, because it must either be an empty file or a now unnecessary backup copy. (The first time this program is used there will not be a Mail Data Backup file.) Line 3060 renames the file Mail Data, which now contains uncorrected information so that it becomes Mail Data Backup. Finally, you open a new Mail Data file and write out the corrected information to it (lines 3095-3120). Line 3150 returns to the menu. At this point, if you have not already done so, you should save this program to the diskette that contains all of the other Mailing List System programs:

```
dsave "@mail.correction {RETURN}
```

It is not necessary to make changes in the mail.menu program in order to include this mail.correction program. Remember that the mail.menu program was created in anticipation of this mail.correction program. Therefore all the necessary instructions were included in the mail.menu program when the program was created. The Mailing List System is now complete. A complete list of the final Mailing List System programs, in proper order, is provided at the end of this chapter.

This general method of correcting or deleting information has the added benefit of providing a backup copy of the uncorrected Mail Data file. The sequence of opening and deleting a backup file, renaming the uncorrected file as the new backup, and writing out the corrected information to a new file under the original file name is a very useful routine. If you have two disk drives, you can put the backup in one drive and the new master in the other drive and have the computer switch between the two drives. The more disk drives you have, the greater the flexibility in manipulating files in this manner.

Even without two drives, the Mail Data file and the Mail Data Backup file can be put on two different diskettes. Some method of making the computer pause after line 3060 would be necessary in order to allow the user to swap diskettes. Two possibilities would be a loop of a certain duration, or an input statement informing the user that it is time to switch diskettes.

Although the Mailing List System is complete, the system is by no means a commercial database program. Many additional routines, programs, and features can be added to accomplish specific needs. There are a number of different display possibilities that could be added, as well as many file maintenance routines. For example, it would be a good idea to have a program that would copy the Mail Data file onto a different diskette. A program that allowed the user to expand the information concerning a particular person (i.e., increase from six lines of information to seven or eight lines) would be handy in certain circumstances. Perhaps the most useful additional utility would be one that translated the Mail Data file into a format that a word processor could use to merge with a form letter to produce personalized form letters.

Figure 7-1 shows all the routines presented in this chapter.

In the next chapter, you will take a look at some more techniques for accessing sequential data files.

CHAPTER 7 QUESTIONS

1. True or False: Under the correction method presented in this chapter, corrected information is immediately written to the diskette.

```
 595 if number = 4 then 4000
 620 :
 655 gosub  7000:rem incorrect choice message
 660 goto 500:rem menu--check again
 665 :
 670 :
1000 rem **--correction routine--**
1005 scnclr
1010 char 0,5,5
1015 print "Type a ";
1020 color chtr,white
1025 print "0";
1030 color chtr,yellow
1035 print " (Zero) when you are finished."
1040 char 0,5,7
1045 print "Display which line? ";
1050 gosub 19000:rem getkey subroutine
1055 nb$ = t$:t$ = ""
1060 nb = val(nb$)
1065 if nb = 0 then print:goto 500:rem menu
1070 if nb > tk then gosub 7000:goto 1000:rem incorrect number
1075 :
1200 rem **--correction routine--**
1220 scnclr
1240 char 0,0,3
1260 print "Do not change the lines with the ";
1280 color chtr,white:print "*";
1285 color chtr,yellow:print " and  ";
1295 color chtr,white:print "!";
1300 color chtr,yellow:print ". ";
1320 print "These symbols are used as separators."
1340 print
1345 j = nb
1350 if tlines$(j) = "!" then 1360
1355 j = j + 1:goto 1350
1360 for i = nb to j
1365 gosub 11000:rem line up numbers
1375 if i > tk then print tab(ta) "End of File":goto 1400
1380 print tab(ta) i;" ";tlines$(i)
1400 next i
1420 print
1440 print "Change any line? ";
1460 gosub 18000:rem y/n input subroutine
1480 print
1500 if yes$ = "y" then 1600
1520 if yes$ = "n" then 1560
1540 :
1560 goto 1000:rem start again
1580 :
1600 rem *--change line--*
1620 print
1640 print "Change which line? ";
1660 gosub 19000:rem getkey subroutine
1680 ln = val(t$):t$ = ""
1700 print
1710 if ln < nb then print "Number too small!":goto 1600
1720 if ln > j  then print "Number too large!":goto 1600
1730 if ln > tk then print "Number too large!":goto 1600
1740 if tlines$(ln) = "*" then print "Line";ln;"is the *":goto 1600
1750 if tlines$(ln) = "!" then print "Line";ln;"is the !":goto 1600
1760 print "Line";ln;"now is: ";tlines$(ln)
1780 print
1800 print "Line";ln;"should be: ";
1820 gosub 19000:rem getkey subroutine
1840 tlines$(ln) = t$:t$ = ""
1860 goto 1200:rem correction routine
1880 :
1900 :
```

2. What happens to the original Mail Data file once information in it has been changed?
3. What is the BASIC command used to remove unwanted files?
4. What is the BASIC command used to change the name of files?
5. True or False: Two disk drives are necessary in order to back up a data file.

Fig. 7-1. The Mailing List System programs.

The mail.correction Program

```
100 rem ***--mail.correction--***
105 :
110 :
115 rem **--initialization--**
120 :black =  1
125 :white =  2
130 :::red = 11
135 yellow =  8
140 ::chtr =  5
145 bckgnd =  0
150 :
200 blank$ = chr$(10):rem blank line
205 print chr$(14):rem upper/lower case mode
210 print chr$(8):rem disable char.set change
215 color bckgnd,black
220 :
225 :
300 rem **--user message--**
305 scnclr:color chr,red
310 char 0,32,5
315 print chr$(15);"ONE MOMENT PLEASE!";chr$(143)
320 :
400 rem **--read file routine--**
405 :
410 rem *--read pointer file--*
415 dopen#1,"Mail Pointer"
420 input#1,rec
425 dclose#1
430 dim tlines$(rec)
435 :
440 rem *--data file--*
445 dopen#2,"Mail Data"
450 for i = 1 to rec
455 input#2,tlines$(i)
460 next i
465 dclose#2
470 :
475 tk = rec:rem total k = # of lines
480 :
485 :
500 rem **--correction menu--**
505 tb = 20:rem tab value--20 spaces to the right
510 scnclr
515 color chtr,white
520 char 0,24,5,"CORRECTION MENU":print
525 color chtr,yellow
530 print:print tab(tb) "1.  CHANGE OR CORRECT INFORMATION"
535 print:print tab(tb) "2.  DELETE INFORMATION"
540 print:print tab(tb) "3.  WRITE REVISED FILE"
545 print:print tab(tb) "4.  RETURN TO PROGRAM MENU"
560 print:print tab(tb)
565 input "Which Number Please ";number$
570 number = val(number$)
575 :
580 if number = 1 then 1000
585 if number = 2 then 2000
590 if number = 3 then 3000
```

```
3045 dopen#3,"@Mail Data Backup",w
3050 dclose#3
3055 scratch "Mail Data Backup"
3060 rename  "Mail Data" to "Mail Data Backup"
3065 :
3070 rem *--pointer file--*
3075 dopen#1,"@Mail Pointer",w
3080 print#1,tk
3085 dclose#1
3090 :
3095 rem *--data file--*
3100 dopen#2,"Mail Data",w
3105 for i = 1 to tk
3110 print#2,tlines$(i)
3115 next i
3120 dclose#2
3125 :
3130 rem *--user message & return to menu--*
3135 scnclr
3140 char 0,32,9,"ALL FINISHED!"
3145 char 0,0,22
3150 goto 20000:rem menu subroutine
3155 :
3160 :
4000 rem **--return to program menu--**
4005 scnclr
4010 char 0,15,9
4015 color chtr,white
4020 print "LOADING THE MAIL MENU PROGRAM"
4025 color chtr,yellow
4030 run "mail.menu"
4035 :
4040 :
7000 rem *--incorrect choice message--*
7005 print:print
7010 color chtr,white:print tab(tb) "Incorrect Choice!"
7015 color chtr,yellow:print:print tab(tb) "Press ";
7020 color chtr,white:print "RETURN";
7025 color chtr,yellow:print " to continue:";
7030 gosub 19000:rem getkey subroutine
7035 return
7040 :
7045 :
11000 rem **--line up numbers--**
11005 if i < 10 then ta = 4
11010 if i > 9 and i < 100 then ta = 3
11015 if i > 99 and i < 1000 then ta = 2
11020 if i > 999 then ta = 1
11025 return
11030 :
11035 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 print
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
```

```
2000 rem **--deletion--**
2005 scnclr
2010 char 0,5,5
2015 print "Type a ";
2020 color chtr,white
2025 print "0";
2030 color chtr,yellow
2035 print " (Zero) when you are finished."
2040 char 0,5,7
2045 print "Delete which line? ";
2050 gosub 19000:rem getkey subroutine
2055 nb$ = t$:t$ = ""
2060 nb = val(nb$)
2065 if nb = 0 then print:goto 500:rem menu
2070 if nb > tk then gosub 7000:goto 2000:rem incorrect number
2075 :
2200 rem *--deletion routine--*
2220 scnclr
2240 char 0,0,5
2345 j = nb
2350 if tlines$(j) = "!" then 2360
2355 j = j + 1:goto 2350
2360 for i = nb to j
2365 gosub 11000:rem line up numbers
2375 if i > tk then print tab(ta) "End of File":goto 2400
2380 print tab(ta) i;" ";tline$(i)
2400 next i
2420 print
2440 print "Are you sure? ";
2460 gosub 18000:rem y/n input subroutine
2480 print
2500 if yes$ = "y" then 2600
2520 if yes$ = "n" then 2560
2540 :
2560 goto 2000:rem start again
2580 :
2600 rem *--delete lines--*
2605 print
2610 print "DELETING THIS INFORMATION!"
2615 for i = nb to j
2620 if i > tk then 2630
2625 tlines$(i) = "DELETED":d = d + 1
2630 next i
2635 :
2640 rem *--renumber without deleted information--*
2645 q = 1
2650 for i = 1 to tk
2655 if tlines$(i) = "DELETED" then 2670
2660 tlines$(q) = tlines$(i)
2665 q = q + 1
2670 next i
2675 :
2680 rem *--subtract # of lines deleted for new total--*
2685 tk = tk - d
2690 d = 0:j = 0
2695 print:print
2700 goto 20000:rem menu
2705 :
2710 :
3000 rem **--write revised file--**
3005 :
3010 rem *--user message--*
3015 scnclr
3020 color chtr,white
3025 char 0,32,9,"UPDATING FILES. PLEASE WAIT!"
3030 color chtr,yellow
3035 :
3040 rem *--delete and rename backup--*
```

```
18085 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print a$;
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
19070 :
20000 rem **--menu return subroutine--**
20015 print
20020 print "Press ";
20025 color chtr,white
20030 print "RETURN";
20035 color chtr,yellow
20040 print " to go to the Correction Menu: ";
20045 gosub 19000:rem getkey subroutine
20050 print
20055 goto 500:rem display menu
```

The mail.menu Program

```
  100 rem ***--mail.menu--***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white =  2
  130 :::red = 11
  135 yellow =  8
  140 ::chtr =  5
  145 bckgnd =  0
  150 :
  205 print chr$(14):rem upper/lower case mode
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 :
  225 :
  500 rem **--mail menu--**
  505 tb = 20:rem tab value--20 spaces to the right
  510 scnclr
  515 color chtr,white
  520 char 0,24,2,"MAIL PROGRAM MENU":print
  525 color chtr,yellow
  530 print:print tab(tb) "1.  FILE CREATION PROGRAM"
  535 print:print tab(tb) "2.  FILE ADDITION PROGRAM"
  540 print:print tab(tb) "3.  FILE DISPLAY PROGRAM"
  545 print:print tab(tb) "4.  FILE CORRECTION PROGRAM"
  550 print:print tab(tb) "5.  LIST OF FILES"
  555 print:print tab(tb) "6.  END"
  560 print:print tab(tb)
  565 input "Which Program Number";number$
  570 number = val(number$)
  575 :
  580 if number = 1 then 1000
  585 if number = 2 then 2000
  590 if number = 3 then 3000
  595 if number = 4 then 4000
  600 if number = 5 then 5000
  605 if number = 6 then 6000
  610 :
  615 rem *--incorrect choice message--*
```

```
 620 print:print
 625 color chtr,white:print tab(tb) "Incorrect Choice!"
 630 color chtr,yellow:print:print tab(tb) "Press ";
 635 color chtr,white:print "RETURN";
 640 color chtr,yellow:print " to continue:";
 645 gosub 9000:rem getkey subroutine
 650 goto 500:rem menu--check again
 655 :
 660 :
1000 rem **--file creation program--**
1005 scnclr
1010 char 0,32,5
1015 color chtr,white
1020 print "WARNING!"
1025 print:print
1030 color chtr,yellow
1035 print "If the 'Mail Data' file already exists, do ";
1040 color chtr,red
1045 print "NOT";
1050 color chtr,yellow
1055 print " run this program!"
1060 print
1065 print "Do you want the file creation program? ";
1070 gosub 8000:rem y/n input routine
1080 if yes$ = "n" then 500:rem menu
1085 print:print
1090 print "Are you sure? Type ";
1095 color chtr,white
1100 print "YES";
1105 color chtr,yellow
1110 print " if you are: ";
1115 gosub 9000:rem getkey subroutine
1120 yes$ = t$:t$ = ""
1125 if yes$ = "YES" or yes$ = "yes" then 1135
1130 goto 1000:rem check again
1135 file$ = "MAIL.CREATE"
1140 gosub 7100:rem new program routine after the question
1145 run "mail.create"
1150 :
1155 :
2000 rem **--file addition program--**
2005 file$ = "MAIL.ADDER2"
2010 gosub 7000:rem new program routine
2015 run "mail.adder2"
2020 :
2025 :
3000 rem **--file display program--**
3005 file$ = "MAIL.READER2"
3010 gosub 7000:rem new program routine
3015 run "mail.reader2"
3020 :
3025 :
4000 rem **--file correction program--**
4005 file$ = "MAIL.CORRECTION"
4010 gosub 7000:rem new program routine
4015 run "mail.correction"
4020 :
4025 :
5000 rem **--list of files routine--**
5005 scnclr
5010 directory
5015 print:print
5020 print "Are you ready to return to the menu? ";
5025 gosub 8000:rem y/n input routine
5030 if yes$ = "y" then 500:rem menu
5035 goto 5000:rem check again
5040 :
5045 :
6000 rem **--end routine--**
```

```
6005 scnclr
6010 color chtr,white
6015 char 0,18,9,"That's all for this session! See you next time."
6020 color chtr,yellow
6025 char 0,0,22
6030 end
6035 :
6040 :
7000 rem **--new program routine--**
7005 scnclr:char 0,15,5
7010 print "You have selected the ";
7015 color chtr,red
7020 print file$;
7025 color chtr,yellow
7030 print " program."
7035 char 0,15,9
7040 print "Is this the program you want? ";
7045 gosub 8000:rem y/n input routine
7050 if yes$ = "n" then 500:rem menu
7055 :
7100 rem *--loading message--*
7105 scnclr
7110 char 0,15,5
7115 print "Please wait! I'm loading....";
7120 color chtr,red
7125 print file$
7130 color chtr,yellow
7135 return
7140 :
7145 :
8000 rem **--y/n input subroutine--**
8005 print "Type a ";
8010 color chtr,white:print "Y";
8015 color chtr,yellow:print " or ";
8020 color chtr,white:print "N";
8025 color chtr,yellow:print " :";
8030 gosub 9000:rem getkey subroutine
8035 yes$ = t$:t$ = ""
8040 print
8045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
8050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
8055 print
8060 color chtr,white:print "Incorrect Choice!"
8065 color chtr,yellow
8070 print
8075 goto 8000:rem ask again
8080 :
8085 :
9000 rem **--getkey subroutine--**
9005 getkey a$
9010 if a$ = chr$(20) then 9040:rem delete char.
9015 if a$ = chr$(13) then return:rem return key
9020 print a$;
9025 t$ = t$ + a$
9030 goto 9000
9035 :
9040 lg = len(t$)
9045 if lg < 1 then 9000
9050 print a$;
9055 t$ = left$(t$,lg - 1)
9060 goto 9000
```

The mail.create Program

```
100 rem ***--mail.create--***
105 :
110 :
115 rem **--initialization--**
120 :black =  1
```

```
125 :white =  2
130 :::red = 11
135 yellow =  8
140 ::chtr =  5
145 bckgnd =  0
150 :
200 dim line$(20)
205 print chr$(14):rem upper/lower case mode
210 print chr$(8):rem disable char.set change
215 color bckgnd,black
220 :
225 :
300 rem **--keyboard input routine--**
305 k = 1:rem line counter
310 scnclr
315 char 0,22,2
320 color chtr,white
325 print "INSTRUCTIONS"
330 print
335 color chtr,yellow
340 print "Type the name and address as if you are addressing an envelope."
345 print
350 color chtr,red
355 print chr$(2);:rem underline
360 print "Do not use a comma or a colon!";
365 color chtr,yellow
370 print chr$(130);" ";:rem underline off
375 print "Press ";
380 color chtr,white:print "RETURN";
385 color chtr,yellow:print " after each line."
390 print
395 print "Type the word ";
400 color chtr,white
405 print "END";
410 color chtr,yellow
415 print " on a separate line when you are finished."
420 print:print
425 print "Type in line ";k;":";
430 gosub 9000:rem getkey subroutine
435 if t$ = "end" or t$ = "END" or t$ = "End" then 500
440 if t$ = "" then print:print "We need some information.":goto 425
445 line$(k) = t$:t$ = ""
450 print
455 k = k + 1
460 goto 425:rem next line
465 :
470 :
500 rem **--get phone number--**
505 line$(k) = "*":rem separator for phone number
510 t$ = ""
515 print:print
520 k = k + 1
525 print "Press ";
530 color chtr,white:print "RETURN";
535 color chtr,yellow:print " if there is no phone #."
540 print
545 print "Type in Phone #: ";
550 gosub 9000:rem getkey subroutine
555 line$(k) = t$:t$ = ""
560 if line$(k) = "" then line$(k) = "none"
565 k = k + 1
570 line$(k) = "!":rem separator between sets of information
575 :
580 :
700 rem **--correction routine--**
705 scnclr
710 char 0,0,3
715 print "Do not change the line with the ";
720 color chtr,white:print "*";
```

```
  725 color chtr,yellow:print ". ";
  730 print "This symbol is used as a separator."
  735 print
  740 for i = 1 to k - 1
  745 print i;" ";line$(i)
  750 next i
  755 print
  760 print "Change any line? ";
  765 gosub 8000:rem y/n input subroutine
  770 print
  775 if yes$ = "y" then 800
  780 if yes$ = "n" then 790
  785 :
  790 goto 3000:rem file creation subroutine
  795 :
  800 rem *--change line--*
  805 print
  810 print "Change which line? ";
  815 gosub 9000:rem getkey subroutine
  820 ln = val(t$):t$ = ""
  825 print
  830 if ln > k - 1 then print "Number too large!":goto 800
  835 if line$(ln) = "*" then print "Line";ln;"is the *":goto 800
  840 print "Line";ln;"now is: ";line$(ln)
  845 print
  850 print "Line";ln;"should be: ";
  855 gosub 9000:rem getkey subroutine
  860 line$(ln) = t$:t$ = ""
  865 goto 700:rem correction routine
  870 :
  875 :
 3000 rem **--file creation subroutine--**
 3005 :
 3200 rem *--data file--*
 3205 dopen#2,"Mail Data",w
 3210 for i = 1 to k
 3215 print#2,line$(i)
 3220 next i
 3225 dclose#2
 3230 :
 3300 rem *--pointer file--*
 3305 dopen#1,"Mail Pointer",w
 3310 print#1,k
 3315 dclose#1
 3320 :
 3325 :
 5000 rem **--return to program menu--**
 5005 scnclr
 5010 char 0,15,9
 5015 color chtr,white
 5020 print "LOADING THE MAIL MENU PROGRAM"
 5025 color chtr,yellow
 5030 run "mail.menu"
 5035 :
 5040 :
 8000 rem **--y/n input subroutine--**
 8005 print "Type a ";
 8010 color chtr,white:print "Y";
 8015 color chtr,yellow:print " or ";
 8020 color chtr,white:print "N";
 8025 color chtr,yellow:print " :";
 8030 gosub 9000:rem getkey subroutine
 8035 yes$ = t$:t$ = ""
 8040 print
 8045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
 8050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
 8055 print
 8060 color chtr,white:print "Incorrect Choice!"
 8065 color chtr,yellow
```

```
8070 print
8075 goto 8000:rem ask again
8080 :
8085 :
9000 rem **--getkey subroutine--**
9005 getkey a$
9010 if a$ = chr$(20) then 9040:rem delete char.
9015 if a$ = chr$(13) then return:rem return key
9020 print a$;
9025 t$ = t$ + a$
9030 goto 9000
9035 :
9040 lg = len(t$)
9045 if lg < 1 then 9000
9050 print a$;
9055 t$ = left$(t$,lg - 1)
9060 goto 9000
```

The mail.adder2 Program

```
100 rem ***--mail.adder2--***
105 :
110 :
115 rem **--initialization--**
120 :black =  1
125 :white =  2
130 :::red = 11
135 yellow =  8
140 ::chtr =  5
145 bckgnd =  0
150 :
200 dim line$(20),tlines$(100)
205 print chr$(14):rem upper/lower case mode
210 print chr$(8):rem disable char.set change
215 color bckgnd,black
220 :
225 :
300 rem **--keyboard input routine--**
305 k = 1:rem line counter
310 scnclr
315 char 0,22,2
320 color chtr,white
325 print "INSTRUCTIONS"
330 print
335 color chtr,yellow
340 print "Type the name and address as if you are addressing an envelope."
345 print
350 color chtr,red
355 print chr$(2);:rem underline
360 print "Do not use a comma or a colon!";
365 color chtr,yellow
370 print chr$(130);" ";:rem underline off
375 print "Press ";
380 color chtr,white:print "RETURN";
385 color chtr,yellow:print " after each line."
390 print
395 print "Type the word ";
400 color chtr,white
405 print "END";
410 color chtr,yellow
415 print " on a separate line when you are finished."
420 print:print
425 print "Type in line ";k;":";
430 gosub 9000:rem getkey subroutine
435 if t$ = "end" or t$ = "END" or t$ = "End" then 500
440 if t$ = "" then print:print "We need some information.":goto 425
445 line$(k) = t$:t$ = ""
450 print
455 k = k + 1
```

```
460 goto 425:rem next line
465 :
470 :
500 rem **--get phone number--**
505 line$(k) = "*":rem separator for phone number
510 t$ = ""
515 print:print
520 k = k + 1
525 print "Press ";
530 color chtr,white:print "RETURN";
535 color chtr,yellow:print " if there is no phone #."
540 print
545 print "Type in Phone #: ";
550 gosub 9000:rem getkey subroutine
555 line$(k) = t$:t$ = ""
560 if line$(k) = "" then line$(k) = "none"
565 k = k + 1
570 line$(k) = "↓":rem separator between sets of information
575 :
580 :
700 rem **--correction routine--**
705 scnclr
710 char 0,0,3
715 print "Do not change the line with the ";
720 color chtr,white:print "*";
725 color chtr,yellow:print ". ";
730 print "This symbol is used as a separator."
735 print
740 for i = 1 to k - 1
745 print i;" ";line$(i)
750 next i
755 print
760 print "Change any line? ";
765 gosub 8000:rem y/n input subroutine
770 print
775 if yes$ = "y" then 800
780 if yes$ = "n" then 790
785 :
790 goto 3000:rem file creation subroutine
795 :
800 rem *--change line--*
805 print
810 print "Change which line? ";
815 gosub 9000:rem getkey subroutine
820 ln = val(t$):t$ = ""
825 print
830 if ln > k - 1 then print "Number too large!":goto 800
835 if line$(ln) = "*" then print "Line";ln;"is the *":goto 800
840 print "Line";ln;"now is: ";line$(ln)
845 print
850 print "Line";ln;"should be: ";
855 gosub 9000:rem getkey subroutine
860 line$(ln) = t$:t$ = ""
865 goto 700:rem correction routine
870 :
875 :
1000 rem **--print label subroutine--**
1005 scnclr
1010 char 0,0,3
1015 print "Would you like to print a mailing label now? ";
1020 gosub 8000:rem y/n input subroutine
1025 if yes$ = "y" then 1100
1030 if yes$ = "n" then 2000:rem repeat subroutine
1035 :
1100 rem *--mailing label is desired--*
1105 scnclr
1110 char 0,0,3
1115 print "Please make sure the printer is on and ready to use."
1120 print
```

```
1125 print "Are you ready to begin printing? ";
1130 gosub 8000:rem y/n input subroutine
1135 if yes$ = "n" then 1000:rem ask again
1140 :
1145 rem *--printer channel--*
1150 open 4,4,7
1155 :
1160 rem first 4 = file #
1165 rem second 4 = printer device #
1170 rem 7 = command for upper/lower case
1175 :
1180 for i = 1 to k
1185 if line$(i) = "*" then i = i + 1:goto 1200
1190 if line$(i) = "!" then 1200
1195 print#4,line$(i)
1200 next i
1205 print#4," "
1210 close 4
1215 :
1220 :
2000 rem **--repeat routine--**
2005 :
2010 rem *--add new lines to existing file lines--*
2015 for i = 1 to k
2020 tlines$(tk + i) = line$(i):rem tk + i, not tk + 1
2025 next i
2030 tk = tk + k
2035 :
2080 scnclr
2085 char 0,0,3
2090 print "Do you want to add more information? ";
2095 gosub 8000:rem y/n input routine
2100 if yes$ = "y" then 300:rem begin again
2105 if yes$ = "n" then 3000:rem file addition subroutine
2110 :
2115 :
3000 rem **--file addition subroutine--**
3005 :
3100 rem *--read pointer file--*
3105 dopen#1,"Mail Pointer"
3110 input#1,rec
3115 dclose#1
3120 rec = rec + tk
3125 :
3200 rem *--data file--*
3205 append#2,"Mail Data"
3210 for i = 1 to tk
3215 print#2,tline$(i)
3220 next i
3225 dclose#2
3230 :
3300 rem *--write pointer file--*
3305 dopen#1,"@Mail Pointer",w
3310 print#1,rec
3315 dclose#1
3320 :
3325 :
5000 rem **--return to program menu--**
5005 scnclr
5010 char 0,15,9
5015 color chtr,white
5020 print "LOADING THE MAIL MENU PROGRAM"
5025 color chtr,yellow
5030 run "mail.menu"
5035 :
5040 :
8000 rem **--y/n input subroutine--**
8005 print "Type a ";
8010 color chtr,white:print "Y";
```

```
8015 color chtr,yellow:print " or ";
8020 color chtr,white:print "N";
8025 color chtr,yellow:print " :";
8030 gosub 9000:rem getkey subroutine
8035 yes$ = t$:t$ = ""
8040 print
8045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
8050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
8055 print
8060 color chtr,white:print "Incorrect Choice!"
8065 color chtr,yellow
8070 print
8075 goto 8000:rem ask again
8080 :
8085 :
9000 rem **--getkey subroutine--**
9005 getkey a$
9010 if a$ = chr$(20) then 9040:rem delete char.
9015 if a$ = chr$(13) then return:rem return key
9020 print a$;
9025 t$ = t$ + a$
9030 goto 9000
9035 :
9040 lg = len(t$)
9045 if lg < 1 then 9000
9050 print a$;
9055 t$ = left$(t$,lg - 1)
9060 goto 9000
```

The mail.reader2 Program

```
100 rem ***--mail.reader2--***
105 :
110 :
115 rem **--initialization--**
120 :black =  1
125 :white =  2
130 :::red = 11
135 yellow =  8
140 ::chtr =  5
145 bckgnd =  0
150 :
200 blank$ = chr$(10):rem blank line
205 print chr$(14):rem upper/lower case mode
210 print chr$(8):rem disable char.set change
215 color bckgnd,black
220 :
225 :
300 rem **--user message--**
305 scnclr:color chr,red
310 char 0,32,5
315 print chr$(15);"ONE MOMENT PLEASE!";chr$(143)
320 :
400 rem **--read file routine--**
405 :
410 rem *--read pointer file--*
415 dopen#1,"Mail Pointer"
420 input#1,rec
425 dclose#1
430 dim tlines$(rec),nd$(rec),u(rec),r(rec),ad$(rec),zi$(rec)
435 :
440 rem *--data file--*
445 dopen#2,"Mail Data"
450 for i = 1 to rec
455 input#2,tlines$(i)
460 next i
465 dclose#2
470 :
475 tk = rec:rem total k = # of lines
```

```
480 :
485 :
500 rem **--reader2 menu--**
505 tb = 20:rem tab value--20 spaces to the right
510 scnclr
515 color chtr,white
520 char 0,24,1,"DISPLAY MENU":print
525 color chtr,yellow
530 print:print tab(tb) "1.  INFORMATION--ORIGINAL ORDER"
535 print:print tab(tb) "2.  NAMES ONLY"
540 print:print tab(tb) "3.  INFORMATION--NO PHONE"
545 print:print tab(tb) "4.  SPECIFIC NAME"
550 print:print tab(tb) "5.  SPECIFIC NAME--NO PHONE"
555 print:print tab(tb) "6.  INFORMATION--RANGE"
557 print:print tab(tb) "7.  INFORMATION--ALPHABETICAL"
559 print:print tab(tb) "8.  RETURN TO PROGRAM MENU"
560 print:print tab(tb)
565 input "Which Number Please ";number$
570 number = val(number$)
575 :
580 if number = 1 then 1000
585 if number = 2 then 2000
590 if number = 3 then 3000
595 if number = 4 then 4000
600 if number = 5 then 5000
605 if number = 6 then 6000
610 if number = 7 then 7000
615 if number = 8 then 8000
620 :
625 rem *--incorrect choice message--*
630 print:print
635 color chtr,white:print tab(tb) "Incorrect Choice!"
640 color chtr,yellow:print:print tab(tb) "Press ";
645 color chtr,white:print "RETURN";
650 color chtr,yellow:print " to continue:";
655 gosub 19000:rem getkey subroutine
660 goto 500:rem menu--check again
665 :
670 :
1000 rem **--original order routine--**
1005 gosub 10000:rem display subroutine
1010 scnclr
1015 for i = 1 to tk
1020 if tlines$(i) = "*" then 1040
1025 if tlines$(i) = "↓" then print#file,blank$:goto 1040
1030 gosub 11000:rem line up numbers
1035 print#file,tab(ta);i;" ";tlines$(i)
1040 next i
1045 goto 20000:rem menu return subroutine
1050 :
1055 :
2000 rem **--name only routine--**
2005 gosub 10000:rem display subroutine
2010 scnclr
2015 for i = 1 to tk - 1
2020 if tlines$(i) = tlines$(1) then 2040
2025 if tlines$(i) = "↓" then 2040
2030 goto 2055
2035 :
2040 gosub 11000:rem line up numbers
2045 if tlines$(i) = tlines$(1) then print#file,tab(ta) i;" ";tlines$(1)
2050 if tlines$(i) = "↓" then print#file,tab(ta) i + 1;" ";tlines$(i + 1)
2055 next i
2060 goto 20000:rem menu return subroutine
2065 :
2070 :
3000 rem **--no phone routine--**
3005 gosub 10000:rem display subroutine
3010 scnclr
```

```
3015 for i = 1 to tk
3020 gosub 11000:rem line up numbers
3025 if tlines$(i) = "*" then i = i + 1:goto 3040
3030 if tlines$(i) = "¦" then print#file,blank$:goto 3040
3035 print#file,tlines$(i)
3040 next i
3045 goto 20000:rem menu return subroutine
3050 :
3055 :
4000 rem **--search routine--**
4005 gosub 10000:rem display subroutine
4010 scnclr
4015 print "Type the word ";
4020 color chtr,white
4025 print "END";
4030 color chtr,yellow
4035 print " when finished."
4040 print
4045 print "Name to find: ";
4050 gosub 19000:rem getkey subroutine
4055 find$ = t$:t$ = ""
4060 if find$ = "END" or find$ = "end" then 4155
4065 print
4070 for i = 1 to tk
4075 if tlines$(i) = find$ then 4085
4080 goto 4140
4085 if tlines$(i) = "*" then 4140
4090 if tlines$(i) = "¦" then 4140
4095 gosub 11000:rem line up numbers
4100 print#file,blank$
4105 print#file,tab(ta) i;" ";tlines$(i)
4110 print#file,tab(ta + (8 - ta)) tlines$(i + 1)
4115 print#file,tab(ta + (8 - ta)) tlines$(i + 2)
4120 if tlines$(i + 3) <> "*" then print#file,tab(ta + (8 - ta)) tlines$(i + 3)
4125 if tlines$(i + 4) = "*" then 4135
4130 print#file,tab(ta + (8 - ta)) tlines$(i + 4):goto 4140
4135 print#file,tab(ta + (8 - ta)) tlines$(i + 5)
4140 next i
4145 print
4150 goto 4015:rem repeat until done
4155 goto 20000:rem menu return subroutine
4160 :
4165 :
5000 rem **--search routine/no phone--**
5005 gosub 10000:rem display subroutine
5010 scnclr
5015 print "Type the word ";
5020 color chtr,white
5025 print "END";
5030 color chtr,yellow
5035 print " when finished."
5040 print
5045 print "Name to find: ";
5050 gosub 19000:rem getkey subroutine
5055 find$ = t$:t$ = ""
5060 if find$ = "END" or find$ = "end" then 5160
5065 print
5070 for i = 1 to tk
5075 if tlines$(i) = find$ then 5085
5080 goto 5145
5085 if tlines$(i) = "*" then i = i + 1:goto 5145
5090 if tlines$(i) = "¦" then print#file,blank$:goto 5145
5095 gosub 11000:rem line up numbers
5100 print#file,blank$
5105 print#file,tab(ta) i;" ";tlines$(i)
5110 print#file,tab(ta + (8 - ta)) tlines$(i + 1)
5115 print#file,tab(ta + (8 - ta)) tlines$(i + 2)
5120 if tlines$(i + 3) <> "*" then print#file,tab(ta + (8 - ta)) tlines$(i + 3)
```

```
5125 if tlines$(i + 3) = "*" then i = i + 1:goto 5145
5130 if tlines$(i + 4) = "*" then i = i + 1:goto 5145
5135 print#file,tab(ta + (8 - ta)) tlines$(i + 4):goto 5145
5140 print#file,tab(ta + (8 - ta)) tlines$(i + 5)
5145 next i
5150 print
5155 goto 5015:rem repeat until done
5160 goto 20000:rem menu return subroutine
5165 :
5170 :
6000 rem **--range routine--**
6005 gosub 10000:rem display subroutine
6010 scnclr
6015 char 0,0,5
6020 input "Type the beginning line number please: ";bg
6025 print:print
6030 if bg < 1 then print "The number is too small! Please try again.":goto 6020
6035 input "Type the ending line number please: ";ed
6040 print
6045 if ed > tk then print "The number is too large! Please try again.":goto 6025
6050 for i = bg to ed
6055 gosub 11000:rem line up numbers
6060 if tlines$(i) = "*" then i = i + 1: goto 6075
6065 if tlines$(i) = "!" then print#file,blank$:goto 6075
6070 print#file,tab(ta) i;" ";tlines$(i)
6075 next i
6080 goto 20000:rem menu return subroutine
6085 :
6090 :
7000 rem **--alphabetical order--**
7005 scnclr
7010 char 0,15,9
7015 color chtr,white
7020 print "WORKING--PLEASE DON'T TOUCH!!"
7025 color chtr,yellow
7030 :
7035 :
7040 rem *--get first info.line--*
7045 for i = 1 to tk - 1
7050 if tlines$(i) = tlines$(1) then 7075
7055 if tlines$(i) = "!" then i = i + 1:goto 7075
7060 goto 7135:rem next i
7065 :
7070 :
7075 rem *--reverse order--*
7080 na = len(tlines$(i))
7085 for va = 1 to na
7090 if asc(mid$(tlines$(i),va,1)) = 32 then vz = va
7095 next va
7100 if vz = 0 or vz > na then ad$(i) = tlines$(i):goto 7110
7105 ad$(i) = mid$(tlines$(i),vz + 1,na - vz) + ", " + left$(tlines$(i),vz)
7110 ad$(i) = ad$(i) + "**" + tlines$(i + 1) + "**" + tlines$(i + 2)
7115 if tlines$(i + 3) <> "*" then ad$(i) = ad$(i) + "**" + tlines$(i + 3)
7120 if tlines$(i + 4) = "*" then 7130
7125 ad$(i) = ad$(i) + "**" + tlines$(i + 4):goto 7135
7130 ad$(i) = ad$(i) + "**" + tlines$(i + 5)
7135 next i
7140 :
7145 :
7150 rem *--renumber for sort--*
7155 j = 1
7160 for i = 1 to tk
7165 if len(ad$(i)) > 0 then nd$(j) = ad$(i): j = j + 1
7170 next i
7175 n = j - 1
7180 :
7185 :
7190 rem *--quicksort--*
```

```
7195 sa = 1
7200 char 0,15,11
7205 print "STILL WORKING--PLEASE DON'T TOUCH!!"
7210 u(1) = 1
7215 r(1) = n
7220 ua = u(sa)
7225 ra = r(sa)
7230 sa = sa - 1
7235 uz = ua
7240 rz = ra
7245 x$ = nd$(int((ua + ra)/2))
7250 c = c + 1
7255 if nd$(uz) = x$ or nd$(uz) > x$ then 7270
7260 uz = uz + 1
7265 goto 7250
7270 c = ca
7275 if x$ = nd$(rz) or x$ > nd$(rz) then 7290
7280 rz = rz - 1
7285 goto 7270
7290 if uz > rz then 7325
7295 s = s + 1
7300 tp$ = nd$(uz)
7305 nd$(uz) = nd$(rz)
7310 nd$(rz) = tp$
7315 uz = uz + 1
7320 rz = rz - 1
7325 if uz = rz or uz < rz then 7250
7330 if uz = ra or uz > ra then 7350
7335 sa = sa + 1
7340 u(sa) = uz
7345 r(sa) = ra
7350 ra = rz
7355 if ua < ra then 7235
7360 if sa > 0 then 7220
7365 rem sort completed!
7370 :
7375 :
7380 rem *--display--*
7385 gosub 10000:rem display subroutine
7390 scnclr
7395 for i = 1 to n
7400 vz = 1:q = 1
7405 na = len(nd$(i))
7410 for va = 1 to na
7415 if mid$(nd$(i),va,2) = "**" then 7425
7420 goto 7440
7425 zi$(q) = mid$(nd$(i),vz,va - vz)
7430 vz = va + 2
7435 q = q + 1
7440 next va
7445 :
7450 zi$(q) = mid$(nd$(i),vz,na - (vz - 1))
7455 :
7460 for e = 1 to q
7465 print#file,zi$(e)
7470 next e
7475 :
7480 print#file,blank$
7485 next i
7490 goto 20000:rem menu return subroutine
7495 :
7500 :
8000 rem **--return to program menu--**
8005 scnclr
8010 char 0,15,9
8015 color chtr,white
8020 print "LOADING THE MAIL MENU PROGRAM"
8025 color chtr,yellow
```

```
8030 run "mail.menu"
8035 :
8040 :
10000 rem **--display subroutine--**
10005 scnclr
10010 char 0,0,3
10015 print "Would you like a paper print out? ";
10020 gosub 18000:rem y/n input subroutine
10025 if yes$ = "y" then 10040:rem printer
10030 if yes$ = "n" then 10115:rem screen
10035 :
10040 rem *--print out is desired--*
10045 scnclr
10050 char 0,0,3
10055 print "Please make sure the printer is on and ready to use."
10060 print
10065 print "Are you ready to begin printing? ";
10070 gosub 18000:rem y/n input subroutine
10075 if yes$ = "n" then 10000:rem ask again
10080 :
10085 rem *--printer display--*
10090 file = 4
10095 dvic = 4
10100 cmnd = 7
10105 goto 10500:rem open instruction
10110 :
10115 rem *--screen display--*
10120 file = 3
10125 dvic = 3
10130 cmnd = 1
10135 goto 10500:rem open instruction
10140 :
10145 :
10500 rem *--open instruction--*
10505 open file,dvic,cmnd
10510 return
10515 :
10520 :
11000 rem **--line up numbers--**
11005 if i < 10 then ta = 4
11010 if i > 9 and i < 100 then ta = 3
11015 if i > 99 and i < 1000 then ta = 2
11020 if i > 999 then ta = 1
11025 return
11030 :
11035 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 print
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
```

```
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print a$;
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
19070 :
20000 rem **--menu return subroutine--**
20005 print#file,""
20010 close file
20015 print
20020 print "Press ";
20025 color chtr,white
20030 print "RETURN";
20035 color chtr,yellow
20040 print " to go to the Display Menu: ";
20045 gosub 19000:rem getkey subroutine
20050 print
20055 goto 500:rem display menu
```

The mail.correction Program (Second Version)

```
100 rem ***--mail.correction--***
105 :
110 :
115 rem **--initialization--**
120 :black =  1
125 :white =  2
130 :::red = 11
135 yellow =  8
140 ::chtr =  5
145 bckgnd =  0
150 :
200 blank$ = chr$(10):rem blank line
205 print chr$(14):rem upper/lower case mode
210 print chr$(8):rem disable char.set change
215 color bckgnd,black
220 :
225 :
300 rem **--user message--**
305 scnclr:color chr,red
310 char 0,32,5
315 print chr$(15);"ONE MOMENT PLEASE!";chr$(143)
320 :
400 rem **--read file routine--**
405 :
410 rem *--read pointer file--*
415 dopen#1,"Mail Pointer"
420 input#1,rec
425 dclose#1
430 dim tlines$(rec)
435 :
440 rem *--data file--*
445 dopen#2,"Mail Data"
450 for i = 1 to rec
455 input#2,tlines$(i)
460 next i
465 dclose#2
470 :
475 tk = rec:rem total k = # of lines
480 :
485 :
500 rem **--correction menu--**
505 tb = 20:rem tab value--20 spaces to the right
510 scnclr
515 color chtr,white
```

```
 520 char 0,24,5,"CORRECTION MENU":print
 525 color chtr,yellow
 530 print:print tab(tb) "1.  CHANGE OR CORRECT INFORMATION"
 535 print:print tab(tb) "2.  DELETE INFORMATION"
 540 print:print tab(tb) "3.  WRITE REVISED FILE"
 545 print:print tab(tb) "4.  RETURN TO PROGRAM MENU"
 560 print:print tab(tb)
 565 input "Which Number Please ";number$
 570 number = val(number$)
 575 :
 580 if number = 1 then 1000
 585 if number = 2 then 2000
 590 if number = 3 then 3000
 595 if number = 4 then 4000
 620 :
 655 gosub  7000:rem incorrect choice message
 660 goto 500:rem menu--check again
 665 :
 670 :
1000 rem **--correction routine--**
1005 scnclr
1010 char 0,5,5
1015 print "Type a ";
1020 color chtr,white
1025 print "0";
1030 color chtr,yellow
1035 print " (Zero) when you are finished."
1040 char 0,5,7
1045 print "Display which line? ";
1050 gosub 19000:rem getkey subroutine
1055 nb$ = t$:t$ = ""
1060 nb = val(nb$)
1065 if nb = 0 then print:goto 500:rem menu
1070 if nb > tk then gosub 7000:goto 1000:rem incorrect number
1075 :
1200 rem **--correction routine--**
1220 scnclr
1240 char 0,0,3
1260 print "Do not change the lines with the ";
1280 color chtr,white:print "*";
1285 color chtr,yellow:print " and  ";
1295 color chtr,white:print "¡";
1300 color chtr,yellow:print ". ";
1320 print "These symbols are used as separators."
1340 print
1345 j = nb
1350 if tlines$(j) = "¡" then 1360
1355 j = j + 1:goto 1350
1360 for i = nb to j
1365 gosub 11000:rem line up numbers
1375 if i > tk then print tab(ta) "End of File":goto 1400
1380 print tab(ta) i;" ";tlines$(i)
1400 next i
1420 print
1440 print "Change any line? ";
1460 gosub 18000:rem y/n input subroutine
1480 print
1500 if yes$ = "y" then 1600
1520 if yes$ = "n" then 1560
1540 :
1560 goto 1000:rem start again
1580 :
1600 rem *--change line--*
1620 print
1640 print "Change which line? ";
1660 gosub 19000:rem getkey subroutine
1680 ln = val(t$):t$ = ""
1700 print
1710 if ln < nb then print "Number too small¡":goto 1600
```

```
1720 if ln > j  then print "Number too large!":goto 1600
1730 if ln > tk then print "Number too large!":goto 1600
1740 if tlines$(ln) = "*" then print "Line";ln;"is the *":goto 1600
1750 if tlines$(ln) = "!" then print "Line";ln;"is the !":goto 1600
1760 print "Line";ln;"now is: ";tlines$(ln)
1780 print
1800 print "Line";ln;"should be: ";
1820 gosub 19000:rem getkey subroutine
1840 tlines$(ln) = t$:t$ = ""
1860 goto 1200:rem correction routine
1880 :
1900 :
2000 rem **--deletion--**
2005 scnclr
2010 char 0,5,5
2015 print "Type a ";
2020 color chtr,white
2025 print "0";
2030 color chtr,yellow
2035 print " (Zero) when you are finished."
2040 char 0,5,7
2045 print "Delete which line? ";
2050 gosub 19000:rem getkey subroutine
2055 nb$ = t$:t$ = ""
2060 nb = val(nb$)
2065 if nb = 0 then print:goto 500:rem menu
2070 if nb > tk then gosub 7000:goto 2000:rem incorrect number
2075 :
2200 rem *--deletion routine--*
2220 scnclr
2240 char 0,0,5
2345 j = nb
2350 if tlines$(j) = "!" then 2360
2355 j = j + 1:goto 2350
2360 for i = nb to j
2365 gosub 11000:rem line up numbers
2375 if i > tk then print tab(ta) "End of File":goto 2400
2380 print tab(ta) i;" ";tline$(i)
2400 next i
2420 print
2440 print "Are you sure? ";
2460 gosub 18000:rem y/n input subroutine
2480 print
2500 if yes$ = "y" then 2600
2520 if yes$ = "n" then 2560
2540 :
2560 goto 2000:rem start again
2580 :
2600 rem *--delete lines--*
2605 print
2610 print "DELETING THIS INFORMATION!"
2615 for i = nb to j
2620 if i > tk then 2630
2625 tlines$(i) = "DELETED":d = d + 1
2630 next i
2635 :
2640 rem *--renumber without deleted information--*
2645 q = 1
2650 for i = 1 to tk
2655 if tlines$(i) = "DELETED" then 2670
2660 tlines$(q) = tlines$(i)
2665 q = q + 1
2670 next i
2675 :
2680 rem *--subtract # of lines deleted for new total--*
2685 tk = tk - d
2690 d = 0:j = 0
2695 print:print
```

```
2700 goto 20000:rem menu
2705 :
2710 :
3000 rem **--write revised file--**
3005 :
3010 rem *--user message--*
3015 scnclr
3020 color chtr,white
3025 char 0,32,9,"UPDATING FILES. PLEASE WAIT!"
3030 color chtr,yellow
3035 :
3040 rem *--delete and rename backup--*
3045 dopen#3,"@Mail Data Backup",w
3050 dclose#3
3055 scratch "Mail Data Backup"
3060 rename  "Mail Data" to "Mail Data Backup"
3065 :
3070 rem *--pointer file--*
3075 dopen#1,"@Mail Pointer",w
3080 print#1,tk
3085 dclose#1
3090 :
3095 rem *--data file--*
3100 dopen#2,"Mail Data",w
3105 for i = 1 to tk
3110 print#2,tlines$(i)
3115 next i
3120 dclose#2
3125 :
3130 rem *--user message & return to menu--*
3135 scnclr
3140 char 0,32,9,"ALL FINISHED!"
3145 char 0,0,22
3150 goto 20000:rem menu subroutine
3155 :
3160 :
4000 rem **--return to program menu--**
4005 scnclr
4010 char 0,15,9
4015 color chtr,white
4020 print "LOADING THE MAIL MENU PROGRAM"
4025 color chtr,yellow
4030 run "mail.menu"
4035 :
4040 :
7000 rem *--incorrect choice message--*
7005 print:print
7010 color chtr,white:print tab(tb) "Incorrect Choice!"
7015 color chtr,yellow:print:print tab(tb) "Press ";
7020 color chtr,white:print "RETURN";
7025 color chtr,yellow:print " to continue:";
7030 gosub 19000:rem getkey subroutine
7035 return
7040 :
7045 :
11000 rem **--line up numbers--**
11005 if i < 10 then ta = 4
11010 if i > 9 and i < 100 then ta = 3
11015 if i > 99 and i < 1000 then ta = 2
11020 if i > 999 then ta = 1
11025 return
11030 :
11035 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
```

```
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 print
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print a$;
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
19070 :
20000 rem **--menu return subroutine--**
20015 print
20020 print "Press ";
20025 color chtr,white
20030 print "RETURN";
20035 color chtr,yellow
20040 print " to go to the Correction Menu: ";
20045 gosub 19000:rem getkey subroutine
20050 print
20055 goto 500:rem display menu
```

Chapter 8

Additional Sequential File Techniques

Some other ways to work with sequential files and additional techniques for file handling are explored in this chapter. I am going to concentrate on the file routines of the various programs presented, and not discuss the rest of the programming. The listings for the complete programs are included at the end of this chapter, along with an explanation of the new commands used in these programs.

The first series of programs allows an individual to practice math and keep a record of the scores achieved. These programs are essentially drill and practice and might not be the best educational use of the computer, but for the purpose of demonstrating how files can be used in a variety of ways, they will be sufficient.

Again, start with careful thought and preparation. You need a separate program for each mathematical operation, along with a program for the scores. This means that another program menu would be convenient. The essential difference between the operation programs is the sign of the operation: + for addition, × for multiplication, and so on. With the exception of division, the numbers can be displayed in basically the same way. Therefore, the same basic program can be used, with changes made for the different operations.

Once you have typed in the math.add program (Fig. 8-1 at the end of the chapter), changes need to be made to six instructions in order to create the math.subtract program. In lines 100, 425, 435, 605, 610, and 615, the references to addition should be changed to subtraction. After the math.add program has been typed in and saved to diskette, the procedure to make the necessary changes is as follows:

1. Load the math.add program:

```
dload "math.add" {RETURN}
```

2. List the line to be changed:

```
list 100 {RETURN}

100 rem ***--addition--***
```

3. Make the necessary change:

```
100 rem ***--subtraction--***
```

4. List the next line and make the changes.
5. Repeat these steps for each of the six instructions.

The six instruction lines with their changes follow:

Line 100: Change the word "addition" to "subtraction"
Line 425: Change the word "adding" to "subtracting"
Line 435: Change the word "adding" to "subtracting"
Line 605: Change the sign from "+" to "−"
Line 610: Change the sign from "+" to "−"
Line 615: Change the word "addition" to "subtraction"

Once all the changes have been made, save the new program:

```
dsave "math.subtract" {RETURN}
```

The programs for multiplication and division have additional code, because the numbers must be formatted differently. All the programs have been structured so that the answers can be entered as if the problem were being done on paper; i.e., for the addition, subtraction, and multiplication programs, the numbers are entered from right to left, while for the division program, the answers are entered from left to right. The multiplication program has been created to allow a maximum multiplier of 999. Once the first line has been entered, the computer automatically drops to the next line and positions itself in the correct column. The program for division has additional code because provision has been made so that all problems come out even. In all the programs, the only keys recognized by the computer during the answer phase are the numbers zero through nine and the DELete key.

To create the math.multiply and math.divide programs, make the necessary changes to the six instructions listed above. In addition, the math.multiply program contains new or different instructions in lines 426 to 429 and 915 to 1790. The math.divide program contains new or different instructions in lines 587, 600 to 627, 855 to 860, and 1000 to 1110. When all the changes and additions have been made to a program, be certain to save that program before either running it or entering the changes for another program.

All these programs can be included in one large program, but the flow of logic in the program would not be as easy to follow as it is with separate programs. Little would be gained by forcing everything into one program, because BASIC allows you to switch

from one program to another. To the user, the series of programs appear as one large program anyway.

Carefully consider what you want to save in the scores data file. There are several pieces of information that might be important to save, but a good rule is to save only what is absolutely necessary—what would be hard or impossible to calculate from existing information. For example, you could save the total number of problems, the number correct, the number wrong, the percentage, the name of the individual, the kind of mathematical operation, the number of digits chosen, and so on. If the programs were slightly altered, you could also save the actual problems missed, the number of tries on a particular problem, and the last question the person tried. Obviously, all this information is not necessary, although certain individuals might value and save information that others would not want.

Thus, the first step is to decide what information to save. In this example, four things will be saved: the type of operation, the number of digits in the operation, the number of correct answers, and the number of wrong answers. After deciding what to save, you need only save the assigned variables for these pieces of information. The code to do this is given in the next section.

THE MATH SYSTEM FILE OUTPUT ROUTINE

```
6000 rem **--file output routine--**
6005 append#1,(nc$) + "'s Scores"
6010 if ds = 62 then begin
6015 :::dclose#1
6020 :::dopen#1,(nc$) + "'s Scores",w
6025 bend
6030 print#1,s$:rem sign
6035 print#1,dt:rem # of digits
6040 print#1,cr:rem # of correct answers
6045 print#1,wr:rem # of wrong answers
6050 dclose#1
6055 :
6060 run "math.menu"
6065 :
6070 :
```

Some of this code should be familiar, but the sequence and a few commands might appear different. Remember that the Mailing List System used one program to create the Mail Data file and another program to add to it. Such a sequence is usually necessary when creating a file that will later be added to. The use of the DOPEN command with the W parameter places the file pointer at the very beginning of the file, and when the file is written to, any information already in the file is destroyed. That is the reason for the APPEND command. It adds information to the end of an existing file, but, in Commodore's BASIC 7.0, it will not create a file if that file does not already exist. Therefore, you need to use two commands—the DOPEN command to create the file and the APPEND command to add to the file after the first use.

The ds (disk status) variable eliminates the need for two separate programs. Now you have the following sequence: the first time the program is run, the computer will attempt to add information to a file that does not yet exist. When it finds that no such file exists,

the ds variable checks the disk drive and indicates an error. That error condition should be "FILE NOT FOUND," and have a value of 62. Therefore, if ds equals 62 you know that this is the first time the user has accessed the Math System and that the computer must create the data file rather than add to it. Thus, the file is created, and the first set of information is written into it. The second time (and succeeding times) the program is run, the computer simply appends information into the file, because it finds that such a file does exist. You have accomplished in one routine what would normally have taken two routines to do.

There are a few unique points to this routine. First, the BEGIN . . . BEND commands have been used. These BASIC 7.0 reserved words mark the beginning and end of a group of instructions that are executed only if specified conditions occur. The instructions that fall between these two commands close file number one and then open it in the write mode. These instructions go into effect only if the computer finds that a file does not exist when it tries to append information into it.

Second, the routine does not work without the dclose#1 instruction. If, for some reason, you have tried to append information into a file that does not exist and have received an error code that indicates that no such file could be found, you must close that nonexisting file before you can open another file. In addition to defying logic, this fact is not documented in any of the Commodore manuals. In fact, information concerning the APPEND command is relatively scarce. Some BASICs do allow the computer to create a file and add information with just the APPEND command, but Commodore BASIC 7.0 is not one of them. Again, this fact is not specified in any of the manuals.

Finally, you should also notice the use of a variable for the file name. In the Mailing List System programs, you always used the constant "Mail Data" for the file name. But in this situation, and in many file routines, it is more convenient to assign a variable as the file name. Every time an individual uses any of these programs, the information is kept in a file under that person's name. By using a variable for the file name, you eliminate the need for separate programs for each person who uses any of the math operation programs.

You must be careful to type your name the same way every time you use the programs. For example, if I answer that my name is DAVID the first time I use these programs, the file will be created under the name of DAVID. If I come back later and answer that my name is DAVE, a new and separate file will be created for DAVE. Therefore, two people with the same name must use different forms of their name, or their individual records will be combined in one file.

The need for consistency is the reason the entered name (line 460) is converted to lowercase characters (7000 to 7085). It is also the reason the variable cv$ (instead of name$) is used for the file name. It is possible to have at least three different logical forms of the same name: DAVID, David, or david. Without the conversion routine, each of these forms would create a new file. With this conversion routine, the name is always changed into lowercase letters with the first letter capitalized, i.e., David.

As with most things, there are advantages to the use of a variable for the file name, but there are also disadvantages. The user might get tired of being required to type his or her name. The use of a variable for the file name, however, remains a popular programming technique. The variable must be a string variable, because no file name can begin with a number. (The file name can contain a number, but it cannot *begin* with a number.) The variable, nc$, must be enclosed in parentheses.

THE MATH SCORES FILE INPUT ROUTINE

The file routine used in the program "math.scores" is very similar to the one just discussed, but instead of writing information to the diskette, this routine reads information from the diskette.

```
400 rem **--file output routine--**
405 :
410 rem *--data file--*
415 k = 1
420 dopen#1,(nc$) + "'s Scores"
425 input#1,s$(k):rem sign
430 input#1,dt(k):rem # of digits
435 input#1,cr(k):rem # of correct answers
440 input#1,wr(k):rem # of wrong answers
445 k = k + 1
450 if status = 0 then 425
455 dclose#1
460 :
465 :
```

This time the status function is used to test for the end of file number one, or the end of the data. If the status function is not included, there is no way of telling how much information or how many records exist in the file. You did not keep track of that information by writing out a counter to the file like you did in the Mailing List System. Without the status function, you would be informed of an "end of data" error and the program would halt. With the status function, the computer is instructed to get information only as long as an error condition does not exist. When the end of the file, or end of data error, occurs, the computer is informed to go to the next instruction (line 455) and proceed from there. At this point, the file is closed, because you are now certain that you have all the information the file contains. The use of the status function saves both programming and disk space.

You should notice one other major difference in this routine. In most of your programs, you have used FOR . . . NEXT loops. But this time, you do not know how many items the file contains, and therefore, you do not know how large the counter eventually needs to become. It is true that you could pick an arbitrary number, but a better method is the one used in this routine. This method is still a loop, because the computer is instructed to follow the instructions down to line 450 and then go back to the instruction at line 425 and do everything over again. The status function gets you out of this loop when the end of the file is encountered. When this loop is finished, you should have the desired values from the file, and can proceed to the display routine.

These math programs provide additional file handling techniques, as well as a set of useful drill and practice programs. The menu program uses the same method you have been using to display a set of choices and then run the appropriate program. Apart from file handling, the math programs also have some programming techniques that might prove interesting. The complete listings for these programs are presented in Fig. 8-1.

This chapter concludes the sequential-access files section of this book. The rest of the book is dedicated to relative-access (random-access) files. The relative-access section of this book will demonstrate that it is not necessary to read in all the data in order to display, add to, or change that information. In fact, some might find that the applications

presented in this sequential-access section might better be implemented by relative-access programs. If that is the case for you, I would encourage you to make the necessary changes in the Mailing List System and Math System programs after reading the relative-access section so that you have relative-access files instead of sequential-access files.

By now, some of you have no doubt found out that certain of these commands can be abbreviated and that there are shortcuts. I have intentionally avoided using these abbreviations and shortcuts because my purpose is to make the code as self-explanatory as possible. I encourage you to follow the example set in this book, but it is not necessary to spell out some of these commands or include the rem statements in order for these programs to work. One argument against including this expanded code format is that the program will operate faster, because each line and each character in the line must be interpreted by the command processor. I have not found a significant difference in speed of operation for the file application programs once the program has been loaded into the computer, but, the inclusion of all the rem statements and unabbreviated code does greatly expand the size of the file, and therefore, the file requires more time to load into the computer. If the slowness of the loading process bothers you, you might want to redo the programs without the rem statements and the unabbreviated code. I would suggest that you maintain a copy of the original program, because I have found that a fully documented program is worth a little extra load time.

NEW COMMANDS AND TERMS

1. ^: Raise to the power of the number following this symbol (when used as a math function).
2. RND: Generate a random number.
3. INT: Take only the integer portion of the number in parentheses.
4. LEN: Find the length of the string in terms of the number of characters in the string.
5. STR$: Convert the specified number into a string value.
6. VAL: Give the numeric value of a string.
7. SGN: Return the signum function.
8. DEF FN: Define a function. This useful command allows a programmer to set a single variable equal to a complete equation.
9. MOD: The remainder after division of one number by another.
10. ASC: Produces the decimal ASCII value of a specified string variable.

CHAPTER 8 QUESTIONS

1. A good rule to follow in deciding what information to save is to save: (a) everything possible, (b) as little as possible, (c) only what is absolutely necessary.
2. True or False: APPEND can open a file.
3. True or False: It is never possible to use a variable as a file name.
4. What type of variable can be used as a file name?
5. Which BASIC statement converts a number into a string?
6. Which BASIC statement converts a string into a number?
7. Which BASIC statement retrieves only the integer portion of a number?
8. Which BASIC statement can be used to test for the end of a sequential-access file?

Fig. 8-1. The Math System programs.

The math.menu Program

```
100 rem ***--math.menu--***
105 :
110 :
115 rem **--initialization--**
120 :black =  1
125 :white =  2
130 :::red =  3
135 :green =  6
140 ::blue =  7
145 yellow =  8
150 purple =  9
155 drkyel = 10
160 ::cyan = 12
165 ::gray = 16
170 ::chtr =  5
175 bckgnd =  0
195 :
200 blank$ = chr$(10):rem blank line
205 print chr$(14):rem upper/lower case mode
210 print chr$(8):rem disable char.set change
215 color bckgnd,black
220 fast
225 :
230 :
500 rem **--math menu--**
505 tb = 20:rem tab value--20 spaces to the right
510 scnclr
515 color chtr,gray
520 char 0,24,1,"MATH PROGRAM MENU":print
525 color chtr,purple
530 print:print tab(tb) "1.  ADDITION"
535 print:print tab(tb) "2.  SUBTRACTION"
540 print:print tab(tb) "3.  MULTIPLICATION"
545 print:print tab(tb) "4.  DIVISION"
550 print:print tab(tb) "5.  SCORES"
555 print:print tab(tb) "6.  INFORMATION"
557 print:print tab(tb) "7.  LIST OF FILES"
559 print:print tab(tb) "8.  END"
560 print:print tab(tb)
565 input "Which Program Number";number$
570 number = val(number$)
575 :
700 rem *--direct computer--*
710 if number = 1 then 1000
720 if number = 2 then 2000
730 if number = 3 then 3000
740 if number = 4 then 4000
750 if number = 5 then 5000
760 if number = 6 then 6000
770 if number = 7 then 7000
780 if number = 8 then 8000
790 :
900 rem *--incorrect choice message--*
905 print:print
910 color chtr,white:print tab(tb) "Incorrect Choice!"
915 color chtr,yellow:print:print tab(tb) "Press ";
920 color chtr,white:print "RETURN";
925 color chtr,yellow:print " to continue:";
930 gosub 19000:rem getkey subroutine
935 goto 500:rem menu--check again
940 :
945 :
1000 rem **--math addition program--**
1005 file$ = "MATH.ADD"
1010 gosub 17000:rem new program routine
```

```
1015 run "math.add"
1020 :
1025 :
2000 rem **--math subtraction program--**
2005 file$ = "MATH.SUBTRACT"
2010 gosub 17000:rem new program routine
2015 run "math.subtract"
2020 :
2025 :
3000 rem **--math multiplication program--**
3005 file$ = "MATH.MULTIPLY"
3010 gosub 17000:rem new program routine
3015 run "math.multiply"
3020 :
3025 :
4000 rem **--math division program--**
4005 file$ = "MATH.DIVIDE"
4010 gosub 17000:rem new program routine
4015 run "math.divide"
4020 :
4025 :
5000 rem **--math scores program--**
5005 file$ = "MATH.SCORES"
5010 gosub 17000:rem new program routine
5015 run "math.scores"
5020 :
5025 :
6000 rem **--information--**
6005 scnclr
6010 print "This is a series of math drill and practice programs."
6015 print
6020 print "It is designed to allow for as much flexibility as"
6025 print
6030 print "possible. The question about the number of digits"
6035 print
6040 print "might, at first, seem confusing. The question simply"
6045 print
6050 print "asks for the greatest number of digits possible in"
6055 print
6060 print "either figure. The next two questions further allow"
6065 print
6070 print "you to limit the possible problems. For example, if"
6075 print
6080 print "you wanted to practice multiplying by '5', you could"
6085 print
6090 print "choose three digit numbers and then answer with a '5'"
6095 print
6100 print "for each of the next two questions. You would then be"
6105 print
6110 print "given problems like: 345 x 5 or 823 x 5."
6115 print
6120 gosub 15000:rem pause routine
6125 print
6130 print "Another example would be to add two digit numbers by"
6135 print
6140 print "answering the questions in this way:"
6145 print
6150 print "HOW MANY DIGITS--2"
6155 print
6160 print "LARGEST NUMBER--99"
6165 print
6170 print "SMALLEST NUMBER--1"
6175 print
6180 print "You could then get problems like:"
6185 print
6190 print "58 + 34 or 87 + 9."
6195 print
6200 print "Trying the different possibilities will soon indicate"
```

```
6205 print
6210 print "the flexibility."
6215 print
6220 gosub 15000:rem pause routine
6225 print
6230 print "The division section will only offer problems that come"
6235 print
6240 print "out even. You may have to wait a short time for the next"
6245 print
6250 print "problem. This is because the numbers generated must meet"
6255 print
6260 print "certain specifications. The process speeds up when the"
6265 print
6270 print "number being divided contains at least two more digits"
6275 print
6280 print "than the divisor."
6285 print
6290 gosub 15000:rem pause routine
6295 print
6300 print "This is not a professional program and, therefore, does"
6305 print
6310 print "not do a lot of error checking. You can crash the programs"
6315 print
6320 print "with confusing answers or miskakes in typing."
6325 print:print:print
6340 print "This series of programs was done mainly to demonstrate,"
6345 print
6350 print "in a useful manner, certain file handling capabilities"
6355 print
6360 print "and techniques."
6365 print
6370 gosub 15000:rem pause routine
6375 :
6380 goto 500:rem menu
6385 :
6390 :
7000 rem **--list of files routine--**
7005 scnclr
7010 directory
7015 print:print
7020 print "Are you ready to return to the menu? ";
7025 gosub 18000:rem y/n input routine
7030 if yes$ = "y" then 500:rem menu
7035 goto 7000:rem check again
7040 :
7045 :
8000 rem **--end routine--**
8005 scnclr
8010 color chtr,grey
8015 char 0,18,9,"That's all for this session! See you next time."
8020 color chtr,drkyel
8025 char 0,0,22
8030 slow
8035 end
8040 :
8045 :
15000 rem **--pause routine--**
15005 char 0,0,24,"Press the "
15010 color chtr,white
15015 print "RETURN";
15020 color chtr,purple
15025 print " key to continue: ";
15030 gosub 19000:rem getkey subroutine
15035 scnclr
15040 return
15045 :
15050 :
17000 rem **--new program routine--**
```

```
17005 scnclr:char 0,15,5
17010 print "You have selected the ";
17015 color chtr,red
17020 print file$;
17025 color chtr,purple
17030 print " program."
17035 char 0,15,9
17040 print "Is this the program you want? ";
17045 gosub 18000:rem y/n input routine
17050 if yes$ = "n" then 500:rem menu
17055 :
17060 rem *--loading message--*
17065 scnclr
17070 char 0,15,5
17075 print "Please wait! I'm loading....";
17080 color chtr,red
17085 print file$
17090 color chtr,purple
17095 return
17100 :
17105 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,purple:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,purple:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 print
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,purple
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print a$;
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
```

The math.add Program

```
100 rem ***--addition--***
105 :
110 :
115 rem **--initialization--**
120 :black =  1
125 :white =  2
130 :::red =  3
135 :green =  6
140 ::blue =  7
145 yellow =  8
150 purple =  9
155 drkyel = 10
160 ::cyan = 12
165 ::gray = 16
```

```
170 ::chtr =  5
175 bckgnd =  0
195 :
200 blank$ = chr$(10):rem blank line
205 print chr$(14):rem upper/lower case mode
210 print chr$(8):rem disable char.set change
215 color bckgnd,black
220 :
225 :
300 rem **--variable list--**
305 rem a  = top number
310 rem b  = bottom number
315 rem c  = correct answer
320 rem d  = student's answer
325 rem q  = counter
330 rem w  = previous answer
335 rem z  = number of tries
340 rem cr = # of correct answers
345 rem wr = # of wrong answers
350 rem dt = # of digits
355 rem da = # of digits in a
360 rem db = # of digits in b
365 rem dc = # of digits in c
370 rem dm = # of digits in max amnt
375 rem other variables are descriptive
380 :
385 :
400 rem **--request user information--**
405 scnclr
410 char 0,0,5
415 input "What is the maximum number of digits any problem should have";digit$
420 print
425 input "What is the largest figure for the number you are adding by";max$
430 print
435 input "What is the smallest figure for the number you are adding by";mn$
440 dt = val(digit$):max = val(max$):mn = val(mn$)
445 digit = val(digit$)
450 dt = digit:digit = 10 ^ digit
455 print
460 input "What is your name ";name$
465 :
470 rem *--convert to lowercase--*
475 cvt$ = name$
480 gosub 7000:rem convert to lowercase routine
485 :
490 :
500 rem **--create problem--**
505 :
510 dm = len(max$)
515 :
520 rem when b's digits < dt
525 if dt = dm + 1 or dt < dm + 1 then 560
530 dm = 10 ^ dm
535 b = int(rnd(0) * dm)
540 if b < mn then 535
545 if b > max then 535
550 goto 585:rem get # for a
555 :
560 z = 1
565 b = int(rnd(0) * digit)
570 if b < mn then 565
575 if b > max then 565
580 :
585 a = int(rnd(0) * digit)
590 :
600 rem *--answer--*
605 c = a + b
610 s$ = "+":rem determine operation
615 sn$ = "addition"
```

```
620 if c < 0 then 565:rem for subtract.
625 if c = w then 565:rem previous ans.
630 w = c
635 :
640 :
700 rem **--display problem--**
705 :
710 rem *--determine lengths--*
715 a$ = str$(a)
720 da = len(a$)
725 a$ = right$(a$,da - 1):rem strip off sign
730 da = len(a$)
735 b$ = str$(b)
740 db = len(b$)
745 b$ = right$(b$,db - 1):rem strip off sign
750 db = len(b$)
755 c$ = str$(c)
760 dc = len(c$)
765 c$ = right$(c$,dc - 1):rem strip off sign
770 dc = len(c$)
775 :
800 rem *--format--*
805 scnclr
810 color chtr,drkyel
815 print tab(24);nc$;"'s ";sn$;" practice."
820 color chtr,red
825 print:print:print
830 print tab(24) "Press the ";
835 color chtr,white
840 print "ESC";
845 color chtr,red
850 print " key when finished!"
855 char 0,(37 - da),10,a$
860 char 0,(37 - (db + 1)),11,s$ + b$
865 print
870 :
875 rem *--draw line--*
880 q = 1
885 if da > db then q = 0
890 print tab(37 - (dt + q))
895 for i = 1 to (dt + q)
900 print "-";
905 next i
910 :
1000 rem **--get answer--**
1005 print
1010 for i = 1 to dc
1015 print tab(37 - i)
1020 getkey ua$
1025 if ua$ = chr$(20) then begin
1030 :::char 0,(37 - (i - 1)),13," "
1035 :::i = i - 1
1040 :::lu = len(answer$)
1045 :::if lu < 1 then 1015
1050 :::answer$ = mid$(answer$,2,lu - 1)
1055 :::goto 1015
1060 bend
1062 if ua$ = chr$(27) then 1070
1065 if asc(ua$) < 48 or asc(ua$) > 57 then 1015
1070 char 0,(37 - i),13,ua$
1075 answer$ = ua$ + answer$
1080 if answer$ = chr$(27) then print:goto 5000
1085 next i
1090 :
1095 :
2000 rem **--check answer--**
2005 print
2010 d = val(answer$):answer$ = ""
2015 if d = c then 3000:rem correct
2020 if z < 3 then 4000:rem wrong
```

```
2025 :
2030 rem *--give answer after 3 tries--*
2035 char 0,24,18,"No, the answer is "
2040 color chtr,drkyel
2045 print c
2050 color chtr,red
2055 char 0,24,20,a$ + " " + s$ + " " + b$ + " = " + c$
2060 print:print:z = 1:wr = wr + 1
2065 char 0,24,22, "Press "
2070 color chtr,white
2075 print "RETURN";
2080 color chtr,red
2085 print " to continue: ";
2090 gosub 19000:rem getkey subroutine
2095 goto 500:rem get another problem
2100 :
2105 :
3000 rem **--correct answer routine--**
3005 char 0,33,20,"GOOD!"
3010 for i = 1 to 100:next i
3015 cr = cr + 1
3020 goto 500:rem get another problem
3025 :
3030 :
4000 rem **--wrong answer routine--**
4005 char 0,24,20,"No, Please try again."
4010 z = z + 1
4015 print
4020 for i = 1 to 100:next i
4025 goto 800
4030 :
4035 :
5000 rem **--total routine--**
5005 scnclr
5010 char 0,24,5,"You answered"
5015 color chtr,yellow
5020 print cr;
5025 color chtr,red
5030 print "correct!"
5035 print:print
5040 char 0,24,7,"You answered"
5045 color chtr,yellow
5050 print wr;
5055 color chtr,red
5060 print "wrong."
5065 nb = cr + wr
5070 pc = int(cr/nb*100)
5075 char 0,24,9,"You got "
5080 pc$ = str$(pc)
5085 print pc$;
5090 print "% correct!"
5092 print:print
5095 :
5100 :
6000 rem **--file output routine--**
6005 append#1,(nc$) + "'s Scores"
6010 if ds = 62 then begin
6015 :::dclose#1
6020 :::dopen#1,(nc$) + "'s Scores",w
6025 bend
6030 print#1,s$:rem sign
6035 print#1,dt:rem # of digits
6040 print#1,cr:rem # of correct answers
6045 print#1,wr:rem # of wrong answers
6050 dclose#1
6055 :
6060 run "math.menu"
6065 :
6070 :
```

```
 7000 rem **--convert to lowercase--**
 7005 for cv = 1 to len(cv$)
 7010 x = asc(mid$(cv$,cv,1))
 7015 if x > 192 then x = x - 128
 7020 nc$ = nc$ + chr$(x)
 7025 next cv
 7030 :
 7035 cv$ = nc$
 7040 f = asc(left$(cv$,1))
 7045 f = f + 128
 7050 nc$ = chr$(f) + right$(cv$,len(cv$)- 1)
 7055 return
 7060 :
 7065 rem na$ = originally typed name
 7070 rem cv$ = converted to lowercase
 7075 rem nc$ = 1st letter/uppercase
 7080 :
 7085 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print a$;
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
```

The math.subtract Program

```
  100 rem ***--subtraction--***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white =  2
  130 :::red =  3
  135 :green =  6
  140 ::blue =  7
  145 yellow =  8
  150 purple =  9
  155 drkyel = 10
  160 ::cyan = 12
  165 ::gray = 16
  170 ::chtr =  5
  175 bckgnd =  0
  195 :
  200 blank$ = chr$(10):rem blank line
  205 print chr$(14):rem upper/lower case mode
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 :
  225 :
  300 rem **--variable list--**
  305 rem a  = top number
  310 rem b  = bottom number
  315 rem c  = correct answer
  320 rem d  = student's answer
  325 rem q  = counter
  330 rem w  = previous answer
  335 rem z  = number of tries
  340 rem cr = # of correct answers
  345 rem wr = # of wrong answers
  350 rem dt = # of digits
  355 rem da = # of digits in a
```

```
360 rem db = # of digits in b
365 rem dc = # of digits in c
370 rem dm = # of digits in max amnt
375 rem other variables are descriptive
380 :
385 :
400 rem **--request user information--**
405 scnclr
410 char 0,0,5
415 input "What is the maximum number of digits any problem should have";digit$
420 print
425 input "What is the largest figure for the number you are subtracting by";max$
430 print
435 input "What is the smallest figure for the number you are subtracting by";mn$
440 dt = val(digit$):max = val(max$):mn = val(mn$)
445 digit = val(digit$)
450 dt = digit:digit = 10 ^ digit
455 print
460 input "What is your name ";name$
465 :
470 rem *--convert to lowercase--*
475 cvt$ = name$
480 gosub 7000:rem convert to lowercase routine
485 :
490 :
500 rem **--create problem--**
505 :
510 dm = len(max$)
515 :
520 rem when b's digits < dt
525 if dt = dm + 1 or dt < dm + 1 then 560
530 dm = 10 ^ dm
535 b = int(rnd(0) * dm)
540 if b < mn then 535
545 if b > max then 535
550 goto 585:rem get # for a
555 :
560 z = 1
565 b = int(rnd(0) * digit)
570 if b < mn then 565
575 if b > max then 565
580 :
585 a = int(rnd(0) * digit)
590 :
600 rem *--answer--*
605 c = a - b
610 s$ = "-":rem determine operation
615 sn$ = "subtraction"
620 if c < 0 then 565:rem for subtract.
625 if c = w then 565:rem previous ans.
630 w = c
635 :
640 :
700 rem **--display problem--**
705 :
710 rem *--determine lengths--*
715 a$ = str$(a)
720 da = len(a$)
725 a$ = right$(a$,da - 1):rem strip off sign
730 da = len(a$)
735 b$ = str$(b)
740 db = len(b$)
745 b$ = right$(b$,db - 1):rem strip off sign
750 db = len(b$)
755 c$ = str$(c)
760 dc = len(c$)
765 c$ = right$(c$,dc - 1):rem strip off sign
770 dc = len(c$)
775 :
```

```
800 rem *--format--*
805 scnclr
810 color chtr,drkyel
815 print tab(24);nc$;"'s ";sn$;" practice."
820 color chtr,red
825 print:print:print
830 print tab(24) "Press the ";
835 color chtr,white
840 print "ESC";
845 color chtr,red
850 print " key when finished!"
855 char 0,(37 - da),10,a$
860 char 0,(37 - (db + 1)),11,s$ + b$
865 print
870 :
875 rem *--draw line--*
880 q = 1
885 if da > db then q = 0
890 print tab(37 - (dt + q))
895 for i = 1 to (dt + q)
900 print "-";
905 next i
910 :
1000 rem **--get answer--**
1005 print
1010 for i = 1 to dc
1015 print tab(37 - i)
1020 getkey ua$
1025 if ua$ = chr$(20) then begin
1030 :::char 0,(37 - (i - 1)),13," "
1035 :::i = i - 1
1040 :::lu = len(answer$)
1045 :::if lu < 1 then 1015
1050 :::answer$ = mid$(answer$,2,lu - 1)
1055 :::goto 1015
1060 bend
1062 if ua$ = chr$(27) then 1070
1065 if asc(ua$) < 48 or asc(ua$) > 57 then 1015
1070 char 0,(37 - i),13,ua$
1075 answer$ = ua$ + answer$
1080 if answer$ = chr$(27) then print:goto 5000
1085 next i
1090 :
1095 :
2000 rem **--check answer--**
2005 print
2010 d = val(answer$):answer$ = ""
2015 if d = c then 3000:rem correct
2020 if z < 3 then 4000:rem wrong
2025 :
2030 rem *--give answer after 3 tries--*
2035 char 0,24,18,"No, the answer is "
2040 color chtr,drkyel
2045 print c
2050 color chtr,red
2055 char 0,24,20,a$ + " " + s$ + " " + b$ + " = " + c$
2060 print:print:z = 1:wr = wr + 1
2065 char 0,24,22, "Press "
2070 color chtr,white
2075 print "RETURN";
2080 color chtr,red
2085 print " to continue: ";
2090 gosub 19000:rem getkey subroutine
2095 goto 500:rem get another problem
2100 :
2105 :
3000 rem **--correct answer routine--**
3005 char 0,33,20,"GOOD!"
3010 for i = 1 to 100:next i
```

```
 3015 cr = cr + 1
 3020 goto 500:rem get another problem
 3025 :
 3030 :
 4000 rem **--wrong answer routine--**
 4005 char 0,24,20,"No, Please try again."
 4010 z = z + 1
 4015 print
 4020 for i = 1 to 100:next i
 4025 goto 800
 4030 :
 4035 :
 5000 rem **--total routine--**
 5005 scnclr
 5010 char 0,24,5,"You answered"
 5015 color chtr,yellow
 5020 print cr;
 5025 color chtr,red
 5030 print "correct!"
 5035 print:print
 5040 char 0,24,7,"You answered"
 5045 color chtr,yellow
 5050 print wr;
 5055 color chtr,red
 5060 print "wrong."
 5065 nb = cr + wr
 5070 pc = int(cr/nb*100)
 5075 char 0,24,9,"You got "
 5080 pc$ = str$(pc)
 5085 print pc$;
 5090 print "% correct!"
 5092 print:print
 5095 :
 5100 :
 6000 rem **--file output routine--**
 6005 append#1,(nc$) + "'s Scores"
 6010 if ds = 62 then begin
 6015 :::dclose#1
 6020 :::dopen#1,(nc$) + "'s Scores",w
 6025 bend
 6030 print#1,s$:rem sign
 6035 print#1,dt:rem # of digits
 6040 print#1,cr:rem # of correct answers
 6045 print#1,wr:rem # of wrong answers
 6050 dclose#1
 6055 :
 6060 run "math.menu"
 6065 :
 6070 :
 7000 rem **--convert to lowercase--**
 7005 for cv = 1 to len(cv$)
 7010 x = asc(mid$(cv$,cv,1))
 7015 if x > 192 then x = x - 128
 7020 nc$ = nc$ + chr$(x)
 7025 next cv
 7030 :
 7035 cv$ = nc$
 7040 f = asc(left$(cv$,1))
 7045 f = f + 128
 7050 nc$ = chr$(f) + right$(cv$,len(cv$)- 1)
 7055 return
 7060 :
 7065 rem na$ = originally typed name
 7070 rem cv$ = converted to lowercase
 7075 rem nc$ = 1st letter/uppercase
 7080 :
 7085 :
19000 rem **--getkey subroutine--**
19005 getkey a$
```

```
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print a$;
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
```

The math.multiply Program

```
  100 rem ***--multiplication--***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white =  2
  130 :::red =  3
  135 :green =  6
  140 ::blue =  7
  145 yellow =  8
  150 purple =  9
  155 drkyel = 10
  160 ::cyan = 12
  165 ::gray = 16
  170 ::chtr =  5
  175 bckgnd =  0
  195 :
  200 blank$ = chr$(10):rem blank line
  205 print chr$(14):rem upper/lower case mode
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 :
  225 :
  300 rem **--variable list--**
  305 rem a  = top number
  310 rem b  = bottom number
  315 rem c  = correct answer
  320 rem d  = student's answer
  325 rem q  = counter
  330 rem w  = previous answer
  335 rem z  = number of tries
  340 rem cr = # of correct answers
  345 rem wr = # of wrong answers
  350 rem dt = # of digits
  355 rem da = # of digits in a
  360 rem db = # of digits in b
  365 rem dc = # of digits in c
  370 rem dm = # of digits in max amnt
  375 rem other variables are descriptive
  380 :
  385 :
  400 rem **--request user information--**
  405 scnclr
  410 char 0,0,5
  415 input "What is the maximum number of digits any problem should have";digit$
  420 print
  425 input "What is the largest figure for the number you are multiplying by";max$
  426 if val(max$) > 999 then begin
  427 :::print
  428 :::print "You may not multiply by a number greater than 999!":print:goto 425
  429 bend
  430 print
  435 input "What is the smallest figure for the number you are multiplying by";mn$
  440 dt = val(digit$):max = val(max$):mn = val(mn$)
  445 digit = val(digit$)
```

```
450 dt = digit:digit = 10 ^ digit
455 print
460 input "What is your name ";name$
465 :
470 rem *--convert to lowercase--*
475 cvt$ = name$
480 gosub 7000:rem convert to lowercase routine
485 :
490 :
500 rem **--create problem--**
505 :
510 dm = len(max$)
515 :
520 rem when b's digits < dt
525 if dt = dm + 1 or dt < dm + 1 then 560
530 dm = 10 ^ dm
535 b = int(rnd(0) * dm)
540 if b < mn then 535
545 if b > max then 535
550 goto 585:rem get # for a
555 :
560 z = 1
565 b = int(rnd(0) * digit)
570 if b < mn then 565
575 if b > max then 565
580 :
585 a = int(rnd(0) * digit)
590 :
600 rem *--answer--*
605 c = a * b
610 s$ = "X":rem determine operation
615 sn$ = "multiplication"
620 if c < 0 then 565:rem for subtract.
625 if c = w then 565:rem previous ans.
630 w = c
635 :
640 :
700 rem **--display problem--**
705 :
710 rem *--determine lengths--*
715 a$ = str$(a)
720 da = len(a$)
725 a$ = right$(a$,da - 1):rem strip off sign
730 da = len(a$)
735 b$ = str$(b)
740 db = len(b$)
745 b$ = right$(b$,db - 1):rem strip off sign
750 db = len(b$)
755 c$ = str$(c)
760 dc = len(c$)
765 c$ = right$(c$,dc - 1):rem strip off sign
770 dc = len(c$)
775 :
800 rem *--format--*
805 scnclr
810 color chtr,drkyel
815 print tab(24);nc$;"'s ";sn$;" practice."
820 color chtr,red
825 print:print:print
830 print tab(24) "Press the ";
835 color chtr,white
840 print "ESC";
845 color chtr,red
850 print " key when finished!"
855 char 0,(37 - da),10,a$
860 char 0,(37 - (db + 1)),11,s$ + b$
865 print
870 :
875 rem *--draw line--*
```

```
 880 q = 1
 885 if da > db then q = 0
 890 print tab(37 - (dt + q))
 895 for i = 1 to (dt + q)
 900 print "-";
 905 next i
 910 :
 915 :
 920 rem **--new lines for multiplication--**
 922 :
 923 rem *--determin individual numbers--*
 925 b1$ = right$(b$,1)
 930 if db = 2 then b2$ = left$(b$,1)
 935 if db = 3 then b2$ = mid$(b$,2,1)
 940 if db = 3 then b3$ = left$(b$,1)
 945 b1 = val(b1$)
 950 if db = 2 then b2 = val(b2$)
 955 if db = 3 then b2 = val(b2$):b3 = val(b3$)
 957 :
 958 rem *--multiply individual numbers by value of a--*
 960 m1 = b1 * a
 965 if db = 2 then m2 = b2 * a
 970 if db = 3 then m2 = b2 * a:m3 = b3 * a
 972 :
 973 rem *--determin number of digits in individual mult.result--*
 975 m1$ = str$(m1)
 980 if db = 2 then m2$ = str$(m2)
 985 if db = 3 then m2$ = str$(m2):m3$ = str$(m3)
 990 g1 = len(m1$) - 1
 995 if db = 2 then g2 = len(m2$) - 1
 997 if db = 3 then g2 = len(m2$) - 1:g3 = len(m3$) - 1
 998 :
 999 :
1000 rem **--get individual results--**
1005 :
1010 rem *--first result--*
1015 print
1020 for i = 1 to g1
1025 getkey ua$
1030 if ua$ = chr$(20) then begin
1035 :::char 0,(37 - (i - 1)),13," "
1040 :::i = i - 1:goto 1025
1045 bend
1047 if ua$ = chr$(27) then 1055
1050 if asc(ua$) < 48 or asc(ua$) > 57 then 1025
1055 char 0,(37 - i),13,ua$
1060 if ua$ = chr$(27) then print:goto 5000
1062 if db = 1 then answer$ = ua$ + answer$
1065 next i
1070 :
1075 if db = 1 then 2000
1080 :
1100 rem *--second result--*
1115 print
1120 for i = 1 to g2
1125 getkey ua$
1130 if ua$ = chr$(20) then begin
1135 :::char 0,(36 - (i - 1)),14," "
1140 :::i = i - 1:goto 1125
1145 bend
1150 if asc(ua$) < 48 or asc(ua$) > 57 then 1125
1155 char 0,(36 - i),14,ua$
1160 if ua$ = chr$(27) then print:goto 5000
1165 next i
1170 :
1175 if db = 2 then 1500
1180 :
1200 rem *--third result--*
1215 print
```

```
1220 for i = 1 to g3
1225 getkey ua$
1230 if ua$ = chr$(20) then begin
1235 :::char 0,(35 - (i - 1)),15," "
1240 :::i = i - 1:goto 1225
1245 bend
1250 if asc(ua$) < 48 or asc(ua$) > 57 then 1225
1255 char 0,(35 - i),15,ua$
1260 if ua$ = chr$(27) then print:goto 5000
1265 next i
1270 :
1275 :
1500 rem **--draw bottom line--**
1502 print
1505 print tab(37 - dc)
1510 for i = 1 to dc
1515 print "-";
1520 next i
1525 :
1530 :
1700 rem **--get answer--**
1705 answer$ = ""
1710 for i = 1 to dc
1715 print tab(37 - i)
1720 getkey ua$
1725 if ua$ = chr$(20) then begin
1730 :::char 0,(37 - (i - 1)),14 + db," "
1735 :::i = i - 1
1740 :::lu = len(answer$)
1745 :::if lu < 1 then 1715
1750 :::answer$ = mid$(answer$,2,lu - 1)
1755 :::goto 1715
1760 bend
1765 if ua$ = chr$(13) then 1715
1770 char 0,(37 - i),14 + db,ua$
1775 answer$ = ua$ + answer$
1780 if answer$ = chr$(27) then print:goto 5000
1785 next i
1790 :
2000 rem **--check answer--**
2005 print
2010 d = val(answer$):answer$ = ""
2015 if d = c then 3000:rem correct
2020 if z < 3 then 4000:rem wrong
2025 :
2030 rem *--give answer after 3 tries--*
2035 char 0,24,18,"No, the answer is "
2040 color chtr,drkyel
2045 print c
2050 color chtr,red
2055 char 0,24,20,a$ + " " + s$ + " " + b$ + " = " + c$
2060 print:print:z = 1:wr = wr + 1
2065 char 0,24,22, "Press "
2070 color chtr,white
2075 print "RETURN";
2080 color chtr,red
2085 print " to continue: ";
2090 gosub 19000:rem getkey subroutine
2095 goto 500:rem get another problem
2100 :
2105 :
3000 rem **--correct answer routine--**
3005 char 0,33,20,"GOOD!"
3010 for i = 1 to 100:next i
3015 cr = cr + 1
3020 goto 500:rem get another problem
3025 :
3030 :
4000 rem **--wrong answer routine--**
```

```
4005 char 0,24,20,"No, Please try again."
4010 z = z + 1
4015 print
4020 for i = 1 to 100:next i
4025 goto 800
4030 :
4035 :
5000 rem **--total routine--**
5005 scnclr
5010 char 0,24,5,"You answered"
5015 color chtr,yellow
5020 print cr;
5025 color chtr,red
5030 print "correct!"
5035 print:print
5040 char 0,24,7,"You answered"
5045 color chtr,yellow
5050 print wr;
5055 color chtr,red
5060 print "wrong."
5065 nb = cr + wr
5070 pc = int(cr/nb*100)
5075 char 0,24,9,"You got "
5080 pc$ = str$(pc)
5085 print pc$;
5090 print "% correct!"
5092 print:print
5095 :
5100 :
6000 rem **--file output routine--**
6005 append#1,(nc$) + "'s Scores"
6010 if ds = 62 then begin
6015 :::dclose#1
6020 :::dopen#1,(nc$) + "'s Scores",w
6025 bend
6030 print#1,s$:rem sign
6035 print#1,dt:rem # of digits
6040 print#1,cr:rem # of correct answers
6045 print#1,wr:rem # of wrong answers
6050 dclose#1
6055 :
6060 run "math.menu"
6065 :
6070 :
7000 rem **--convert to lowercase--**
7005 for cv = 1 to len(cv$)
7010 x = asc(mid$(cv$,cv,1))
7015 if x > 192 then x = x - 128
7020 nc$ = nc$ + chr$(x)
7025 next cv
7030 :
7035 cv$ = nc$
7040 f = asc(left$(cv$,1))
7045 f = f + 128
7050 nc$ = chr$(f) + right$(cv$,len(cv$)- 1)
7055 return
7060 :
7065 rem na$ = originally typed name
7070 rem cv$ = converted to lowercase
7075 rem nc$ = 1st letter/uppercase
7080 :
7085 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
```

```
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print a$;
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
```

The math.divide Program

```
  100 rem ***--division--***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white =  2
  130 :::red =  3
  135 :green =  6
  140 ::blue =  7
  145 yellow =  8
  150 purple =  9
  155 drkyel = 10
  160 ::cyan = 12
  165 ::gray = 16
  170 ::chtr =  5
  175 bckgnd =  0
  195 :
  200 blank$ = chr$(10):rem blank line
  205 print chr$(14):rem upper/lower case mode
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 :
  225 :
  300 rem **--variable list--**
  305 rem a  = top number
  310 rem b  = bottom number
  315 rem c  = correct answer
  320 rem d  = student's answer
  325 rem q  = counter
  330 rem w  = previous answer
  335 rem z  = number of tries
  340 rem cr = # of correct answers
  345 rem wr = # of wrong answers
  350 rem dt = # of digits
  355 rem da = # of digits in a
  360 rem db = # of digits in b
  365 rem dc = # of digits in c
  370 rem dm = # of digits in max amnt
  375 rem other variables are descriptive
  380 :
  385 :
  400 rem **--request user information--**
  405 scnclr
  410 char 0,0,5
  415 input "What is the maximum number of digits any problem should have";digit$
  420 print
  425 input "What is the largest figure for the number you are dividing by";max$
  430 print
  435 input "What is the smallest figure for the number you are dividing by";mn$
  440 dt = val(digit$):max = val(max$):mn = val(mn$)
  445 digit = val(digit$)
  450 dt = digit:digit = 10 ^ digit
  455 print
  460 input "What is your name ";name$
  465 :
  470 rem *--convert to lowercase--*
  475 cvt$ = name$
  480 gosub 7000:rem convert to lowercase routine
  485 :
  490 :
```

```
 500 rem **--create problem--**
 505 :
 510 dm = len(max$)
 515 :
 520 rem when b's digits < dt
 525 if dt = dm + 1 or dt < dm + 1 then 560
 530 dm = 10 ^ dm
 535 b = int(rnd(0) * dm)
 540 if b < mn then 535
 545 if b > max then 535
 550 goto 585:rem get # for a
 555 :
 560 z = 1
 565 b = int(rnd(0) * digit)
 570 if b < mn then 565
 575 if b > max then 565
 580 :
 585 a = int(rnd(0) * digit)
 587 if a = 0 or a < b then 585
 590 :
 600 rem *--answer--*
 602 def fn mod(c) = int((a/b - int(a/b)) * b + .05) * sgn(a/b)
 605 c = int(a)/(b)
 607 c = int(c)
 610 s$ = "/":rem determine operation
 615 sn$ = "division"
 620 if c < 0 then 565:rem for subtract.
 625 if c = w then 565:rem previous ans.
 627 if fn mod(rm) <> 0 then 585
 630 w = c
 635 :
 640 :
 700 rem **--display problem--**
 705 :
 710 rem *--determine lengths--*
 715 a$ = str$(a)
 720 da = len(a$)
 725 a$ = right$(a$,da - 1):rem strip off sign
 730 da = len(a$)
 735 b$ = str$(b)
 740 db = len(b$)
 745 b$ = right$(b$,db - 1):rem strip off sign
 750 db = len(b$)
 755 c$ = str$(c)
 760 dc = len(c$)
 765 c$ = right$(c$,dc - 1):rem strip off sign
 770 dc = len(c$)
 775 :
 800 rem *--format--*
 805 scnclr
 810 color chtr,drkyel
 815 print tab(24);nc$;"'s ";sn$;" practice."
 820 color chtr,red
 825 print:print:print
 830 print tab(24) "Press the ";
 835 color chtr,white
 840 print "ESC";
 845 color chtr,red
 850 print " key when finished!"
 855 char 0,37,10
 856 for i = 1 to dt + 1
 857 print chr$(2);" ";:rem top line
 858 next i:print chr$(130)
 860 print:char 0,(37 - db),11,b$ + ")" + a$
 870 :
 880 :
1000 rem **--get answer--**
1005 for i = 1 to dc
1010 print tab(37 - i)
```

```
1015 getkey ua$
1020 if ua$ = chr$(20) then begin
1025 print chr$(2);
1030 char 0,(((37 + da) - (dc - 1)) + (i - 2)),10," "
1035 print chr$(130)
1040 :::i = i - 1
1045 :::lu = len(answer$)
1050 :::if lu < 1 then 1010
1055 :::answer$ = left$(answer$,lu - 1)
1060 :::goto 1010
1065 bend
1067 if ua$ = chr$(27) then 1075
1070 if asc(ua$) < 48 or asc(ua$) > 57 then 1010
1075 print chr$(2);
1080 char 0,(((37 + da) - (dc - 1)) + (i - 1)),10,ua$
1085 print chr$(130)
1090 answer$ = answer$ + ua$
1095 if answer$ = chr$(27) then print:goto 5000
1100 next i
1105 :
1110 :
2000 rem **--check answer--**
2005 print
2010 d = val(answer$):answer$ = ""
2015 if d = c then 3000:rem correct
2020 if z < 3 then 4000:rem wrong
2025 :
2030 rem *--give answer after 3 tries--*
2035 char 0,24,18,"No, the answer is "
2040 color chtr,drkyel
2045 print c
2050 color chtr,red
2055 char 0,24,20,a$ + " " + s$ + " " + b$ + " = " + c$
2060 print:print:z = 1:wr = wr + 1
2065 char 0,24,22, "Press "
2070 color chtr,white
2075 print "RETURN";
2080 color chtr,red
2085 print " to continue: ";
2090 gosub 19000:rem getkey subroutine
2095 goto 500:rem get another problem
2100 :
2105 :
3000 rem **--correct answer routine--**
3005 char 0,33,20,"GOOD!"
3010 for i = 1 to 100:next i
3015 cr = cr + 1
3020 goto 500:rem get another problem
3025 :
3030 :
4000 rem **--wrong answer routine--**
4005 char 0,24,20,"No, Please try again."
4010 z = z + 1
4015 print
4020 for i = 1 to 100:next i
4025 goto 800
4030 :
4035 :
5000 rem **--total routine--**
5005 scnclr
5010 char 0,24,5,"You answered"
5015 color chtr,yellow
5020 print cr;
5025 color chtr,red
5030 print "correct!"
5035 print:print
5040 char 0,24,7,"You answered"
5045 color chtr,yellow
5050 print wr;
```

```
5055 color chtr,red
5060 print "wrong."
5065 nb = cr + wr
5070 pc = int(cr/nb*100)
5075 char 0,24,9,"You got "
5080 pc$ = str$(pc)
5085 print pc$;
5090 print "% correct!"
5092 print:print
5095 :
5100 :
6000 rem **--file output routine--**
6005 append#1,(nc$) + "'s Scores"
6010 if ds = 62 then begin
6015 :::dclose#1
6020 :::dopen#1,(nc$) + "'s Scores",w
6025 bend
6030 print#1,s$:rem sign
6035 print#1,dt:rem # of digits
6040 print#1,cr:rem # of correct answers
6045 print#1,wr:rem # of wrong answers
6050 dclose#1
6055 :
6060 run "math.menu"
6065 :
6070 :
7000 rem **--convert to lowercase--**
7005 for cv = 1 to len(cv$)
7010 x = asc(mid$(cv$,cv,1))
7015 if x > 192 then x = x - 128
7020 nc$ = nc$ + chr$(x)
7025 next cv
7030 :
7035 cv$ = nc$
7040 f = asc(left$(cv$,1))
7045 f = f + 128
7050 nc$ = chr$(f) + right$(cv$,len(cv$)- 1)
7055 return
7060 :
7065 rem na$ = originally typed name
7070 rem cv$ = converted to lowercase
7075 rem nc$ = 1st letter/uppercase
7080 :
7085 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print a$;
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
```

The math.scores Program

```
  100 rem ***--math.scores--***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white =  2
  130 :::red = 11
  135 yellow =  8
  140 ::chtr =  5
  145 bckgnd =  0
```

```
  150 :
  200 dim s$(100),dt(100),cr(100),wr(100)
  205 print chr$(14):rem upper/lower case mode
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 :
  225 :
  300 rem **--user message--**
  305 scnclr:color chr,yellow
  310 char 0,20,5,"Student's name please: "
  315 gosub 19000:rem getkey subroutine
  320 cv$ = t$:t$ = ""
  325 gosub 7000:rem convert to lower case
  330 gosub 10000:rem type of display routine
  335 scnclr:char 0,20,5
  340 print chr$(15);"ONE MOMENT PLEASE!";chr$(143)
  345 :
  350 :
  400 rem **--read file routine--**
  405 :
  410 rem *--data file--*
  415 k = 1
  420 dopen#1,(nc$) + "'s Scores"
  425 input#1,s$(k):rem sign
  430 input#1,dt(k):rem # of digits
  435 input#1,cr(k):rem # of correct answers
  440 input#1,wr(k):rem # of wrong answers
  445 k = k + 1
  450 if status = 0 then 425
  455 dclose#1
  460 :
  465 :
  500 rem **--display scores--**
  505 gosub 6000:rem titles routine
  510 j = 1
  515 for i = 1 to k - 1
  520 if s$(i) = "+" then s$(i) = "ADD"
  525 if s$(i) = "-" then s$(i) = "SUB"
  530 if s$(i) = "X" then s$(i) = "MLT"
  535 if s$(i) = "/" then s$(i) = "DIV"
  540 if i < 10 then print#file,"";tab(17) i;:goto 550
  545 print#file,"";tab(16) i;
  550 print#file,"";tab(29) s$(i);
  555 print#file,"";tab(39) dt(i);
  560 if cr(i) > 9 then u = -1
  565 print#file,"";tab(49 + u) cr(i);
  570 u = 0
  575 if wr(i) > 9 then u = -1
  580 print#file,"";tab(58 + u) wr(i)
  585 u = 0
  590 j = j + 1
  595 if j > 15 then gosub 20000:gosub 6000
  600 next i
  610 char 0,15,22
  615 gosub 20000:rem continue routine
  620 :
  625 :
  800 rem **--return to program menu--**
  805 scnclr
  810 char 0,15,9
  815 color chtr,white
  820 print "LOADING THE MATH MENU PROGRAM"
  825 color chtr,yellow
  830 run "math.menu"
  835 :
  840 :
 6000 rem **--titles routine--**
 6005 scnclr
 6010 char 0,38,1,nc$
```

```
6020 char 0,15,3,"SESSION"
6025 char 0,26,3,"OPERATION"
6030 char 0,38,3,"DIGITS"
6035 char 0,47,3,"CORRECT"
6040 char 0,57,3,"WRONG"
6042 print
6045 return
6050 :
6055 :
7000 rem **--convert to lowercase--**
7005 for cv = 1 to len(cv$)
7010 x = asc(mid$(cv$,cv,1))
7015 if x > 192 then x = x - 128
7020 nc$ = nc$ + chr$(x)
7025 next cv
7030 :
7035 cv$ = nc$
7040 f = asc(left$(cv$,1))
7045 f = f + 128
7050 nc$ = chr$(f) + right$(cv$,len(cv$)- 1)
7055 return
7060 :
7065 rem na$ = originally typed name
7070 rem cv$ = converted to lowercase
7075 rem nc$ = 1st letter/uppercase
7080 :
7085:
10000 rem **--display subroutine--**
10005 scnclr
10010 char 0,0,3
10015 print "Would you like a paper print out? ";
10020 gosub 18000:rem y/n input subroutine
10025 if yes$ = "y" then 10040:rem printer
10030 if yes$ = "n" then 10115:rem screen
10035 :
10040 rem *--print out is desired--*
10045 scnclr
10050 char 0,0,3
10055 print "Please make sure the printer is on and ready to use."
10060 print
10065 print "Are you ready to begin printing? ";
10070 gosub 18000:rem y/n input subroutine
10075 if yes$ = "n" then 10000:rem ask again
10080 :
10085 rem *--printer display--*
10090 file = 4
10095 dvic = 4
10100 cmnd = 7
10105 goto 10500:rem open instruction
10110 :
10115 rem *--screen display--*
10120 file = 3
10125 dvic = 3
10130 cmnd = 1
10135 goto 10500:rem open instruction
10140 :
10145 :
10500 rem *--open instruction--*
10505 open file,dvic,cmnd
10510 return
10515 :
10520 :
11000 rem **--line up numbers--**
11005 if i < 10 then ta = 4
11010 if i > 9 and i < 100 then ta = 3
11015 if i > 99 and i < 1000 then ta = 2
11020 if i > 999 then ta = 1
11025 return
11030 :
```

```
11035 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 print
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print a$;
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
19070 :
20000 rem **--continue subroutine--**
20005 print#file,""
20010 close file
20015 char 0,15,22
20020 print "Press ";
20025 color chtr,white
20030 print "RETURN";
20035 color chtr,yellow
20040 print " to continue: ";
20045 gosub 19000:rem getkey subroutine
20050 print
20055 return
```

Chapter 9

Relative Files

The biggest barrier I have found in explaining relative-access files is fear. People are afraid that relative access is too hard to learn. Actually, once you understand the principles behind sequential access, learning to work with relative access is not that difficult. Please do not become intimidated by the different approach relative access requires.

If you have skipped the sequential-access chapters and are just starting this book, there are a few points that must be understood. For those who have read all the previous chapters, the following information is repeated from Chapter 3.

Data files have two ways of storing and retrieving information. (Remember that the information really stays on the diskette and you are just getting a copy of the information when you retrieve it.) These two ways of storing and retrieving information are *sequential access* and *direct access*. A sequential-access data file basically means that the information stored in the file is kept in sequential order—that is, one right after the other. A direct-access data file usually means that each part of the file is divided equally and can be reached directly and at random instead of going through all previous records. The process of looking at each record in order (sequence) to decide if it is the record you want is a characteristic of sequential files, and can require more time than the direct-access method.

The basic difference between sequential data files and direct-access data files is somewhat like the difference between a cassette tape and a phonograph record. If you want to find a specific song on a cassette tape, even using the best available tape deck, you must begin at the current location of the tape and proceed either forward or backward, passing over all intervening songs until you have found the song you want. The process proceeds in sequence, one song after another. For example, if you want to play the fourth song on the tape, you have to advance the tape through the first, second, and third songs until you get to it.

On the other hand, if the songs were on a phonograph record, all you would have to do to play the fourth song would be to place the phonograph cartridge containing the needle at the start of the fourth division, instead of at the start of the first song. You can do that because you are able to clearly see the divisions between songs and because those individual songs are directly accessible. You do not have to go through the grooves of the first three songs in order to get to the fourth, and moving the needle by hand only takes seconds.

Using this example, imagine a DATA drawer containing two basic divisions: the first division contains files that operate in a way similar to cassette tapes, while the second division contains files that operate like phonograph records. (Commodore refers to direct-access files as *relative* files, but the term "relative" in the computer world often means "in relation to" and can be misleading when used as the name of the method for directly accessing records within a file. It is important to remember that, from now on, whenever I use the term *relative access*, I am talking about *direct-access* files.

Despite their differences, these two kinds of data files do have things in common, just like tapes have things in common with phonograph records. The most obvious common characteristic is that they both usually contain information that is not in the form of instructions for the computer. In other words, they contain information like lists, addresses, receipts, and inventories. Second, both files make use of some of the same BASIC file commands, but with different *parameters*.

Because these data files are not computer instructions, they cannot be used in the same manner as BASIC program files. In other words, you cannot RUN, DSAVE, or DLOAD a data file. Those three commands, when combined with a file name, are the computer's means of access to BASIC diskette files. The obvious question, then, is that if RUN, DSAVE, or DLOAD cannot be used with data files, how does the computer get the information on the disk in a data file, or back off the diskette from a data file?

In order to gain access to data files, you must use certain BASIC file commands in specific ways, depending on the kind of data file you are accessing. Both sequential and relative-access data files primarily use four types of commands: (1) OPEN, (2) CLOSE, (3) some way of reading the file (INPUT# or GET#), and (4) some method of writing to the file (PRINT#).

Again, the example of the filing cabinet is useful. In much the same way that a secretary must open a file folder removed from the filing cabinet before making use of any information contained in the file, so also must all data files be opened before the information they contain can be put to use. And, just as the secretary should close the file folder before replacing it in the filing cabinet, so also should all data files be closed before ending the program or turning off the computer. If a secretary does not close the file folder, some information might drop out and get lost. The same is true if data files are not properly closed. This is usually only the case after new information has been written to the file and the file not closed. Loss of information should not occur if a data file has only been read.

INPUT#, PRINT#, or GET# are the main processes by which information is either read from or written to the file. If you only want to see information already in a data file, the BASIC file commands INPUT# and GET# are the commands you would use. If you want to add information to the file or create a new file, use the BASIC file command PRINT#.

RELATIVE-ACCESS FILES

There are usually two kinds of relative-access files: (1) relative files that consist of un-

divided equal-length records, and (2) relative files that consist of divided equal-length records. Notice the only difference is that in one kind, the records are divided into parts, while in the other kind, the records remain as a whole. This latter kind is the easier to explain and use (but much less useful), so I will discuss this type first.

Undivided Relative Files

The main difference between sequential files and undivided relative files is that with relative files you can access any record in any order. With sequential files, it is necessary to begin with the first record and proceed past all intervening records until you come to the record you want. With relative files, you are able to go directly to the desired record.

At this point, a short tutorial might best explain the exact syntax and operation of relative-access files. Take a formatted diskette that has plenty of room on it or format a new one (refer to Chapter 2 if necessary). Enter the following program (in 80 column, lowercase mode):

```
100 rem ***--relative access file example--***
105 :
110 :
115 rem **--file output routine--**
120 dopen#1,"Relative Example",l50
125 for i = 1 to 25
130 :
135 record#1,(i)
140 record#1,(i)
145 print#1,"Record = #";i
170 :
175 next i
180 dclose#1
185 :
190 :
```

In this example, you can see how easy it is to use relative files of the undivided kind. Once a file has been opened, you can either read information from it or write information to it, without the sequential-access requirement of closing the file between tasks. You must, however, specify the length of each record within the file. Once you open the file properly, you are able to directly read or write to any record you choose in any order you choose. Please notice the qualification about opening the file properly.

Both kinds of relative files must consist of equal length records. This means that you must decide on the length of the longest record you will ever have. You do not need to specify this number in sequential files, because the next record begins immediately after the end of the last character and the record delimiter, no matter what the actual length of the record. You must specify the maximum length in relative files because the next record does not begin immediately after the last character in the previous record. It actually begins the specified record length after the beginning of the previous record, regardless of the number of characters in that record, as illustrated in the examples below:

```
Sequential Access Storage:
1234567890123 123456789012 12345678901234
David Miller, Mark Miller, Paul D. Miller....etc.
```

```
Relative Access Storage (Length of 15 Characters per record):
123456789012345 123456789012345 123456789012345
David Miller--- Mark Miller---- Paul D. Miller-....etc.
```

In the program sample, the length (l) in line 120 is given as 50. That means that each record has a maximum of 50 characters possible, and that each record begins 50 characters, or bytes, from the start of the previous record. If the first record begins at byte 0, the second begins at 50, the third at 100, etc. You do not need to be concerned with the actual location on the disk, but it is important to understand that each record must be of equal length, so it is very easy for the computer to calculate the starting position of each record and for you to specify any record in any order. If, for example, you indicate that you want access to record number 10, the computer multiplies the number of records—10—times the length of each record—50—and immediately jumps to the 500th byte of the file. You must provide the computer with the maximum length by assigning a value after the "l" parameter in relative files. The number given after the "l" in a DOPEN statement indicates the maximum number of characters, or bytes, you expect in *any* record in that file. It also indicates that *each* record will be that number of characters, or bytes, long.

If you have a record that is not as long as the number given after the "l," you will have a certain amount of disk space that is unused, so it is important to figure carefully and keep the number after "l" as low as possible. If the number is very large but most of your records are rather small, you will waste a lot of disk space (as in the above example). A certain amount of wasted disk space is inevitable in using relative files, because few files will contain information of exactly equal length. In using relative files, you are willing to waste a little disk space in order to gain much faster disk access.

Add the following lines:

```
195 input "Press the 'RETURN' key to continue: ";key$
200 :
205 :
300 rem **--file input routine--**
305 dopen#1,"Relative Example",150
310 for i = 1 to 25
315 :
320 record#1,(i)
325 record#1,(i)
330 input#1,a$
355 :
360 print a$
365 next i
370 dclose#1
375 :
380 :
```

In order to read back what was just written, you reopen the file in line 305 (because the file has been closed in line 180) and specify the record number in lines 320 and 325 as the value of i. Otherwise, this input and display routine looks very similar to what a sequential file routine would look like. The main difference is the addition of lines 320 and 325, which indicate the specific record that is being accessed. Commodore recommends

the use of *two* record instructions every time. I tried using just one record command and found that everything seemed to work properly, but with additional testing, and, specifically, with rapid access, I began to get repeated disk errors. The apparent solution is the use of more record commands. (The program at the end of this chapter uses only one record command for each disk access to demonstrate that a single command does work, but I recommend that if you begin getting disk access errors, the first solution you try is to add more record instructions.)

The RECORD command has three possible parameters: (1) the file number, (2) the record number, and (3) the position within the record. In the above example of undivided relative-access records, only the first two parameters are included. The third parameter is used for divided relative-access records. As with the DOPEN command, variables may be used with the RECORD command, but the variables must be enclosed by parentheses.

As you can see, there is not much difference in learning to use sequential-access files and learning to use undivided relative-access files. There also is not much difference in the usefulness of these undivided relative files. Far more useful is the second kind of relative access files—those with divided equal-length records.

Divided Relative Files

The divided relative file consists of records that are broken into varying-length parts, or *fields*. Each record is the same length, but within each record, the fields, or parts of the record, can be of varying lengths. In other words, a relative-access file that consists of records all with a length of 100 bytes can have each record divided into fields of unequal lengths. The first field might be 25 characters (bytes) long, the second field 10 bytes, the third field 15 bytes, and the last field 50 bytes. The total number of bytes equals 100, but no two fields need be the same length. Each record in the file would have the same make-up. Therefore, the file would look something like the following:

Record #	Field #1 (25 Bytes)	Field #2 (10 Bytes)	Field #3 (15 Bytes)	Field #4 (50 Bytes)
1	.....	.....	.....	.....
2	.....	.....	.....	.....
3	.....	.....	.....	.....
4	.....	.....	.....	.....
5	.....	.....	.....	.....
6	.....	.....	.....	.....
etc.				

In a Mailing List System, for example, with relative files that contain divided records, you could specify a certain length for the first name, and other lengths for the last name, city, zip code, etc. Each record, therefore, could contain a complete set of the information needed for the Mailing List System. For instance, if you decide that each line of information or each record would have no more than 150 characters in it, you could further decide that the first field of each record would exist from byte 0 to byte 10, the second from byte 11 to byte 15, the third field from byte 16 to byte 30, and so on.

You could also decide that each record would contain the first name in the first field, the middle initial in the second field, the last name in the third field, the numerical address in the fourth field, the street address in the fifth field, any apartment number in the

sixth field, the city in the seventh field, the state in the eighth field, and the zip code in the ninth field. Under this setup, it would be easy to access any part of any record in any order desired. For example, if you just wanted the zip code and first name in that order, you would have no trouble accessing just that information.

I have been using the term "byte" in connection with the word "character" so that you might get used to the idea that the length of a record is measured in bytes. Each character or number is one byte. If a file has equal length records of 50, that is 50 bytes. If the second field begins 27 characters from the first character, that is the 27th byte of the file. To access that byte, you use another parameter with the record command.

To continue the tutorial, add the following instruction lines to the example program.

```
150 :
155 record#1,(i)
160 record#1,(i),25
165 print#1,"abcdefgh = #";i

335 :
340 record#1,(i)
345 record#1,(i),25
350 input#1,b$
360 print a$,b$
```

The example program should then have the following instructions:

```
100 rem ***--relative access file example--***
105 :
110 :
115 rem **--file output routine--**
120 dopen#1,"Relative Example",l50
125 for i = 1 to 25
130 :
135 record#1,(i)
140 record#1,(i)
145 print#1,"Record = #";i
150 :
155 record#1,(i)
160 record#1,(i),25
165 print#1,"abcdefgh = #";i
170 :
175 next i
180 dclose#1
185 :
190 :
195 input "Press the 'RETURN' key to continue: ";key$
200 :
205 :
300 rem **--file input routine--**
305 dopen#1,"Relative Example",l50
310 for i = 1 to 25
315 :
```

```
320 record#1,(i)
325 record#1,(i)
330 input#1,a$
335 :
340 record#1,(i)
345 record#1,(i),25
350 input#1,b$
355 :
360 print a$,b$
365 next i
370 dclose#1
375 :
380 :
```

Commodore refers to the field position, or byte parameter, as the *offset* parameter. Therefore, the 25 in lines 160 and 345 indicate that the second piece of information is to begin at the 25th space of the record. Without the offset parameter in lines 140 and 340, the computer writes and reads the first piece of information beginning at the first byte of the record. The information that is to be written to the first and 25th locations is given in the lines that follow each of the record instructions (lines 145 and 165). In line 145 that information consists of the phrase "Record = #" and the value of the variable i. In line 165, the information consists of the characters "abcdefgh = #" and the value of the variable i. Now, the diskette file will have the following format:

```
Record # Field #1                  Field #2
         (25 Bytes)                (25 Bytes)
1        Record = #1.............abcdefgh = #1.............
2        Record = #2.............abcdefgh = #2.............
3        Record = #3.............abcdefgh = #3.............
4        Record = #4.............abcdefgh = #4.............
5        Record = #5.............abcdefgh = #5.............
6        Record = #6.............abcdefgh = #6.............
7        Record = #7.............abcdefgh = #7.............
8        Record = #8.............abcdefgh = #8.............
9        Record = #9.............abcdefgh = #9.............
10       Record = #10............abcdefgh = #10............
11       Record = #11............abcdefgh = #11............
12       Record = #12............abcdefgh = #12............
13       Record = #13............abcdefgh = #13............
14       Record = #14............abcdefgh = #14............
15       Record = #15............abcdefgh = #15............
16       Record = #16............abcdefgh = #16............
17       Record = #17............abcdefgh = #17............
18       Record = #18............abcdefgh = #18............
19       Record = #19............abcdefgh = #19............
20       Record = #20............abcdefgh = #20............
21       Record = #21............abcdefgh = #21............
22       Record = #22............abcdefgh = #22............
23       Record = #23............abcdefgh = #23............
24       Record = #24............abcdefgh = #24............
25       Record = #25............abcdefgh = #25............
```

The file-input routine is essentially the same as the file-output routine. The only real difference between the two routines is that in the file input routine, information is obtained from the diskette and then displayed on the screen instead of written to the diskette. Add the following instructions to the program:

```
500 rem **--random access of relative file--**
505 dopen#1,"Relative Example",l50
510 input "Which record number ";nb$
515 nb = val(nb$)
520 if nb = 0 then 900:rem end
525 if nb > 25 then 510:rem ask again
530 :
535 record#1,(nb)
540 record#1,(nb),25
545 input#1,b$
550 :
555 record#1,(nb)
560 record#1,(nb),1
565 input#1,a$
570 :
575 print b$,a$
580 goto 510
585 :
590 :
900 rem **--end routine--**
905 dclose#1
910 end
```

The full program now is:

```
100 rem ***--relative access file example--***
105 :
110 :
115 rem **--file output routine--**
120 dopen#1,"Relative Example",l50
125 for i = 1 to 25
130 :
135 record#1,(i)
140 record#1,(i)
145 print#1,"Record = #";i
150 :
155 record#1,(i)
160 record#1,(i),25
165 print#1,"abcdefgh = #";i
170 :
175 next i
180 dclose#1
185 :
190 :
195 input "Press the 'RETURN' key to continue: ";key$
```

```
200 :
205 :
300 rem **--file input routine--**
305 dopen#1,"Relative Example",l50
310 for i = 1 to 25
315 :
320 record#1,(i)
325 record#1,(i)
330 input#1,a$
335 :
340 record#1,(i)
345 record#1,(i),25
350 input#1,b$
355 :
360 print a$,b$
365 next i
370 dclose#1
375 :
380 :
500 rem **--random access of relative file--**
505 dopen#1,"Relative Example",l50
510 input "Which record number ";nb$
515 nb = val(nb$)
520 if nb = 0 then 900:rem end
525 if nb > 25 then 510:rem ask again
530 :
535 record#1,(nb)
540 record#1,(nb),25
545 input#1,b$
550 :
555 record#1,(nb)
560 record#1,(nb),1
565 input#1,a$
570 :
575 print b$,a$
580 goto 510
585 :
590 :
900 rem **--end routine--**
905 dclose#1
910 end
```

This routine provides the first real example of what relative access is all about. The user can indicate any record number between 1 and 25 in line 510, and the computer will go to that record number and obtain the information in whatever order the programmer has specified. Notice that the record statements have been turned around. The first information obtained and displayed is the information stored beginning at the 25th byte, or space, of the record.

The record numbers do not have to be given in order. You can ask for record 21 and then ask for record 2, or ask for record 9 and then record 19. The order does not matter,

because the computer has been instructed to go directly to the specified record and the specified location within that record.

When you have finished entering the program instructions, save them to the diskette as "relative test."

```
dsave"relative test  {RETURN}
```

Then, run the program:

```
run  {RETURN}
```

For a short time, the disk drive green light should come on, and the cursor should disappear. Eventually, the statement from line 195 will be displayed on the screen and the green light will go out. When you are ready, press the RETURN key and watch the screen carefully. Two columns should appear. If you do not get two columns on the same line, check line 360 for the comma. The left-most column should contain the phrase "Record = #" and an increasing number, up to the number 25. The right-most column should contain the characters "abcdefgh = #" and the same increasing number, up to 25. When the number 25 has been reached, the statement from line 510 should be displayed. The screen should look something like the following:

```
Record = #1     abcdefgh = #1
Record = #2     abcdefgh = #2
Record = #3     abcdefgh = #3
Record = #4     abcdefgh = #4
Record = #5     abcdefgh = #5
Record = #6     abcdefgh = #6
Record = #7     abcdefgh = #7
Record = #8     abcdefgh = #8
Record = #9     abcdefgh = #9
Record = #10    abcdefgh = #10
Record = #11    abcdefgh = #11
Record = #12    abcdefgh = #12
Record = #13    abcdefgh = #13
Record = #14    abcdefgh = #14
Record = #15    abcdefgh = #15
Record = #16    abcdefgh = #16
Record = #17    abcdefgh = #17
Record = #18    abcdefgh = #18
Record = #19    abcdefgh = #19
Record = #20    abcdefgh = #20
Record = #21    abcdefgh = #21
Record = #22    abcdefgh = #22
Record = #23    abcdefgh = #23
Record = #24    abcdefgh = #24
Record = #25    abcdefgh = #25
Which record number ?
```

You will be asked to indicate which record number you wish to see. Indicate record

number seven. This time, you should see two columns, but in reverse order. The characters "abcdefgh = #7" should be in the left-most column, and the phrase "Record = #7" should be in the right hand column:

```
abcdefgh = #7  Record = #7
Which record number ?
```

At this point, you can try out additional record numbers. I would also encourage you to experiment with different combinations of these instructions and parameters, just to see what happens.

There are a few things to watch out for in working with relative files with the offset parameter. The most important item is that when writing records to the diskette, the offset parameter (or field position) must be written in order and all fields must be specified before the next record is written. (As just demonstrated, this is not the case when simply reading the file.) In other words, you cannot write information to the third field and then write information to the first field. Neither can you update just one field unless it is the last field.

Apparently, Commodore fills the remaining space in a record with the character "@" (chr$(255)). Therefore, if you attempt to update a single field, all fields following the updated field contain chr$(255) characters. The solution is to write out fields in order and make certain that all fields are included any time one or more fields have been changed.

You might also find that occasionally the Commodore indicates an error message when clearly no such error message should be displayed. For example, in the above relative example program, when I rapidly gave the computer different record numbers to obtain and display, every once in a while, the computer would display the "device not present" error. Obviously, the device (the disk drive) was present, and all I had to do was instruct the computer to "goto 500" and reenter the same record number. The computer always responded properly, indicating that the programming was correct, and all cable connections were secure. This problem might exist in just the system I am using, or it might be part of the normal functioning of the Commodore 128 operating system. Because Commodore includes the "Safety Note" on page 56 of the Commodore 1571 disk drive user's guide, regarding the necessity of specifying the record command twice, and seems to insist that it is necessary to check the error channel after every disk access, I believe the problem I encountered is not just with my equipment. I found some additional problems when doing the programming for the next two chapters, but with one exception, everything else is solved with detailed programming.

The one exception is that sometimes a record appears to have been written when in fact it never reaches the diskette, and no error message indicates that fact. The only solution is to rewrite the record.

You now have the complete structure of the usual relative-access file. Diskettes are used to store *files*. Files contain *records*. Records are made up of *fields* of information. To go back to the file cabinet example, the filing cabinet is used to store files. Files contain pages. Each page contains some information. The examples are not completely parallel, but the concepts are similar.

There are other factors involved with relative-access files, but the main thing to understand is the concept that information, regardless of what it is, has to be organized in a predetermined manner. Information that does not fit the predetermined manner must be made to fit, or that information cannot be included in the file. For example, if it has

been determined that all names to be added to a file will have no more than 10 characters, any name that exceeds 10 characters will need to be abbreviated. This example points up the importance of careful planning prior to creating a relative-access file.

Sequential-access files often do not require much preplanning. A programmer can simply set up the file and dump information into that file. Sequential files are easy to create, but often are either very hard or impossible to get useful information out of. Relative files, on the other hand, are more difficult to create, but if they are created properly, the information in the files can be taken out and displayed in just about any manner desired. Because the reason for storing information is to be able to make use of that information, most useful programs work with divided relative-access files rather than sequential files. In other words, it is more important to easily access your information than it is to easily store your information. Again, the file cabinet can be used to illustrate this concept.

It is easier to simply pile all your papers, one on top of the other, in a large box, than it is to organize your file cabinet properly, and then sort and file all your papers. But when you want to find a specific paper, it will usually take far less time to go to the file cabinet, find the right folder, and remove the desired paper than it will to go to a large box and look one by one through all your papers until you find the one you want.

THE MEDICAL RECORDS SYSTEM

With this background, let's go over a useful program with a file used to store personal family medical records. In these relative file examples, I will not go over all the routines as I did in the sequential file examples. Instead, I will concentrate on the file routines. The complete listing for the program will be found at the end of the chapter.

If you take a look at the complete listing, you will see that it begins with a menu routine. The first thing that needs to be done the first time the program is used is to set the value of the record pointer to zero and write that value out to a sequential file. After setting the pointer value that first time, the program instructions automatically update the pointer value. The record pointer file is used to keep track of the total number of records in the relative file.

Next, a keyboard input routine is used to obtain the original information. The user is asked to supply the name of the individual (name$), the date (dte$), the type of record (that is, whether it is a record of a doctor's visit, medication, illness, accident or injury, shot or immunization, or x-ray) (type$), and any miscellaneous information, such as the name of the medication and frequency of use, the kind of illness, or the location of the injury (misc$).

The File Output Routine

Once you have entered all of the information and have verified that it is correct, you are ready to write that information out to the diskette file.

```
1485 rem **--file output routine--**
1490 :
1495 rem *--get # of records from pointer file--*
1500 dopen#1,"Med.Rec.Ptr"
1505 input#1,ptr
1510 dclose#1
1515 ptr = ptr + 1
1520 :
```

```
1525 rem *--data file--*
1530 dopen#2,"Medical Records",150
1535 :
1540 record#2,(ptr),1
1545 print#2,name$
1550 :
1555 record#2,(ptr),15
1560 print#2,dte$
1565 :
1570 record#2,(ptr),26
1575 print#2,type$
1580 :
1585 record#2,(ptr),28
1590 print#2,misc$
1595 :
1600 dclose#2
1605 :
1610 rem *--update pointer file--*
1615 dopen#1,"@Med.Rec.Ptr",w
1620 print#1,ptr
1625 dclose#1
1630 :
1635 goto 500:rem menu
1640 :
1645 :
1650 :
```

There are three parts to this output routine. The first part (lines 1495-1515) accesses the pointer file and updates the pointer value. The second part (lines 1525-1600) defines the data file and writes the information to it. Finally, the last part (lines 1610-1625) again accesses the pointer file, this time to write out the new value of the pointer. The second part is of most importance, because that is the part that deals with a relative-access file.

One point should be emphasized before continuing. It is not necessary to use string arrays, such as name$( ). You do not have to use string arrays because of the versatility of random files. In this program, the information for a complete record is written to the diskette before additional information is obtained from the user. The idea that you can use the disk drive and diskette without extensive use of string arrays will become more apparent in the section on reading and displaying medical information.

The File Input Routine

The following section of the program enables the user to see the information stored in the Medical Records file. In this first section, you read the file and immediately display the information.

```
2000 rem **--read file routine--**
2005 gosub 10000:rem display routine
2010 if file = 4 then gosub 22000:rem titles routine
2015 :
2020 rem *--pointer file--*
```

```
2025 dopen#1,"Med.Rec.Ptr"
2030 input#1,ptr
2035 dclose#1
2040 :
2045 rem *--titles--*
2050 scnclr:print chr$(2);:rem underline
2055 char 0, 0,0,"NAME"
2060 char 0,20,0,"DATE"
2065 char 0,35,0,"TYPE"
2070 char 0,55,0,"MISC"
2075 print chr$(130):rem underline off
2080 window 0,2,79,24:rem set window/fix titles
2085 :
2090 rem *--data file--*
2095 dopen#2,"Medical Records"
2100 for rec = 1 to ptr
2105 :
2110 record#2,(rec),1
2115 input#2,name$
2120 :
2125 record#2,(rec),15
2130 input#2,dte$
2135 :
2140 record#2,(rec),26
2145 input#2,type$
2150 :
2155 record#2,(rec),28
2160 input#2,misc$
2165 :
2170 :
2175 rem *--display information--*
2180 gosub 16000:rem get full type message
2185 if file = 4 then gosub 23000:rem printer display
2190 char 0, 0,rec,(name$)
2195 char 0,20,rec,(dte$)
2200 char 0,35,rec,(tp$)
2205 char 0,55,rec,(misc$)
2210 next rec
2215 :
2220 dclose#2
2225 gosub 15000:rem pause routine
2230 window 0,0,79,24:rem reset window
2235 close file
2240 goto 500:rem menu
2245 :
2250 :
2255 :
```

The first thing that is done is to name the routine (line 2000) and ask if the user wants a paper printout (line 2005). Next, the pointer file is accessed, and the value of the pointer is loaded into the computer. The number of records currently in the data file is stored

in the numeric variable ptr. It is not necessary to immediately close the pointer file (Med.Rec.Ptr), but it is a good programming habit.

The titles used in the display of information must be set up outside the loop used to obtain the data from the diskette. These titles are all underlined and positioned on the same line across the screen. This "titles" line is then protected by establishing the display window below the screen line used for the titles. Therefore, no matter how many lines of information the file contains, the titles line will never disappear from view.

The data file, Medical Records, is opened in line 2095. Notice that this time the length parameter is missing and therefore not necessary. Once a relative file has been created with a specific length, the computer assumes the same length with any future reference to that file. In most computer systems, the type of file (sequential or relative) is evident by the statement used to open the file, but Commodore has eliminated the necessity of including parameters that indicate whether the file is sequential or relative, after the file has been created. Instead, evidence of the type of file occurs with the RECORD command, which is not used with sequential files.

Lines 2110 to 2160 are similar to lines 1540 to 1590 except that information is read from the file with these instructions instead of written to the file. Once values have been established for all the desired variables, the display portion of the routine is executed—lines 2175-2205. The entire routine is repeated until the complete file has been read and displayed.

The subroutine at 16000 has a very specialized purpose. Its purpose is to match the single character symbol with its complete corresponding "type" name: for example, it must exchange "d" for "Dr. Visit." Once the exchange has been made, control is returned to the statement immediately following the GOSUB statement. Then, instead of displaying the value of the variable brought in from the diskette, it displays the value of the variable tp$ obtained from the subroutine. This technique can be used for information that is often repeated in a database. In essence, you have stored a code instead of the actual information. Using a code whenever possible saves diskette space and increases the speed of operation since less information is written to or read from the diskette.

The Search Routine

The only section of the program left to examine is the search routine. Lines 3000 to 3155 establish exactly what to search for, and lines 3165 through 3480 conduct the actual search and display the results.

```
3165 rem **--file input routine--**
3170 :
3175 rem *--pointer file--*
3180 dopen#1,"Med.Rec.Ptr"
3185 input#1,ptr
3190 dclose#1
3195 :
3200 rem *--data file--*
3205 dopen#2,"Medical Records"
3210 ln = len(srch$)
3215 :
3220 for rec = 1 to ptr
3225 record#2,(rec),(pztn)
```

```
3230 input#2,find$
3235 :
3240 rem *--convert to lowercase--*
3245 cv$ = left$(find$,ln)
3250 gosub 17000:rem convert routine
3255 if srch$ <> cv$ then 3415:rem next record
3260 :
3265 rem *--matched, now get rest of info.--*
3270 for k = 1 to 4
3275 if k = 1 then byte =  1
3280 if k = 2 then byte = 15
3285 if k = 3 then byte = 26
3290 if k = 4 then byte = 28
3295 record#2,(rec),(byte)
3300 input#2,info$(k)
3305 next k
3310 :
3315 type$ = info$(3)
3320 gosub 16000:rem get full type message
3325 info$(3) = tp$
3330 :
3335 :
3340 rem *--display information--*
3345 scnclr
3350 color chtr,white
3355 char 0,col,row,"MEDICAL INFORMATION"
3360 color chtr,drkyel
3365 print
3370 :
3375 print:print#file,""tab(col) "1.  Name: ";info$(1)
3380 print:print#file,""tab(col) "1.  Name: ";info$(2)
3385 print:print#file,""tab(col) "1.  Name: ";info$(3)
3390 print:print#file,""tab(col) "1.  Name: ";info$(4)
3395 print:print#file,""tab(col) " "
3400 gosub 15000:rem pause routine
3405 :
3410 :
3415 next rec
3420 :
3425 :
3430 dclose#2
3435 scnclr
3440 color chtr,white
3445 char 0,col,row,"Search Completed!"
3450 color chtr,drkyel
3455 gosub 15000:rem pause routine
3460 close file
3465 goto 3000:rem search again
3470 :
3475 :
3480 :
```

This is an elementary search and display routine. Lines 3175 to 3190 open the pointer file and obtain the value of the pointer. The data file is then opened (line 3205) and defined.

Next comes a technique that allows the user to search for just the beginning portion of a field in case he or she does not know the complete spelling of the entire field. The search is limited to the number of characters the user has supplied in answer to the question in line 3055 (see the program listing at the end of this chapter). This number is determined (line 3210) and then the number in line 3245 is used to limit the number of characters that will go through the process of conversion to lowercase. Once those characters have been converted to lowercase, they can be compared to the characters supplied by the user (srch$). If they are not equal, the computer is instructed to increment the value of rec and proceed.

Line 3220 establishes the boundaries for a loop. Within that loop, you look for the desired part of each record. When that part is located, the rest of the information associated with that part is read in (lines 3265 to 3305) and displayed (lines 3340 to 3400). Those lines are skipped for information that does not match or equal the string variable for which you are searching (line 3255). When the entire file has been searched, the file is closed, and control is transferred back to the beginning of the search routine to see if the user wishes to search for more information.

This program provides a reasonable example of techniques involved with creating, adding to, and reading from a relative-access file. It does not get too fancy, yet it is a useful program. You might want to supply additional routines.

At this point, you should find yourself capable of reading a program listing. Therefore, as you progress through the remainder of this book, the amount of text will decrease, while the number of program instructions that you should read increases. In Chapter 10, relative files will be used in a more elaborate manner.

One additional comment needs to be made in concluding this chapter. There are a variety of ways of using relative-access files of either the divided or undivided kind. The method presented in this chapter is not meant to suggest itself as the only method or even the best method. It is a method that does work and is understandable. In working with files, I have found that comprehension is of more value than program efficiency or speed.

A complete listing of the programs in this chapter is presented in Fig. 9-1.

CHAPTER 9 QUESTIONS

1. Name the two kinds of relative files.
2. True or False: Relative files can contain records of different lengths.
3. What are the two BASIC words used to obtain information from a diskette and place information on a diskette?
4. True or False: In relative files, the next record begins immediately after the last character in the previous record.
5. What parameter must a DOPEN command have in a relative file?
6. True or False: Relative files waste diskette space but have much faster disk access than do sequential files.
7. True or False: The relationship that exists between the various parts of a divided relative-access file can be defined in the following way: A relative-access file consists of equal-length records. Each record may consist of equal and/or unequal length fields. The number following the l parameter in a DOPEN statement indicates the length of each record.

8. What is the length of each record measured in?
9. True or False: Relative files require greater use of string arrays than do sequential files.
10. What BASIC word is used to locate individual records in a relative file?
11. What parameter is used to locate individual fields within a relative file?

Fig. 9-1. The Medical Records System program.

Relative-Access File Example

```
100 rem ***--relative access file example--***
105 :
110 :
115 rem **--file output routine--**
120 dopen#1,"Relative Example",l50
125 for i = 1 to 25
130 :
135 record#1,(i)
140 record#1,(i)
145 print#1,"Record = #";i
150 :
155 record#1,(i)
160 record#1,(i),25
165 print#1,"abcdefgh = #";i
170 :
175 next i
180 dclose#1
185 :
190 :
195 input "Press the 'RETURN' key to continue: ";key$
200 :
205 :
300 rem **--file input routine--**
305 dopen#1,"Relative Example",l50
310 for i = 1 to 25
315 :
320 record#1,(i)
325 record#1,(i)
330 input#1,a$
335 :
340 record#1,(i)
345 record#1,(i),25
350 input#1,b$
355 :
360 print a$,b$
365 next i
370 dclose#1
375 :
380 :
500 rem **--random access of relative file--**
505 dopen#1,"Relative Example",l50
510 input "Which record number ";nb$
515 nb = val(nb$)
520 if nb = 0 then 900:rem end
525 if nb > 25 then 510:rem ask again
530 :
535 record#1,(nb)
540 record#1,(nb),25
545 input#1,b$
550 :
555 record#1,(nb)
560 record#1,(nb),1
565 input#1,a$
570 :
575 print b$,a$
580 goto 510
585 :
```

```
590 :
900 rem **--end routine--**
905 dclose#1
910 end
```

The Medical Records System Program

```
100 rem ***--medical records system--***
105 :
110 :
115 rem **--initialization--**
120 :black =  1
125 :white =  2
130 ::::rd =  3
135 :green =  6
140 ::blue =  7
145 yellow = 10
150 purple =  9
155 drkyel = 10
160 ::cyan = 12
165 ::gray = 16
170 ::chtr =  5
175 bckgnd =  0
195 :
200 blank$ = chr$(10):rem blank line
205 print chr$(14):rem upper/lower case mode
210 print chr$(8):rem disable char.set change
215 color bckgnd,black
220 fast
225 :
230 :
300 rem **--variable list--**
305 rem name$ = name associated with information
310 rem  dte$ = date of medical information
315 rem type$ = type of medical information
320 rem misc$ = misc. medical information
325 rem   cv$ = convert to lower case
330 rem srch$ = search for information
335 rem find$ = information in data file
340 rem info$ = associated srch$ information
345 rem  pztn = position within record
350 rem  byte = position within record
355 rem   ptr = pointer for number of records
360 rem   tp$ = full type message
365 rem   col = column number
370 rem   row = row number
375 rem other variables are descriptive
380 :
385 :
500 rem **--medical menu--**
505 col = 24:row = 2
510 scnclr
515 color chtr,white
520 char 0,col,row,"MEDICAL RECORDS MENU":print
525 color chtr,yellow
530 char 0,col,row +  2,"1.  WRITE RECORD"
535 char 0,col,row +  4,"2.  READ RECORD"
540 char 0,col,row +  6,"3.  SEARCH RECORD"
545 char 0,col,row +  8,"4.  SET POINTER"
550 char 0,col,row + 10,"5.  LIST OF FILES"
555 char 0,col,row + 12,"6.  END SESSION"
560 char 0,col,row + 14
565 input "Which Number Please ";number$
570 number = val(number$)
575 :
580 if number = 1 then 1000
585 if number = 2 then 2000
590 if number = 3 then 3000
595 if number = 4 then 4000
```

```
 600 if number = 5 then 5000
 605 if number = 6 then 6000
 620 :
 625 rem *--incorrect choice message--*
 630 gosub 14000:rem incorrect choice
 635 goto 500:rem menu--check again
 640 :
 645 :
1000 rem **--write record routine--**
1005 scnclr
1010 color chtr,white
1015 char 0,25,2,"ENTER INFORMATION"
1020 color chtr,blue
1025 char 0,13,5,"Please do not go beyond the end of the line!"
1030 color chtr,drkyel
1035 :
1040 :
1045 rem *--name information--*
1050 print:print:print
1055 char 0,13,8
1060 print "Type in the individual's name please: ";
1065 sp = 13
1070 gosub 21000:rem underline routine
1075 char 0,51,8
1080 print chr$(2);
1085 gosub 19000:rem getkey subroutine
1090 print chr$(130)
1095 name$ = t$:t$ = ""
1100 if len(name$) > 13 then name$ = left$(name$,13)
1105 :
1110 :
1115 rem *--date information--*
1120 char 0,13,10
1125 print "Type in the date in the form: 2-9-90: ";
1130 sp = 10
1135 gosub 21000:rem underline routine
1140 char 0,51,10
1145 print chr$(2);
1150 gosub 19000:rem getkey subroutine
1155 print chr$(130)
1160 dte$ = t$:t$ = ""
1165 if len(dte$) > 10 then dte$ = left$(dte$,10)
1170 :
1175 :
1180 rem *--type of information--*
1185 gosub 12000:rem display information types
1190 char 0,61,16
1195 sp = 1
1200 gosub 21000:rem underline routine
1205 char 0,61,16
1210 print chr$(2);
1215 gosub 19000:rem getkey subroutine
1220 print chr$(130)
1225 type$ = t$:t$ = ""
1230 cv$ = type$
1235 gosub 17000:rem convert to lowercase
1240 type$ = cv$
1245 if len(type$) > 1 then type$ = left$(type$,1)
1250 :
1255 :
1260 rem *--misc.information--*
1265 scnclr
1270 char 0,1,5,"Type in any miscellaneous information:"
1275 char 0,1,7
1280 sp = 21
1285 gosub 21000:rem underline routine
1290 char 0,1,7
1295 print chr$(2);
1300 gosub 19000:rem getkey subroutine
```

```
1305 print chr$(130)
1310 misc$ = t$:t$ = ""
1315 if len(misc$) > 21 then misc$ = left$(misc$,21)
1320 :
1325 :
1330 rem *--display for correction--*
1335 gosub 16000:rem get full type message
1340 scnclr
1345 col = 25:row = 2
1350 color chtr,white
1355 char 0,col,row,"ENTERED INFORMATION"
1360 color chtr,drkyel
1365 char 0,col,row +  2,"1.  Name: " + name$
1370 char 0,col,row +  4,"2.  Date: " + dte$
1375 char 0,col,row +  6,"3.  Type: " + tp$
1380 char 0,col,row +  8,"4.  Misc: " + misc$
1385 char 0,col,row + 10,"Is this correct? "
1390 gosub 18000:rem y/n input routine
1395 if yes$ = "y" then 1485:rem file output routine
1400 :
1405 char 0,col,row + 12,"Which number is wrong? "
1410 gosub 19000:rem getkey subroutine
1415 nb$ = t$:t$ = ""
1420 nb = val(nb$)
1425 if nb < 1 or nb > 4 then gosub 14000:goto 1405
1430 if nb = 3 then gosub 12000:rem display information types
1435 char 0,col,row + 16,"Type in the correct information: "
1440 gosub 19000:rem getkey subroutine
1445 cinfo$ = t$:t$ = ""
1450 if nb = 1 then name$ = left$(cinfo$,13):goto 1330:rem ask again
1455 if nb = 2 then  dte$ = left$(cinfo$,10):goto 1330:rem ask again
1460 if nb = 3 then type$ = left$(cinfo$,1) :goto 1330:rem ask again
1465 if nb = 4 then misc$ = left$(cinfo$,21):goto 1330:rem ask again
1470 :
1475 :
1480 :
1485 rem **--file output routine--**
1490 :
1495 rem *--get # of records from pointer file--*
1500 dopen#1,"Med.Rec.Ptr"
1505 input#1,ptr
1510 dclose#1
1515 ptr = ptr + 1
1520 :
1525 rem *--data file--*
1530 dopen#2,"Medical Records",l50
1535 :
1540 record#2,(ptr),1
1545 print#2,name$
1550 :
1555 record#2,(ptr),15
1560 print#2,dte$
1565 :
1570 record#2,(ptr),26
1575 print#2,type$
1580 :
1585 record#2,(ptr),28
1590 print#2,misc$
1595 :
1600 dclose#2
1605 :
1610 rem *--update pointer file--*
1615 dopen#1,"@Med.Rec.Ptr",w
1620 print#1,ptr
1625 dclose#1
1630 :
1635 goto 500 rem menu
1640 :
1645 :
```

```
1650 :
2000 rem **--read file routine--**
2005 gosub 10000:rem display routine
2010 if file = 4 then gosub 22000:rem titles routine
2015 :
2020 rem *--pointer file--*
2025 dopen#1,"Med.Rec.Ptr"
2030 input#1,ptr
2035 dclose#1
2040 :
2045 rem *--titles--*
2050 scnclr:print chr$(2);:rem underline
2055 char 0,0,0,"NAME"
2060 char 0,20,0,"DATE"
2065 char 0,35,0,"TYPE"
2070 char 0,55,0,"MISC."
2075 print chr$(130):rem underline off
2080 window 0,2,79,24:rem set window/fix titles
2085 :
2090 rem *--data file--*
2095 dopen#2,"Medical Records"
2100 for rec = 1 to ptr
2105 :
2110 record#2,(rec),1
2115 input#2,name$
2120 :
2125 record#2,(rec),15
2130 input#2,dte$
2135 :
2140 record#2,(rec),26
2145 input#2,type$
2150 :
2155 record#2,(rec),28
2160 input#2,misc$
2165 :
2170 :
2175 rem *--display information--*
2180 gosub 16000:rem get full type message
2185 if file = 4 then gosub 23000:rem printer display
2190 char 0, 0,rec,(name$)
2195 char 0,20,rec,(dte$)
2200 char 0,35,rec,(tp$)
2205 char 0,55,rec,(misc$)
2210 next rec
2215 :
2220 dclose#2
2225 gosub 15000:rem pause routine
2230 window 0,0,79,24
2235 close file
2240 goto 500:rem menu
2245 :
2250 :
2255 :
3000 rem **--search routine--**
3005 scnclr
3010 col = 25:row = 3
3015 color chtr,white
3020 char 0,col,row,"SEARCH FOR...."
3025 color chtr,drkyel
3030 char 0,col,row +  2,"1.  Name"
3035 char 0,col,row +  4,"2.  Date"
3040 char 0,col,row +  6,"3.  Type"
3045 char 0,col,row +  8,"4.  Misc"
3050 char 0,col,row + 10,"5.  End Search"
3055 char 0,col,row + 12,"Which Number Please? "
3060 gosub 19000:rem getkey routine
3065 nb$ = t$:t$ = ""
3070 nb = val(nb$)
3075 :
```

```
3080 if nb = 1 then pztn =  1:b$ = "Name"
3085 if nb = 2 then pztn = 15:b$ = "Date"
3090 if nb = 3 then pztn = 26:b$ = "Type"
3095 if nb = 4 then pztn = 28:b$ = "Misc"
3100 if nb = 5 then 500:rem menu
3105 if nb < 1 or nb > 5 then gosub 14000:goto 3000
3110 :
3115 char 0,col,row + 14,"Which " + b$ + "? "
3120 gosub 19000:rem getkey subroutine
3125 srch$ = t$:t$ = ""
3130 gosub 10000:rem display routine
3135 :
3140 rem *--convert to lowercase--*
3145 cv$ = srch$
3150 gosub 17000:rem convert routine
3155 srch$ = cv$
3160 :
3165 rem *--file input routine--*
3170 :
3175 rem *--pointer file--*
3180 dopen#1,"Med.Rec.Ptr"
3185 input#1,ptr
3190 dclose#1
3195 :
3200 rem *--data file--*
3205 dopen#2,"Medical Records"
3210 ln = len(srch$)
3215 :
3220 for rec = 1 to ptr
3225 record#2,(rec),(pztn)
3230 input#2,find$
3235 :
3240 rem *--convert to lowercase--*
3245 cv$ = left$(find$,ln)
3250 gosub 17000:rem convert routine
3255 if srch$ <> cv$ then 3415:rem next record
3260 :
3265 rem *--matched, now get rest of info.--*
3270 for k = 1 to 4
3275 if k = 1 then byte = 1
3280 if k = 2 then byte = 15
3285 if k = 3 then byte = 26
3290 if k = 4 then byte = 28
3295 record#2,(rec),(byte)
3300 input#2,info$(k)
3305 next k
3310 :
3315 type$ = info$(3)
3320 gosub 16000:rem get full type message
3325 info$(3) = tp$
3330 :
3335 :
3340 rem *--display information--*
3345 scnclr
3350 color chtr,white
3355 char 0,col,row,"MEDICAL INFORMATION"
3360 color chtr,drkyel
3365 print
3370 :
3375 print:print#file,""tab(col) "1.  Name: ";info$(1)
3380 print:print#file,""tab(col) "2.  Date: ";info$(2)
3385 print:print#file,""tab(col) "3.  Type: ";info$(3)
3390 print:print#file,""tab(col) "4.  Misc: ";info$(4)
3395 print:print#file,""tab(col) " "
3400 gosub 15000:rem pause routine
3405 :
3410 :
3415 next rec
```

```
3420 :
3425 :
3430 dclose#2
3435 scnclr
3440 color chtr,white
3445 char 0,col,row,"Search Completed!"
3450 color chtr,drkyel
3455 gosub 15000:rem pause routine
3460 close file
3465 goto 3000:rem search again
3470 :
3475 :
3480 :
4000 rem **--set pointer routine--**
4005 scnclr
4010 char 0,0,5
4015 print "You should only need to set the pointer the first time the program"
4020 print
4025 print "is used. That first time, the value should be set to a '0'. After"
4030 print
4035 print "that, if the pointer file 'Med.Rec.Ptr' is not erased, this routine"
4040 print
4045 print "should not be needed. If the pointer file is erased, use this routine"
4050 print
4055 print "to reset the value of the pointer to the correct number of records."
4060 print:print
4065 print "Do you want to set a value for the pointer? ";
4070 gosub 18000:rem y/n routine
4075 if yes$ = "y" then 4085
4080 goto 500:rem menu
4085 print
4090 print "Type in a value for the pointer: ";
4095 gosub 19000:rem getkey subroutine
4100 ptr$ = t$:t$ = ""
4105 print:print
4110 print "Is this the correct value--pointer = ";ptr$;" ";
4115 gosub 18000:rem y/n routine
4120 if yes$ = "y" then 4135
4125 goto 4000:rem begin again
4130 :
4135 rem *--pointer file--*
4140 scnclr
4145 color chtr,red
4150 char 0,25,7,"ONE MOMENT PLEASE!"
4155 color chtr,drkyel
4160 ptr = val(ptr$)
4165 :
4170 dopen#1,"@Med.Rec.Ptr",w
4175 print#1,ptr
4180 dclose#1
4185 :
4190 scnclr:char 0,15,7
4195 print "The pointer has now been set to: ";ptr
4200 col = 15
4205 gosub 15000:rem pause routine
4210 goto 500:rem menu
4215 :
4220 :
4225 :
5000 rem **--list of files routine--**
5005 scnclr
5010 directory
5015 print:print
5020 print "Are you ready to return to the menu? ";
5025 gosub 18000:rem y/n input routine
5030 if yes$ = "y" then 500:rem menu
5035 goto 5000:rem check again
5040 :
5045 :
```

```
6000 rem **--end routine--**
6005 scnclr
6010 color chtr,grey
6015 char 0,18,9,"That's all for this session! See you next time."
6020 color chtr,drkyel
6025 char 0,0,22
6030 slow
6035 end
6040 :
6045 :
10000 rem **--display subroutine--**
10005 scnclr
10010 char 0,0,3
10015 print "Would you like a paper print out? ";
10020 gosub 18000:rem y/n input subroutine
10025 if yes$ = "y" then 10040:rem printer
10030 if yes$ = "n" then 10115:rem screen
10035 :
10040 rem *--print out is desired--*
10045 scnclr
10050 char 0,0,3
10055 print "Please make sure the printer is on and ready to use."
10060 print
10065 print "Are you ready to begin printing? ";
10070 gosub 18000:rem y/n input subroutine
10075 if yes$ = "n" then 10000:rem ask again
10080 :
10085 rem *--printer display--*
10090 file = 4
10095 dvic = 4
10100 cmnd = 7
10105 goto 10150:rem open instruction
10110 :
10115 rem *--screen display--*
10120 file = 3
10125 dvic = 3
10130 cmnd = 1
10135 goto 10150:rem open instruction
10140 :
10145 :
10150 rem *--open instruction--*
10155 open file,dvic,cmnd
10160 return
10165 :
10170 :
10175 :
12000 rem **--display type of information--**
12005 scnclr
12010 col = 0:row = 2
12015 color chtr,white
12020 char 0,col,row,"Type of Record"
12025 color chtr,drkyel
12030 char 0,col,row +  2,"D--Dr. Visit"
12035 char 0,col,row +  4,"M--Medication"
12040 char 0,col,row +  6,"I--Illness"
12045 char 0,col,row +  8,"A--Accident/Injury"
12050 char 0,col,row + 10,"S--Shot/Immunization"
12055 char 0,col,row + 12,"X--X-Ray"
12060 char 0,col,row + 14
12065 print "Which type of record? (Type a letter: D, M, I, A, S, or X): ";
12070 return
12075 :
12080 :
12085 :
14000 rem *--incorrect choice message--*
14005 print:print
14010 color chtr,white:print tab(col) "Incorrect Choice!"
14015 color chtr,yellow:print:print tab(col) "Press ";
```

```
14020 color chtr,white:print "RETURN";
14025 color chtr,yellow:print " to continue:";
14030 gosub 19000:rem getkey subroutine
14035 return
14040 :
14045 :
14050 :
15000 rem **--pause routine--**
15005 color chtr,purple
15010 char 0,col,22,"Press the "
15015 color chtr,white
15020 print "RETURN";
15025 color chtr,purple
15030 print " key to continue: ";
15035 gosub 19000:rem getkey subroutine
15040 scnclr
15045 return
15050 :
15055 :
15060 :
16000 rem **--full type message--**
16005 if type$ = "d" then tp$ = "Dr. Visit"
16010 if type$ = "m" then tp$ = "Medication"
16015 if type$ = "i" then tp$ = "Illness"
16020 if type$ = "a" then tp$ = "Accident/Injury"
16025 if type$ = "s" then tp$ = "Shot/Immunization"
16030 if type$ = "x" then tp$ = "X-Rays"
16035 return
16040 :
16045 :
16050 :
17000 rem **--convert to lowercase--**
17005 nc$ = ""
17010 for cv = 1 to len(cv$)
17015 x = asc(mid$(cv$,cv,1))
17020 if x > 192 then x = x - 128
17025 nc$ = nc$ + chr$(x)
17030 next cv
17035 :
17040 cv$ = nc$
17045 f = asc(left$(cv$,1))
17050 f = f + 128
17055 nc$ = chr$(f) + right$(cv$,len(cv$)- 1)
17060 return
17065 :
17070 rem na$ = originally typed name
17075 rem cv$ = converted to lowercase
17080 rem nc$ = 1st letter/uppercase
17085 :
17090 :
17095 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 print
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
```

```
18090 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print a$;
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
19070 :
19075 :
21000 rem **--underline routine--**
21005 for i = 1 to sp
21010 print chr$(2);" ";:rem underline on
21015 next i
21020 print chr$(130):rem underline off
21025 return
21030 :
21035 :
21040 :
22000 rem **--printer titles routine--**
22005 print#file,""chr$(12)
22010 print#file,""tab(25) "LIST OF MEDICAL RECORDS"
22015 print#file," "
22020 print#file,""tab(00) "NAME" spc(17) "DATE";
22025 print#file,""spc(12) "TYPE" spc(18) "MISC"
22030 return
22035 :
22040 :
22045 :
23000 rem **--printer display--**
23005 print#file,""tab(00) name$ spc(20 - len(name$)) dte$;
23010 print#file,""spc(14 - len(dte$)) tp$;
23015 print#file,""spc(23 - len(tp$)) misc$
23020 if rec > 50 then gosub 22000
23025 return
```

Chapter 10

Advanced Relative-Access File Manipulation

This chapter provides an examination of a simple, yet fairly complete, relative-access system for home inventory. The file handling portions of the various programs are examined in detail with the expectation of modifying them for use with other applications. The purpose of such modification is to suggest the possibility of the development of a general-purpose database system.

There are six programs in this Home Inventory System: homeinv.menu, homeinv.write, homeinv.read, homeinv.search, homeinv.correct, and homeinv.copy. Each program name attempts to describe the main function of the particular program. Homeinv.menu is the general menu that allows the user to easily switch among the other programs. Homeinv.write is used to create and add to the inventory file. Homeinv.read displays the entire inventory file in the order the information was entered. Homeinv.search is really the heart of the system. This program has a menu of its own with seven options. Six of these options relate to obtaining specific information from the file and displaying it. The next program, homeinv.correct, allows the user to change or delete information in the inventory file. The final program, homeinv.copy, produces a backup copy of the file on any diskette the user chooses. All of the programs are found at the end of this chapter.

CREATING THE HOME INVENTORY FILE

The homeinv.menu, and homeinv.copy programs do not contain any new programming code in relation to files or code not contained in another program, so they will not be discussed. They do, however, contain new instructions that are not file related and might be of interest, so I would encourage you to read through these programs. The homeinv.menu program uses a different approach that will be used from now on. The

first program discussed here, therefore, is the homeinv.write program. You will probably find it helpful to first look over the program (at the end of this chapter) before reading this description.

There is one piece of code that needs explaining. Each of the input sections includes an "sp" value. This value is the number of spaces that the various inputs are allowed in the file. The value checked to see that the user does not exceed the allotted amount. The gosub routine is used to print the varying numbers of underline spaces. This same technique is used in the Medical Records System to limit the number of characters entered by the user to the number allowed in the file.

Lines 1000 to 2000 are the instructions used to check the information and allow the user to change anything before the information is written to the diskette.

Lines 2000 to 3000 are the file-handling lines, and will be discussed in detail.

```
2000 rem **--file output routine--**
2005 :
2010 rem *--get # of records from pointer file--*
2015 dopen#1,"Inv.Rec.Ptr"
2020 input#1,ptr
2025 dclose#1
2030 if status = 16 then 3000
2035 ptr = ptr + 1
2040 :
```

Line 2030 checks whether or not the file has already been created. If the file exists, the status variable will not equal 16 and no error should occur in bringing in the value of the pointer. If, however, this is the first time the program has been used, an error will occur. The error will occur when line 2020 tries to bring in a value for the variable ptr, because no such value has yet been written to the diskette. When the error occurs, the status variable will register a value of 16 (unrecoverable read error). The program is not supposed to halt when this error occurs; rather, the problem should be fixed. Therefore, the routine located between lines 3000 and 4000 is used to write out a value for ptr and then return to the beginning of the file output routine to start the process over. After the use of this error routine, a value will exist on the diskette, and line 2020 can input a value for ptr without an error occurring. After you have a value for the pointer, you add one to that value.

```
2045 rem *--data file--*
2050 dopen#2,"Inventory Record",l100
2055 record#2,(ptr)
2060 :
2065 record#2,(rec),1
2070 print#2,item$
2075 :
2080 record#2,(rec),25
2085 print#2,serl$
2090 :
2095 record#2,(rec),40
2100 print#2,cst$
2105 :
```

```
2110 record#2,(rec),50
2115 print#2,room$
2120 :
2125 record#2,(rec),70
2130 print#2,desc$
2135 :
2140 record#2,(ptr)
2145 dclose#2
2150 :
2155 rem *--update pointer file--*
2160 dopen#1,"@Inv.Rec.Ptr",w
2165 print#1,ptr
2170 dclose#1
2175 :
2180 :
2185 goto 4000: rem repeat routine
2190 :
2195 :
```

Lines 2065 and 2130 instruct the computer to write out the information collected from the user to the inventory file. Each piece of information is given a certain maximum number of spaces. Most information will not take up the maximum, so some space in each field will be left blank. Item$ information can contain up to 24 characters or bytes of information (23 characters of information and 1 byte for the delimiter). Serial$ information can have up to 15 bytes of information. Cst$ information has a maximum of 10 bytes, room$ information 20 bytes, and Desc$ information 30 bytes.

When all the information has been transferred to the diskette, the pointer value is written out to its sequential file. The user is queried about adding more information to the file, and the appropriate action taken upon obtaining a response.

DISPLAYING THE HOME INVENTORY INFORMATION

The homeinv.read program is really the reverse of the routine just covered. INPUT# is substituted for PRINT#, and the values of the variables are formatted for display on the screen rather than being written to the diskette. Otherwise, the routines are very similar. Each field of each record is read into the computer from the diskette and displayed. When all records have been read in and displayed, the user is transferred to the homeinv.menu program.

```
2000 rem **--read file routine--**
2002 color chtr,yellow
2005 gosub 10000:rem display routine
2010 if file = 4 then gosub 22000:rem titles routine
2015 :
2020 rem *--pointer file--*
2025 dopen#1,"Inv.Rec.Ptr"
2030 input#1,ptr
2035 dclose#1
2040 :
```

```
2045 rem *--titles--*
2050 scnclr:print chr$(2);:rem underline on
2055 char 0,0,0,"ITEM"
2060 char 0,20,0,"SER#"
2065 char 0,35,0,"COST"
2070 char 0,45,0,"ROOM"
2072 char 0,60,0,"DESC"
2075 print chr$(130):rem underline off
2080 :
2085 rem *--data file--*
2090 window 0,2,79,24:rem fix window
2095 dopen#2,"Inventory Record"
2100 for rec = 1 to ptr
2105 :
2110 record#2,(rec),1
2115 input#2,item$
2120 :
2125 record#2,(rec),25
2130 input#2,serl$
2135 :
2140 record#2,(rec),40
2145 input#2,cst$
2150 :
2155 record#2,(rec),50
2160 input#2,room$
2165 :
2166 record#2,(rec),70
2167 input#2,desc$
2168 :
2170 :
2175 rem *--display information--*
2185 if file = 4 then gosub 23000:rem printer display
2190 char 0, 0,rec,(item$)
2195 char 0,20,rec,(serl$)
2200 char 0,35,rec,(cst$ )
2205 char 0,60,rec,(desc$)
2210 next rec
2215 :
2220 dclose#2
2225 gosub 15000:rem pause routine
2230 window 0,0,79,24,1:rem reset window and clear
2235 close file
2240 :
2245 :
2250 :
```

SEARCHING AND SORTING THE FILE

The main program of the Home Inventory System is the homeinv.search program, consisting of six sort or search routines and an option to return to the main home inventory menu:

1. Search For Item
2. Search For Serial #
3. Search For Cost
4. Search For Room Items
5. Sort Items—Alpha. Order
6. Sort Items—Serial #
7. Return to Home Inv.Menu

Numbers 1, 2, and 4 use a common search subroutine. The two sort options (numbers 5 and 6) use a common sort subroutine, the Shell-Metzner sort. Option number 3 uses its own search routines for both parts of this selection. The common search subroutine will be discussed first.

The Common Search Routine

```
13000 rem **--common search routine--**
13005 color chtr,cyan
13010 char 0,4,row + 17
13015 print chr$(27) + chr$(81);:rem clear line
13020 print "Search for which ";b$;"? ";
13025 gosub 19000:rem getkey subroutine
13030 srch$ = t$:t$ = ""
13035 gosub 10000:rem display routine
13040 color chtr,rd
13045 char 0,4,row + 17
13050 print chr$(27) + chr$(81);:rem clear line
13055 print chr$(15);:print "Searching.....Please Wait!"
13060 print chr$(143):rem turn flash off
13065 :
13070 rem *--convert to lowercase--*
13075 cv$ = srch$
13080 gosub 17000:rem convert routine
13085 srch$ = cv$
13090 :
13095 rem *--file input routine--*
13100 :
13105 rem *--data file--*
13110 dopen#2,"Inventory Record"
13115 ln = len(srch$)
13120 :
13125 for rec = 1 to ptr
13130 record#2,(rec),(pztn)
13135 input#2,find$
13140 :
13145 rem *--convert to lowercase--*
13150 cv$ = left$(find$,ln)
13155 gosub 17000:rem convert routine
13160 if srch$ <> cv$ then 13315:rem next record
13165 :
13170 rem *--matched, now get rest of info.--*
```

```
13175 for k = 1 to 5
13180 if k = 1 then byte =  1
13185 if k = 2 then byte = 25
13190 if k = 3 then byte = 40
13195 if k = 4 then byte = 50
13200 if k = 5 then byte = 70
13205 record#2,(rec),(byte)
13210 input#2,info$(k)
13215 next k
13220 :
13225 :
13230 :
13235 rem *--display information--*
13240 scnclr
13245 color chtr,white
13250 char 0,col,row,"HOME INVENTORY INFORMATION"
13255 color chtr,drkyel
13260 print
13265 :
13270 print:print#file,""tab(col) "1.  Item: ";info$(1)
13275 print:print#file,""tab(col) "2.  Ser#: ";info$(2)
13280 print:print#file,""tab(col) "3.  Cost: ";info$(3)
13285 print:print#file,""tab(col) "4.  Room: ";info$(4)
13290 print:print#file,""tab(col) "5.  Desc: ";info$(5)
13295 print:print#file,""tab(col) " "
13300 gosub 15000:rem pause routine
13305 :
13310 :
13315 next rec
13320 :
13325 :
13330 dclose#2
13335 scnclr
13340 color chtr,white
13345 char 0,col,row,"Search Completed!"
13350 color chtr,drkyel
13355 gosub 15000:rem pause routine
13360 close file
13365 return
13370 :
13375 :
13380 :
```

This subroutine is common to the first two options and to the Search For Items option. Each option routine that uses this subroutine establishes the necessary conditions prior to entering the subroutine. The values of srch$ and pztn are determined prior to the GOSUB statement in each of the option routines. Once these values are known, the specified part of the file can be searched for any match (line 13160). If a match occurs, control passes to the instructions at lines 13170 to 13300. These instructions read in from the diskette the information associated with the item searched for and then display the information.

The RETURN statement in line 13365 returns control to the instruction following the GOSUB statement in the original option routine (option 1, 2, or 4). When a match does not occur, the record counter (rec) is incremented and the progress is repeated.

The Search for Cost Routine

The next section of code discussed is part one of the Search-For-Cost option. In lines 3000 to 3110, a decision is made by the user: whether to search for items above a certain cost or items below a certain cost. The appropriate part of this option routine is then given control. The following code is for items above a specific value.

```
3200 rem *--items above $ amount--*
3205 char 0,col,row + 10,"Above which amount? "
3210 gosub 19000:rem getkey subroutine
3215 amt$ = t$:t$ = ""
3220 amt = val(amt$)
3225 gosub 10000:rem display subroutine
3230 scnclr
3235 col = 25:row = 2
3240 print#file,"" tab(25) "Items Above ";
3245 print#file,using "#$####.##";amt
3250 print#file,blank$:print#file,blank$
3255 dopen#2,"Inventory Record",l100
3260 for w = 1 to ptr
3265 :
3270 record#2,w,1
3275 input#2,item$
3280 :
3285 record#2,w,40
3290 input#2,cst$
3295 :
3300 c$ = cst$:cst = val(cst$)
3305 if left$(c$,7) = "DELETED" then 3340:rem next w
3310 if left$(c$,1) = "$" then gosub 12000:rem strip $
3315 if val(c$) > amt then 3325
3320 goto 3340:rem next w
3325 ttlamt = ttlamt + val(c$)
3330 print#file,item$;
3335 print#file,"" tab(30):print#file,using "#$####.##";cst
3340 next w
3345 :
3350 print
3355 print#file,"Total Value = ";
3360 print#file,"" tab(30)
3365 print#file,using "#$####.##";ttlamt
3370 dclose#2:print#file:close file
3375 gosub 24000:rem clear variables
3380 gosub 15000:rem pause routine
3385 goto 500:rem menu
3390 :
3395 :
```

The items that are valued above a certain amount are searched for in line 3315. The amount is previously determined in line 3205 and displayed in line 3240. Line 3260 begins a loop that extends through line 3340. Each record is searched for costs that exceed the specified amount. Line 3315 says that, if the cost of the record being examined exceeds the amount specified, control is passed to line 3325. When such an item has been found, a running total is kept of the cumulative value of these items (line 3325), and the item and its value are displayed (lines 3330 and 3335). After all the records have been examined, the total value of all items above the specific amount is given (lines 3355 to 3365), and control is transferred to the clear variables (line 3375) and pause subroutines (line 3380). Finally, control is shifted back to the menu for further instructions (line 3385).

The routine to find items below a certain value is virtually the same as that just given. The only significant difference occurs in line 3615 where the sign is reversed. We are looking for items whose value is less than the specified amount. Those items whose value is greater than the specified amount are passed over.

You have looked briefly at the first four options, the search options. The next two options are sort options and use a common sort subroutine, the Shell-Metzner sort. I will explain only the procedures involved in setting up and using a sort subroutine with Commodore diskette files. The alphabetizing routine will be covered first.

The Alphabetical Order of Items

```
5000 rem **--sort items alpha.order--**
5005 scnclr
5010 color chtr,rd
5015 col = 25:row = 5
5020 print chr$(15);:rem flash
5025 char 0,col,row,"WORKING--PLEASE DON'T TOUCH!"
5030 print chr$(143):rem flash off
5035 color chtr,yellow
5040 :
5045 rem **--get information from file--**
5050 q = 1
5055 dopen#2,"Inventory Record",l100
5060 for w = 1 to ptr
5065 :
5070 record#2,w,1
5075 input#2,item$
5080 :
5085 if item$ = "DELETED" then 5130:rem next w
5090 :
5095 rem *--convert to upper case--*
5100 cv$ = item$
5105 gosub 17500:rem upper case
5110 item$ = cv$
5115 :
5125 a$(q) = item$:rem store in array for internal sort
5130 next w
5135 :
5140 :
```

```
5145 dclose#2
5150 n = q - 1
5155 color chtr,rd
5160 print chr$(15);:rem flash
5165 char 0,col,row,"STILL WORKING--PLEASE WAIT!"
5170 print chr$(143):rem flash off
5175 color chtr,yellow
5180 gosub 25000:rem sort routine
5185 t$ = ""
5190 :
5195 :
5200 rem *--display results--*
5205 scnclr
5210 gosub 10000:rem display routine
5215 scnclr
5220 color chtr,white
5225 print#file,"" tab(col) "Items Sorted By Alphabetical Order"
5230 color chtr,yellow
5235 print#file," ":print#file," "
5240 for i = 1 to q - 1
5245 print#file,using "####";i;
5250 print#file,". ";a$(i)
5255 next i
5260 :
5265 gosub 24000:rem clear variables
5270 print#file,"":close file
5275 gosub 15000:rem pause routine
5280 goto 500:rem return to menu
5285 :
5290 :
5295 :
```

The keys to this routine are reading in only the item names, storing them in a string array, sorting them with the sort subroutine located between lines 25000 and 25075, and displaying them in their alphabetized order.

A separate record counter is used (line 5050) to keep track of the valid records, because there might be some records that have been deleted and now contain the value "DELETED." If there are such records, they are skipped and the loop (w) is increased by one, but the valid record counter (q) is not increased. If the record is not valid (it contains "DELETED"), it is also not included in the string array of valid records to be sorted. Once the loop is completed, the string array a$( ) should contain all the valid item names. A new warning message is displayed (line 5165), and control is transferred to the sort subroutine. When the sorting has been completed, the results are displayed through another loop (lines 5240 to 5255).

Two lines in this routine (line 5265 and line 5275) are common to all the routines. They simply "clean up" various conditions that might have been set during execution of the routine, and allow the user to pause before going on to the next step in any routine.

Sorting By Serial Number

The last of the options, sort by serial number, again makes use of the LEFT$ and MID$

string array commands. It is also the longest of the routines. This routine sorts by serial number and then displays the resulting list in serial number order, along with the associated item name. It is conceivable that an individual or insurance company would need all the associated information instead of just the item name. Therefore, if you are interested in developing a completely useful Home Inventory System, you might wish to add the code necessary to display all related information in both serial number order and alphabetical order.

```
6000 rem **--sort items by serial number--**
6005 scnclr
6010 color chtr,rd
6015 col = 25:row = 6
6020 print chr$(15);:rem flash
6025 char 0,col,row,"WORKING--PLEASE DON'T TOUCH!"
6030 print chr$(143):rem flash off
6035 color chtr,yellow
6040 :
6045 rem **--get information from file--**
6050 q = 1
6055 dopen#2,"Inventory Record",l100
6060 for w = 1 to ptr
6065 :
6070 record#2,w,1
6075 input#2,item$
6077 :
6078 record#2,w,25
6079 input#2,serl$
6080 :
6085 if item$ = "DELETED" then 6130:rem next w
6090 :
6095 rem *--convert to upper case--*
6100 cv$ = serl$
6105 gosub 17500:rem upper case
6110 serl$ = cv$
6115 :
6120 a$(q) = serl$ + "*" item$:rem store in array for internal sort
6125 q = q + 1
6130 next w
6135 :
6140 :
6145 dclose#2
6150 n = q - 1
6155 color chtr,rd
6160 print chr$(15);:rem flash
6165 char 0,col,row,"STILL WORKING--PLEASE WAIT!"
6170 print chr$(143):rem flash off
6175 color chtr,yellow
6180 gosub 25000:rem sort routine
6185 t$ = ""
6190 :
6195 :
```

```
6200 rem *--display results--*
6205 scnclr
6210 gosub 10000:rem display routine
6215 scnclr
6220 color chtr,white
6225 print#file,"" tab(col) "Items Sorted By Serial Number"
6230 color chtr,yellow
6235 print#file," ":print#file," "
6240 :
6245 for i = 1 to q - 1
6250 la = len(a$(i))
6255 j = instr(a$(i),"*")
6260 print#file,using "####";i;
6265 print#file,". ";left$(a$(i),j - 1);
6270 if file = 4 then 23000
6270 print#file,"" tab(25) mid$(a$(i),j + 1,la)
6275 next i
6280 :
6285 gosub 24000:rem clear variables
6290 print#file,"":close file
6295 gosub 15000:rem pause routine
6300 goto 500:rem return to menu
6305 :
6310 :
6315 :
```

Lines 6078 and 6079 bring in the serial number of each item. If the serial number has been deleted (contains the word "DELETED"), the record is skipped, as in the previous routine. In fact, the two sort routines have nearly identical beginnings. The main difference occurs in lines 6095 to 6130, when the serial number, instead of the item name, goes through the conversion to uppercase. The only other major difference occurs when the item name is concatenated (joined) to the serial number. Line 6120 combines the existing value of the serial number (serl$), the current value of the item name (item$), and a separator (the asterisk) into one new string array value, a$(q).

After the entire file is read and the correct number of valid records determined, control is passed to the sort subroutine (line 6180). Lines 6200 to 6275 are used to display the result of the sort. Here again, you need to make use of the power of the LEFT$, MID$, and LEN functions. The numeric variable la is set to equal the length of each of the string arrays (line 6250). The INSTR function is used to determine where in the string the asterisk is located (6255). The LEFT$ and MID$ functions are used to print out the desired parts of the string in an acceptable format (6260 to 6270). This sequence is repeated until all valid records have been displayed in serial number order. The end of this routine is the same as the end of the other five routines.

This concludes the discussion of the Search/Sort Home Inventory (homeinv.search) program. There are a number of other points that could be discussed, but those points relate mainly to different techniques of programming in BASIC rather than techniques for working with Commodore BASIC files. By now, if you have worked through all the programs, you should be able to read a program yourself to recognize some of the different techniques used.

CORRECTING THE HOME INVENTORY INFORMATION

The next program in this Home Inventory system provides the ability to change or delete information in the inventory record file. Both parts of this program make use of two routines: a file-output routine and a file-input routine. These two routines have been used in our other programs in this system. The correct-record routine (lines 1000 to 2000) is essentially the same as the correction routine in the homeinv.write program (lines 1000 to 2000). The difference is that in the homeinv.write program, the information being checked for accuracy comes from the keyboard. In the homeinv.correct program, the information comes from the diskette. That is the reason for line 1085. This line transfers control to the file input routine, which inputs from the specified record on the diskette the values for item$, serl$, cst$, room$, and desc$. These values are then displayed, and a check is made to see if they are correct.

At this point, one other new line of code is encountered (line 1002). Lines 1002, 1585, 1587, and 1637 are all related. All deal with a string variable called flag$. Line 1002 sets the original value of flag$ equal to the word "no." This indicates that no information has yet been changed. Lines 1585 and 1587 check the value of flag$ and direct the computer accordingly. If the information has been changed, the value of flag$ will have been changed by line 1637 to "yes," indicating altered information. If the information is correct and has been changed, you are now ready to write that information back out to the file on the diskette (the file output routine). This technique allows the user to scan through the records if he or she is not sure of the record number of the incorrect information.

The deletion routine is relatively uncomplicated. The suspected record is brought in from diskette (line 2085) and displayed (lines 2500 to 2555). A request is made of the user to see if this is the information to be deleted. If it is not, the deletion routine starts again. If the information is to be deleted, the user is required to type the word "YES" rather than just the "Y." If "YES" is typed, all string variables are given the value "DELETED," and control is passed to the file output routine, where "DELETED" replaces the information. Notice that the entire file does not need to be resequenced and rewritten to the diskette. Instead, only the information requiring change is affected.

The change and delete routines for relative-access files are considerably easier than similar routines for sequential-access files. This ease is one of the major strengths of relative files. Access is direct to any part of the file desired. In fact, in a very large inventory system, it is possible to read from diskette and check only the desired part of the record, rather than the entire record. Programming can often be simpler and easier to read. There is less need for string arrays and, therefore, less need for large amounts of internal computer memory. The diskette can be used as an extension of the internal memory with relative files, because the same principles are involved. The major difference is in the time involved; diskette access is much slower than internal memory access.

At the end of the chapter, I have included the first two programs of another system, A Magazine Article System, which has been created by modifying this current Home Inventory System. The modification is not extensive. The main reason for including the first portion of the Magazine Article System is to suggest the possibility of a general purpose database program. All systems discussed here have included some method for (1) creating and adding to a file, (2) displaying information from that file in various ways, and (3) editing the file. These are the essential characteristics in any database system. It should be possible to create a general purpose database system that would request certain necessary information from the user. Based on the supplied information, this general database system would create a file and set up the procedures to display and edit information in that file.

The better commercial database programs have expanded on these essential characteristics. They have added features that some users might need but others will never use. In the Introduction, I said that "reading this book will not make you capable of creating complete database programs . . .," but at this point, you should have an appreciation of the effort that goes into creating a good general-purpose database system. For your individual use, you might find that you can create a semi-general purpose database system, a system that can serve your needs but would not be universal in meeting the needs of everyone. This is the reason for including the beginning of the Magazine Article System as a modification of the Home Inventory System. Structured carefully, with enough user-supplied variables, this series of programs can form the basis for such a personal database system.

The next chapter will deal with the planning necessary to create the programs for any file system. The example will be a Stock Market System for keeping track of the price and volume changes of certain issues.

Figure 10-1 shows the programs discussed in this chapter.

CHAPTER 10 QUESTIONS

1. What BASIC reserved phrase is only used the first time the homeinv.write program is run?
2. Give the name of the string variable that allows you to use one search routine for three different program modules.
3. What sort routine is used in both systems in this chapter?
4. What word means "joining string variables together"?
5. True or False: It is more difficult to change information in a relative-access file than in a sequential file.

Fig. 10-1. The Home Inventory and Magazine Article Systems programs.

The homeinv.menu Program

```
100 rem ***--home inventory system menu--***
105 :
110 :
115 rem **--initialization--**
120 :black =  1
125 :white = 16
130 ::::rd =  3
135 :green =  6
140 ::blue =  7
145 yellow = 10
150 purple =  9
155 drkyel = 10
160 ::cyan = 12
165 ::gray = 16
170 ::chtr =  5
175 bckgnd =  0
195 :
200 blank$ = chr$(10):rem blank line
205 print chr$(14):rem upper/lower case mode
210 print chr$(8):rem disable char.set change
215 color bckgnd,black
220 fast
```

```
225 ttle$ = "HOME INVENTORY SYSTEM"
230 ln = len(ttle$)
235 center = (80 - ln) / 2:rem center title
240 key 1,"goto 9000" + chr$(13)
245 :
250 :
500 rem **--home inventory menu--**
505 scnclr
510 color chtr,purple
515 :
520 rem *--draw top line--*
525 for i = 2 to 79
530 print chr$(2);" ";
535 next i
540 print chr$(130)
545 :
550 rem *--display choices--*
555 col = center:row = 2
560 color chtr,white
565 char 0,4,row,"Inventory"
570 char 0,col,row,"HOME INVENTORY MENU            Select Action"
575 color chtr,yellow
580 col =  8:row = 4
585 char 0,col,row +  2,"1.  WRITE RECORD"
590 char 0,col,row +  4,"2.  READ RECORD"
595 char 0,col,row +  6,"3.  SEARCH RECORD"
600 char 0,col,row +  8,"4.  CORRECT RECORD"
605 char 0,col,row + 10,"5.  COPY FILE"
610 col = 50:row = 4
615 char 0,col,row +  2,"6.  LIST OF FILES"
620 char 0,col,row +  4,"7.  END SESSION"
625 col = 4:row = 4
630 char 0,col,row + 14
635 print "Type your Selection ";
640 color chtr,white:print "(1-7)";
645 color chtr,yellow:print " and press ";
650 color chtr,white:print "RETURN";
655 color chtr,yellow:print ": ---> ";
660 x = pos(0)
665 color chtr,purple:char 0,0,row + 15
670 :
675 rem *--draw bottom line--*
680 for i = 2 to 79
685 print chr$(2);" ";
690 next i
695 print chr$(130)
700 :
705 rem *--options--*
710 color chtr,yellow
715 char 0, 4,row + 17,"Options: "
720 color chtr,white:print "ESC";:
725 color chtr,yellow:print " key to leave.  ";
730 color chtr,white:print "HELP";
735 color chtr,yellow:print " key for information."
740 :
745 rem *--wait for keypress--*
750 char 0,x,row + 14
755 gosub 19000:rem getkey routine
760 number$ = t$:t$ = ""
765 if number$ = chr$(13) then 750
770 number = val(number$)
775 :
780 if number = 1 then 1000
785 if number = 2 then 2000
790 if number = 3 then 3000
795 if number = 4 then 4000
800 if number = 5 then 5000
805 if number = 6 then 6000
```

```
 810 if number = 7 then 7000
 820 :
 825 rem *--incorrect choice message--*
 830 gosub 14000:rem incorrect choice
 835 goto 500:rem menu--check again
 840 :
 845 :
1000 rem **--write record program--**
1005 file$ = "HOMEINV.WRITE"
1010 gosub 17000:rem new program routine
1015 run "homeinv.write"
1020 :
1025 :
2000 rem **--read record program--**
2005 file$ = "HOMEINV.READ"
2010 gosub 17000:rem new program routine
2015 run "homeinv.read"
2020 :
2025 :
3000 rem **--search record program--**
3005 file$ = "HOMEINV.SEARCH"
3010 gosub 17000:rem new program routine
3015 run "homeinv.search"
3020 :
3025 :
4000 rem **--correct record program--**
4005 file$ = "HOMEINV.CORRECT"
4010 gosub 17000:rem new program routine
4015 run "homeinv.correct"
4020 :
4025 :
5000 rem **--copy file program--**
5005 file$ = "HOMEINV.COPY"
5010 gosub 17000:rem new program routine
5015 run "homeinv.copy"
5020 :
5025 :
6000 rem **--list of files routine--**
6005 scnclr
6010 directory
6015 print:print
6020 print "Are you ready to return to the menu? ";
6025 gosub 18000:rem y/n input routine
6030 if yes$ = "y" then 500:rem menu
6035 goto 6000:rem check again
6040 :
6045 :
7000 rem **--end routine--**
7005 scnclr
7010 color chtr,grey
7015 char 0,18,9,"That's all for this session! See you next time."
7020 color chtr,drkyel
7025 char 0,0,22
7030 slow
7035 end
7040 :
7045 :
9000 rem **--information section--**
9005 color chtr,cyan
9010 getkey b$,c$,d$
9012 char 0,4,row + 17
9013 print chr$(27) + chr$(81);:rem erase line
9015 input "Which Selection do you need information about";e$
9017 getkey e$
9018 e = val(e$)
9020 color chtr,yellow
9021 if e = 1 then 9100
9022 if e = 2 then 9200
```

```
9023 if e = 3 then 9300
9024 if e = 4 then 9400
9025 if e = 5 then 9500
9026 if e = 6 then 9600
9027 if e = 7 then 9700
9028 gosub 14000:rem incorrect choice
9098 :
9099 :
9100 rem **--information about option #1--**
9105 scnclr
9110 char 0,4,7,"This option allows you to enter information about the contents"
9115 print:print
9120 print "    of your home. You will be asked to provide the item name, serial"
9125 print
9130 print "    number, cost, and room location for each item you wish to inventory."
9135 print
9140 print "    Each entry allows only a certain amount of space. Please do not"
9145 print
9150 print "    exceed the alloted number of characters."
9155 gosub 15000:rem pause routine
9195 goto 500:rem return to menu
9198 :
9199 :
9200 rem **--information about option #2--**
9205 scnclr
9210 char 0,4,7,"This option displays the contents of the Home Inventory file"
9215 print:print
9220 print"    in a very elementry form. The entire file is read and displayed."
9225 print
9230 print "    You are given the choice of displaying the information either on"
9235 print
9240 print "    the screen or on paper with a printer."
9245 gosub 15000:rem pause routine
9295 goto 500:rem return to menu
9298 :
9299 :
9300 rem **--information about option #3--**
9305 scnclr
9310 char 0,4,7,"This option is the heart of the Home Inventory System. You are "
9315 print:print
9320 print "    presented with another menu of seven choices allowing you to"
9325 print
9330 print "    search and/or sort the file information in a number of different"
9335 print
9340 print "    formats. There are four search routines, two sort routines and an"
9345 print
9350 print "    option allowing you to return to this main program menu."
9355 gosub 15000:rem pause routine
9395 goto 500:rem return to menu
9398 :
9399 :
9400 rem **--information about option #4--**
9405 scnclr
9410 char 0,4,7,"This option allows you to correct or delete information in"
9415 print:print
9420 print "    the Home Inventory file. You may change information as often"
9425 print
9430 print "    as you want. Deleted information is actually removed, not just"
9435 print
9440 print "    flagged. Therefore, be certain that you do not need the"
9445 print
9450 print "    information before deleting it."
9455 gosub 15000:rem pause routine
9495 goto 500:rem return to menu
9498 :
9499 :
9500 rem **--information about option #5--**
9505 scnclr
```

```
 9510 char 0,4,7,"This option allows you to make a copy of the Home Inventory"
 9515 print:print
 9520 print "    file. You may NOT make the copy on the same diskette but must"
 9525 print
 9530 print "    use a different diskette and the new diskette must have been"
 9535 print
 9540 print "    formatted previously!"
 9545 gosub 15000:rem pause routine
 9595 goto 500:rem return to menu
 9598 :
 9599 :
 9600 rem **--information about option #6--**
 9605 scnclr
 9610 char 0,4,7,"This option simply displays a directory of the current diskette."
 9615 gosub 15000:rem pause routine
 9695 goto 500:rem return to menu
 9698 :
 9699 :
 9700 rem **--information about option #7--**
 9705 scnclr
 9710 char 0,4,7,"This option allows you to stop using the Home Inventory System."
 9715 print:print:
 9720 print "    You are returned to BASIC and can then safely remove the diskette"
 9725 print
 9730 print "    from the disk drive."
 9735 gosub 15000:rem pause routine
 9795 goto 500:rem return to menu
 9798 :
 9799 :
14000 rem *--incorrect choice message--*
14005 char 0,4,23
14010 color chtr,white:print tab(col) "Incorrect Choice! ";
14015 color chtr,yellow:print "Press the ";
14020 color chtr,white:print "RETURN";
14025 color chtr,yellow:print " key to continue:";
14030 gosub 19000:rem getkey subroutine
14035 return
14040 :
14045 :
14050 :
15000 rem **--pause routine--**
15005 color chtr,purple
15010 char 0,col,22,"Press the "
15015 color chtr,white
15020 print "RETURN";
15025 color chtr,purple
15030 print " key to continue: ";
15035 gosub 19000:rem getkey subroutine
15040 scnclr
15045 return
15050 :
15055 :
15060 :
17000 rem **--new program routine--**
17005 scnclr:char 0,15,5
17010 print "You have selected the ";
17015 color chtr,red
17020 print file$;
17025 color chtr,yellow
17030 print " program."
17035 char 0,15,9
17040 print "Is this the program you want? ";
17045 gosub 18000:rem y/n input routine
17050 if yes$ = "n" then 500:rem menu
17055 :
17060 rem *--loading message--*
17065 scnclr
17070 color chtr,purple:char 0,15,5
```

```
17075 print "Please wait! I'm loading....";
17080 color chtr,red
17085 print file$
17090 color chtr,purple
17095 return
17100 :
17105 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 print
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
18090 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19007 if a$ = chr$(27) then 7000
19008 if a$ = chr$(72) then 9000:rem help key
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print a$;
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
```

The homeInv.write Program

```
  100 rem ***--write inventory record--***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white = 16
  130 ::::rd =  3
  135 :green =  6
  140 ::blue =  7
  145 yellow = 10
  150 purple =  9
  155 drkyel = 10
  160 ::cyan = 12
  165 ::gray = 16
  170 ::chtr =  5
  175 bckgnd =  0
  195 :
  200 blank$ = chr$(10):rem blank line
  205 print chr$(14):rem upper/lower case mode
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 fast
  225 :
  230 :
  300 rem **--variable list--**
```

```
  305 rem item$ = name of item inventoried
  310 rem serl$ = serial # of item
  315 rem  cst$ = cost of item
  320 rem room$ = location of item
  322 rem desc$ = description of item
  325 rem   cv$ = convert to lower case
  330 rem srch$ = search for information
  335 rem find$ = information in data file
  340 rem info$ = associated srch$ information
  345 rem  pztn = position within record
  350 rem  byte = position within record
  355 rem   ptr = pointer for number of records
  365 rem   col = column number
  370 rem   row = row number
  375 rem other variables are descriptive
  380 :
  385 :
 1000 rem **--write record routine--**
 1005 scnclr:col = 10
 1010 color chtr,white
 1015 char 0,25,2,"ENTER INFORMATION"
 1020 color chtr,blue
 1025 char 0,col,5,"Please do not go beyond the end of the line!"
 1030 color chtr,drkyel
 1035 :
 1040 :
 1100 rem *--name of item--*
 1105 print:print:print
 1110 char 0,col,8
 1115 msg$ = "Type in the item's name please: "
 1117 lm = len(msg$)
 1118 print msg$;
 1120 sp = 23
 1125 gosub 21000:rem underline routine
 1130 char 0,lm + col,8
 1135 print chr$(2);
 1140 gosub 19000:rem getkey subroutine
 1145 print chr$(130)
 1150 item$ = t$:t$ = ""
 1155 if len(item$) > 23 then item$ = left$(item$,23)
 1160 :
 1165 :
 1200 rem *--serial number of item--*
 1205 char 0,col,10
 1210 msg$ = "Type in the item's serial # please: "
 1215 lm = len(msg$)
 1220 print msg$;
 1225 sp = 14
 1230 gosub 21000:rem underline routine
 1235 char 0,lm + col,10
 1240 print chr$(2);
 1245 gosub 19000:rem getkey subroutine
 1250 print chr$(130)
 1255 serl$ = t$:t$ = ""
 1260 if len(serl$) > 14 then serl$ = left$(serl$,14)
 1265 :
 1270 :
 1300 rem *--cost of item--*
 1305 char 0,col,12
 1310 msg$ = "Type in the item's cost please: "
 1315 lm = len(msg$)
 1320 print msg$;
 1325 sp = 9
 1330 gosub 21000:rem underline routine
 1335 char 0,lm + col,12
 1340 print chr$(2);
 1345 gosub 19000:rem getkey subroutine
 1350 print chr$(130)
 1355 cst$ = t$:t$ = ""
```

```
1360 if len( cst$) >  9 then  cst$ = left$( cst$, 9)
1365 :
1370 :
1400 rem *--room of item--*
1405 char 0,col,14
1410 msg$ = "Type in the item's room please: "
1415 lm = len(msg$)
1420 print msg$;
1425 sp = 19
1430 gosub 21000:rem underline routine
1435 char 0,lm + col,14
1440 print chr$(2);
1445 gosub 19000:rem getkey subroutine
1450 print chr$(130)
1455 room$ = t$:t$ = ""
1460 if len(room$) > 19 then room$ = left$(room$,19)
1465 :
1470 :
1500 rem *--description of item--*
1505 char 0,col,16
1510 msg$ = "Type in the item's description please: "
1515 lm = len(msg$)
1520 print msg$;
1525 sp = 29
1530 gosub 21000:rem underline routine
1535 char 0,lm + col,16
1540 print chr$(2);
1545 gosub 19000:rem getkey subroutine
1550 print chr$(130)
1555 desc$ = t$:t$ = ""
1560 if len(desc$) > 29 then desc$ = left$(desc$,29)
1565 :
1570 :
1800 rem *--display for correction--*
1810 scnclr
1815 col = 17:row = 2
1820 color chtr,white
1825 char 0,col,row,"ENTERED INFORMATION"
1830 color chtr,drkyel
1835 char 0,col,row +  2,"1.  Item: " + item$
1840 char 0,col,row +  4,"2.  Ser#: " + serl$
1845 char 0,col,row +  6,"3.  Cost: " + cst$
1850 char 0,col,row +  8,"4.  Room: " + room$
1855 char 0,col,row + 10,"5.  Desc: " + desc$
1875 char 0,col,row + 12,"Is this correct? "
1880 gosub 18000:rem y/n input routine
1885 if yes$ = "y" then 2000:rem file output routine
1890 :
1895 char 0,col,row + 14,"Which number is wrong? "
1900 gosub 19000:rem getkey subroutine
1905 nb$ = t$:t$ = ""
1910 nb = val(nb$)
1915 if nb < 1 or nb > 5 then gosub 14000:goto 1895
1925 char 0,col,row + 16,"Type in the correct information: "
1930 gosub 19000:rem getkey subroutine
1935 cinfo$ = t$:t$ = ""
1940 if nb = 1 then item$ = left$(cinfo$,23):goto 1800:rem ask again
1945 if nb = 2 then serl$ = left$(cinfo$,14):goto 1800:rem ask again
1950 if nb = 3 then  cst$ = left$(cinfo$,9) :goto 1800:rem ask again
1955 if nb = 4 then room$ = left$(cinfo$,19):goto 1800:rem ask again
1960 if nb = 5 then desc$ = left$(cinfo$,29):goto 1800:rem ask again
1990 :
1995 :
2000 rem **--file output routine--**
2005 :
2010 rem *--get # of records from pointer file--*
2015 dopen#1,"Inv.Rec.Ptr"
2020 input#1,ptr
2025 dclose#1
```

```
 2030 if status = 16 then 3000
 2035 ptr = ptr + 1
 2040 :
 2045 rem *--data file--*
 2050 dopen#2,"Inventory Record",l100
 2055 record#2,(ptr)
 2060 :
 2065 record#2,(ptr),1
 2070 print#2,item$
 2075 :
 2080 record#2,(ptr),25
 2085 print#2,serl$
 2090 :
 2095 record#2,(ptr),40
 2100 print#2,cst$
 2105 :
 2110 record#2,(ptr),50
 2115 print#2,room$
 2120 :
 2125 record#2,(ptr),70
 2130 print#2,desc$
 2135 :
 2140 record#2,(ptr)
 2145 dclose#2
 2150 :
 2155 rem *--update pointer file--*
 2160 dopen#1,"@Inv.Rec.Ptr",w
 2165 print#1,ptr
 2170 dclose#1
 2175 :
 2180 :
 2185 goto 4000:rem repeat routine
 2190 :
 2195 :
 3000 rem **--first time use only--**
 3005 dclose#1
 3010 dopen#1,"Inv.Rec.Ptr",w
 3015 print#1,"0"
 3020 dclose#1
 3025 :
 3030 goto 2000:rem begin file routine again
 3035 :
 3040 :
 4000 rem **--repeat routine--**
 4005 scnclr
 4010 char 0,col,row,"Do you want to add more items? "
 4015 gosub 18000:rem y/n input routine
 4020 if yes$ = "y" then 1000:rem get more info
 4025 if yes$ = "n" then 6000:rem return to main menu
 4030 :
 4035 :
 6000 rem **--return to program menu--**
 6005 scnclr
 6010 char 0,15,9
 6015 color chtr,red
 6020 print "LOADING THE HOMEINV.MENU PROGRAM"
 6025 color chtr,yellow
 6030 run "homeinv.menu"
 6035 :
 6040 :
14000 rem *--incorrect choice message--*
14005 print:print
14010 color chtr,white:print tab(col) "Incorrect Choice!"
14015 color chtr,yellow:print:print tab(col) "Press ";
14020 color chtr,white:print "RETURN";
14025 color chtr,yellow:print " to continue:";
14030 gosub 19000:rem getkey subroutine
14035 return
14040 :
```

```
14045 :
14050 :
15000 rem **--pause routine--**
15005 color chtr,purple
15010 char 0,col,22,"Press the "
15015 color chtr,white
15020 print "RETURN";
15025 color chtr,purple
15030 print " key to continue: ";
15035 gosub 19000:rem getkey subroutine
15040 scnclr
15045 return
15050 :
15055 :
15060 :
17000 rem **--convert to lowercase--**
17005 nc$ = ""
17010 for cv = 1 to len(cv$)
17015 x = asc(mid$(cv$,cv,1))
17020 if x > 192 then x = x - 128
17025 nc$ = nc$ + chr$(x)
17030 next cv
17035 :
17040 cv$ = nc$
17045 f = asc(left$(cv$,1))
17050 f = f + 128
17055 nc$ = chr$(f) + right$(cv$,len(cv$)- 1)
17060 return
17065 :
17070 rem na$ = originally typed name
17075 rem cv$ = converted to lowercase
17080 rem nc$ = 1st letter/uppercase
17085 :
17090 :
17095 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 gosub 14000:goto 1800
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
18090 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print chr$(157) " ";:rem cursor left
19052 print chr$(157);
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
```

```
19070 :
19075 :
21000 rem **--underline routine--**
21005 for i = 1 to sp
21010 print chr$(2);" ";:rem underline on
21015 next i
21020 print chr$(130):rem underline off
21025 return
```

The homeInv.read Program

```
  100 rem ***--read inventory records--***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white =  2
  130 ::::rd =  3
  135 :green =  6
  140 ::blue =  7
  145 yellow = 10
  150 purple =  9
  155 drkyel = 10
  160 ::cyan = 12
  165 ::gray = 16
  170 ::chtr =  5
  175 bckgnd =  0
  195 :
  200 blank$ = chr$(10):rem blank line
  205 print chr$(14):rem upper/lower case mode
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 fast
  225 :
  230 :
  300 rem **--variable list--**
  305 rem item$ = name of item inventoried
  310 rem serl$ = serial # of item
  315 rem  cst$ = cost of item
  320 rem room$ = location of item
  322 rem desc$ = description of item
  325 rem   cv$ = convert to lower case
  330 rem srch$ = search for information
  335 rem find$ = information in data file
  340 rem info$ = associated srch$ information
  345 rem  pztn = position within record
  350 rem  byte = position within record
  355 rem   ptr = pointer for number of records
  360 rem   tp$ = full type message
  365 rem   col = column number
  370 rem   row = row number
  375 rem other variables are descriptive
  380 :
  385 :
 2000 rem **--read file routine--**
 2002 color chtr,yellow
 2005 gosub 10000:rem display routine
 2010 if file = 4 then gosub 22000:rem titles routine
 2015 :
 2020 rem *--pointer file--*
 2025 dopen#1,"Inv.Rec.Ptr"
 2030 input#1,ptr
 2035 dclose#1
 2040 :
 2045 rem *--titles--*
 2050 scnclr:print chr$(2);
 2055 char 0,0,0,"ITEM"
 2060 char 0,20,0,"SER#"
 2065 char 0,35,0,"COST"
```

```
2070 char 0,45,0,"ROOM"
2072 char 0,60,0,"DESC"
2075 print chr$(130)
2080 :
2085 rem *--data file--*
2090 window 0,2,79,24
2095 dopen#2,"Inventory Record"
2100 for rec = 1 to ptr
2105 :
2110 record#2,(rec),1
2115 input#2,item$
2120 :
2125 record#2,(rec),25
2130 input#2,serl$
2135 :
2140 record#2,(rec),40
2145 input#2, cst$
2150 :
2155 record#2,(rec),50
2160 input#2,room$
2165 :
2166 record#2,(rec),70
2167 input#2,desc$
2168 :
2170 :
2175 rem *--display information--*
2185 if file = 4 then gosub 23000:rem printer display
2190 char 0, 0,rec,(item$)
2195 char 0,20,rec,(serl$)
2200 char 0,35,rec,(cst$)
2205 char 0,45,rec,(room$)
2207 char 0,60,rec,(desc$)
2210 next rec
2215 :
2220 dclose#2
2225 gosub 15000:rem pause routine
2230 window 0,0,79,24,1
2235 close file
2245 :
2250 :
2255 :
6000 rem **--return to program menu--**
6005 scnclr
6010 char 0,15,9
6015 color chtr,rd
6020 print "LOADING THE HOMEINV.MENU PROGRAM"
6025 color chtr,yellow
6030 run "homeinv.menu"
6035 :
6040 :
10000 rem **--display subroutine--**
10005 scnclr
10010 char 0,0,3
10015 print "Would you like a paper print out? ";
10020 gosub 18000:rem y/n input subroutine
10025 if yes$ = "y" then 10040:rem printer
10030 if yes$ = "n" then 10115:rem screen
10035 :
10040 rem *--print out is desired--*
10045 scnclr
10050 char 0,0,3
10055 print "Please make sure the printer is on and ready to use."
10060 print
10065 print "Are you ready to begin printing? ";
10070 gosub 18000:rem y/n input subroutine
10075 if yes$ = "n" then 10000:rem ask again
10080 :
10085 rem *--printer display--*
10090 file = 4
```

```
10095 dvic = 4
10100 cmnd = 7
10105 goto 10150:rem open instruction
10110 :
10115 rem *--screen display--*
10120 file = 3
10125 dvic = 3
10130 cmnd = 1
10135 goto 10150:rem open instruction
10140 :
10145 :
10150 rem *--open instruction--*
10155 open file,dvic,cmnd
10160 return
10165 :
10170 :
10175 :
14000 rem *--incorrect choice message--*
14005 print:print
14010 color chtr,white:print tab(col) "Incorrect Choice!"
14015 color chtr,yellow:print:print tab(col) "Press ";
14020 color chtr,white:print "RETURN";
14025 color chtr,yellow:print " to continue:";
14030 gosub 19000:rem getkey subroutine
14035 return
14040 :
14045 :
14050 :
15000 rem **--pause routine--**
15005 color chtr,purple
15010 char 0,col,22,"Press the "
15015 color chtr,white
15020 print "RETURN";
15025 color chtr,purple
15030 print " key to continue: ";
15035 gosub 19000:rem getkey subroutine
15040 print
15045 return
15050 :
15055 :
15060 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 print
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
18090 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
```

```
19045 if lg < 1 then 19000
19050 print a$;
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
19070 :
19075 :
22000 rem **--printer titles routine--**
22005 print#file,""chr$(12)
22010 print#file,""tab(25) "LIST OF HOME INVENTORY RECORDS"
22015 print#file," "
22020 print#file,""tab(00) "ITEM" spc(17) "SER#";
22025 print#file,""spc(12) "COST" spc(18) "ROOM"
22027 print#file,""spc(12) "DESC"
22030 return
22035 :
22040 :
22045 :
23000 rem **--printer display--**
23005 print#file,""tab(00) item$ spc(20 - len(item$))serl$;
23010 print#file,""spc(14 - len(serl$)) cst$;
23015 print#file,""spc(23 - len(cst$)) room$;
23017 print#file,""spc(23 - len(room$)) desc$;
23020 if rec > 50 then gosub 22000
23025 return
```

The homeInv.search Program

```
  100 rem ***--home inventory search/sort--***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white = 16
  130 ::::rd =  3
  135 :green =  6
  140 ::blue =  7
  145 yellow = 10
  150 purple =  9
  155 drkyel = 10
  160 ::cyan = 12
  165 ::gray = 16
  170 ::chtr =  5
  175 bckgnd =  0
  195 :
  200 blank$ = chr$(10):rem blank line
  205 print chr$(14):rem upper/lower case mode
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 fast
  225 ttl$ = "SEARCH/SORT DISPLAY MENU"
  230 ln = len(ttle$)
  235 center = (80 - ln) / 2:rem center title
  245 :
  250 :
  300 rem **--variable list--**
  305 rem item$ = name of item inventoried
  310 rem serl$ = serial # of item
  315 rem  cst$ = cost of item
  320 rem room$ = location of item
  322 rem desc$ = description of item
  325 rem   cv$ = convert to lower case
  330 rem srch$ = search for information
  335 rem find$ = information in data file
  340 rem info$ = associated srch$ information
  345 rem  pztn = position within record
  350 rem  byte = position within record
  355 rem   ptr = pointer for number of records
  360 rem   tp$ = full type message
  365 rem   col = column number
```

```
370 rem   row = row number
375 rem other variables are descriptive
380 :
385 :
400 rem **--pointer file--**
405 dopen#1,"Inv.Rec.Ptr"
410 input#1,ptr
415 dclose#1
420 :
425 :
500 rem **--home inventory menu--**
505 scnclr
510 color chtr,purple
515 :
520 rem *--draw top line--*
525 for i = 2 to 79
530 print chr$(2);" ";
535 next i
540 print chr$(130)
545 :
550 rem *--display choices--*
555 col = center:row = 2
560 color chtr,white
565 char 0,4,row,"Inventory"
570 char 0,col,row,(ttle$)
572 char 0,60,row,"Select Action"
575 color chtr,yellow
580 col =  8:row = 4
585 char 0,col,row +  2,"1.  Search For Item"
590 char 0,col,row +  4,"2.  Search For Serial #"
595 char 0,col,row +  6,"3.  Search For Cost"
600 char 0,col,row +  8,"4.  Search For Room Items"
605 col = 45:row = 4
610 char 0,col,row +  2,"5.  Sort Items--Alpha.Order"
615 char 0,col,row +  4,"6.  Sort Items--Serial #"
620 char 0,col,row +  6,"7.  Return to Home Inv.Menu"
625 col = 4:row = 4
630 char 0,col,row + 14
635 print "Type your Selection ";
640 color chtr,white:print "(1-7)";
645 color chtr,yellow:print " and press ";
650 color chtr,white:print "RETURN";
655 color chtr,yellow:print ": ---> ";
660 x = pos(0)
665 color chtr,purple:char 0,0,row + 15
670 :
675 rem *--draw bottom line--*
680 for i = 2 to 79
685 print chr$(2);" ";
690 next i
695 print chr$(130)
700 :
705 rem *--options--*
710 color chtr,yellow
715 char 0, 4,row + 17,"Options: "
720 color chtr,white:print "ESC";:
725 color chtr,yellow:print " key to leave.  ";
730 color chtr,white:print "HELP";
735 color chtr,yellow:print " key for information."
740 :
745 rem *--wait for keypress--*
750 char 0,x,row + 14
755 gosub 19000:rem getkey routine
760 number$ = t$:t$ = ""
765 if number$ = chr$(13) then 750
770 number = val(number$)
775 :
780 if number = 1 then 1000
785 if number = 2 then 2000
```

```
 790 if number = 3 then 3000
 795 if number = 4 then 4000
 800 if number = 5 then 5000
 805 if number = 6 then 6000
 810 if number = 7 then 7000
 820 :
 825 rem *--incorrect choice message--*
 830 gosub 14000:rem incorrect choice
 835 goto 500:rem menu--check again
 840 :
 845 :
1000 rem **--search for item--**
1005 b$ = "item"
1010 pztn = 1
1015 gosub 13000:rem common search routine
1020 goto 500:rem return to menu
1025 :
2000 rem **--search for serial #--**
2005 b$ = "serial number"
2010 pztn = 25
2015 gosub 13000:rem common search routine
2020 goto 500:rem return to menu
2025 :
3000 rem **--search for cost--**
3005 ttlamt = 0:rem zero out total amt
3007 col = 25:row = 5
3010 scnclr
3015 color chtr,white
3020 char 0,col,row,"Search For Items...."
3025 color chtr,yellow
3030 char 0,col,row + 2,"A....Above a certain amount"
3035 char 0,col,row + 4,"B....Below a certain amount"
3040 char 0,col,row + 6,"Which letter "
3045 color chtr,white
3050 print "A";
3055 color chtr,yellow
3060 print " or ";
3065 color chtr,white
3070 print "B";
3075 color chtr,yellow
3080 print ": ";
3085 gosub 19000:rem getkey subroutine
3090 lt$ = t$:t$ = ""
3095 if lt$ = "A" or lt$ = "a" then 3200
3100 if lt$ = "B" or lt$ = "b" then 3500
3105 gosub 14000:rem incorrect choice
3110 goto 3000:rem ask again
3115 :
3120 :
3200 rem *--items above $ amount--*
3205 char 0,col,row + 10,"Above which amount? "
3210 gosub 19000:rem getkey subroutine
3215 amt$ = t$:t$ = ""
3220 amt = val(amt$)
3225 gosub 10000:rem display subroutine
3230 scnclr
3235 col = 25:row = 2
3240 print#file,"" tab(25) "Items Above ";
3245 print#file,using "#$####.##";amt
3250 print#file,blank$:print#file,blank$
3255 dopen#2,"Inventory Record",l100
3260 for w = 1 to ptr
3265 :
3270 record#2,w,1
3275 input#2,item$
3280 :
3285 record#2,w,40
3290 input#2,cst$
```

```
3295 :
3300 c$ = cst$:cst = val(cst$)
3305 if left$(c$,7) = "DELETED" then 3340:rem next w
3310 if left$(c$,1) = "$" then gosub 12000:rem strip $
3315 if val(c$) > amt then 3325
3320 goto 3340:rem next w
3325 ttlamt = ttlamt + val(c$)
3330 print#file,item$;
3335 print#file,"" tab(30):print#file,using "#$####.##";cst
3340 next w
3345 :
3350 print
3355 print#file,"Total Value = ";
3360 print#file,"" tab(30)
3365 print#file,using "#$####.##";ttlamt
3370 dclose#2:print#file:close file
3375 gosub 24000:rem clear variables
3380 gosub 15000:rem pause routine
3385 goto 500
3390 :
3395 :
3500 rem *--items below $ amount--*
3505 char 0,col,row + 10,"Below which amount? "
3510 gosub 19000:rem getkey subroutine
3515 amt$ = t$:t$ = ""
3520 amt = val(amt$)
3525 gosub 10000:rem display subroutine
3530 scnclr
3535 col = 25:row = 2
3540 print#file,"" tab(25) "Items Below ";
3545 print#file,using "#$####.##";amt
3550 print#file,blank$:print#file,blank$
3555 dopen#2,"Inventory Record",l100
3560 for w = 1 to ptr
3565 :
3570 record#2,w,1
3575 input#2,item$
3580 :
3585 record#2,w,40
3590 input#2,cst$
3595 :
3600 c$ = cst$:cst = val(cst$)
3605 if left$(c$,7) = "DELETED" then 3640:rem next w
3610 if left$(c$,1) = "$" then gosub 12000:rem strip $
3615 if val(c$) < amt then 3625
3620 goto 3640:rem next w
3625 ttlamt = ttlamt + val(c$)
3630 print#file,item$;
3635 print#file,"" tab(30):print#file,using "#$####.##";cst
3640 next w
3645 :
3650 print
3655 print#file,"Total Value = ";
3660 print#file,"" tab(30)
3665 print#file,using "#$####.##";ttlamt
3670 dclose#2:print#file:close file
3675 gosub 24000:rem clear variables
3680 gosub 15000:rem pause routine
3685 goto 500
3690 :
3695 :
3700 :
4000 rem **--search for room items--**
4005 b$ = "room"
4010 pztn = 50
4015 gosub 13000:rem common search routine
4020 goto 500:rem return to menu
4025 :
```

```
4030 :
4035 :
5000 rem **--sort items alpha.order--**
5005 scnclr
5010 color chtr,red
5015 col = 25:row = 5
5020 print chr$(15);:rem flash
5025 char 0,col,row,"WORKING--PLEASE DON'T TOUCH!"
5030 print chr$(143):rem flash off
5035 color chtr,yellow
5040 :
5045 rem **--get information from file--**
5050 q = 1
5055 dopen#2,"Inventory Record",l100
5060 for w = 1 to ptr
5065 :
5070 record#2,w,1
5075 input#2,item$
5080 :
5085 if item$ = "DELETED" then 5130:rem next w
5090 :
5095 rem *--convert to upper case--*
5100 cv$ = item$
5105 gosub 17500:rem upper case
5110 item$ = cv$
5115 :
5120 a$(q) = item$:rem store in array for internal sort
5125 q = q + 1
5130 next w
5135 :
5140 :
5145 dclose#2
5150 n = q - 1
5155 color chtr,rd
5160 print chr$(15);:rem flash
5165 char 0,col,row,"STILL WORKING--PLEASE WAIT!"
5170 print chr$(143):rem flash off
5175 color chtr,yellow
5180 gosub 25000:rem sort routine
5185 t$ = ""
5190 :
5195 :
5200 rem *--display results--*
5205 scnclr
5210 gosub 10000:rem display routine
5215 scnclr
5220 color chtr,white
5225 print#file,"" tab(col) "Items Sorted By Alphabetical Order"
5230 color chtr,yellow
5235 print#file," ":print#file," "
5240 for i = 1 to q - 1
5245 print#file,using "####";i;
5250 print#file,". ";a$(i)
5255 next i
5260 :
5265 gosub 24000:rem clear variables
5270 print#file,"":close file
5275 gosub 15000:rem pause routine
5280 goto 500:rem return to menu
5285 :
5290 :
5295 :
6000 rem **--sort items serial #--**
6005 scnclr
6010 color chtr,red
6015 col = 25:row = 5
6020 print chr$(15);:rem flash
6025 char 0,col,row,"WORKING--PLEASE DON'T TOUCH!"
```

```
6030 print chr$(143):rem flash off
6035 color chtr,yellow
6040 :
6045 rem **--get information from file--**
6050 q = 1
6055 dopen#2,"Inventory Record",l100
6060 for w = 1 to ptr
6065 :
6070 record#2,w,1
6075 input#2,item$
6077 :
6078 record#2,w,25
6079 input#2,serl$
6080 :
6085 if item$ = "DELETED" then 6130:rem next w
6090 :
6095 rem *--convert to upper case--**
6100 cv$ = serl$
6105 gosub 17500:rem upper case
6110 serl$ = cv$
6115 :
6120 a$(q) = serl$ + "*" + item$
6125 q = q + 1
6130 next w
6135 :
6140 :
6145 dclose#2
6150 n = q - 1
6155 color chtr,rd
6160 print chr$(15);:rem flash
6165 char 0,col,row,"STILL WORKING--PLEASE WAIT!"
6170 print chr$(143):rem flash off
6175 color chtr,yellow
6180 gosub 25000:rem sort routine
6185 t$ = ""
6190 :
6195 :
6200 rem *--display results--*
6205 scnclr
6210 gosub 10000:rem display routine
6215 scnclr
6220 color chtr,white
6225 print#file,"" tab(col) "Items Sorted By Serial Number"
6230 color chtr,yellow
6235 print#file," ":print#file," "
6240 :
6245 for i = 1 to q - 1
6250 la = len(a$(i))
6255 j = instr(a$(i),"*")
6260 print#file,using "####";i;
6265 print#file,". ";left$(a$(i),j - 1);
6267 if file = 4 then 23000
6270 print#file,""tab(25) mid$(a$(i),j + 1,la)
6275 next i
6280 :
6285 gosub 24000:rem clear variables
6290 print#file,"":close file
6295 gosub 15000:rem pause routine
6300 goto 500:rem return to menu
6305 :
6310 :
6315 :
7000 rem **--return to program menu--**
7005 scnclr
7010 char 0,15,9
7015 color chtr,red
7020 print "LOADING THE HOMEINV.MENU PROGRAM"
7025 color chtr,yellow
```

```
7030 run "homeinv.menu"
7035 :
7040 :
9000 rem **--information section--**
9002 sm$ = "Inventory Record File"
9005 color chtr,cyan
9010 getkey b$,c$,d$
9012 char 0,4,row + 17
9013 print chr$(27)+chr$(81);
9015 input "Which Selection do you need information about";e$
9017 getkey e$
9018 e = val(e$)
9020 color chtr,yellow
9021 if e = 1 then 9100
9022 if e = 2 then 9200
9023 if e = 3 then 9300
9024 if e = 4 then 9400
9025 if e = 5 then 9500
9026 if e = 6 then 9600
9027 if e = 7 then 9700
9028 gosub 14000:rem incorrect choice
9098 :
9099 :
9100 rem **--information about option #1--**
9105 scnclr
9110 char 0,4,7,"This option will search the entire " + sm$
9115 print:print
9120 print "    for all items that match the specified item."
9155 gosub 15000:rem pause routine
9195 goto 500:rem return to menu
9198 :
9199 :
9200 rem **--information about option #2--**
9205 scnclr
9210 char 0,4,7,"This option will search the entire " + sm$
9215 print:print
9220 print "    for all serial numbers that match the specified serial #."
9245 gosub 15000:rem pause routine
9295 goto 500:rem return to menu
9298 :
9299 :
9300 rem **--information about option #3--**
9305 scnclr
9310 char 0,4,7,"This option is the heart of the " + sm$ + ". You are "
9315 print:print
9355 gosub 15000:rem pause routine
9395 goto 500:rem return to menu
9398 :
9399 :
9400 rem **--information about option #4--**
9405 scnclr
9410 char 0,4,7,"This option will search the entire " + sm$
9415 print:print
9420 print "    for all rooms that match the specified room."
9455 gosub 15000:rem pause routine
9495 goto 500:rem return to menu
9498 :
9499 :
9500 rem **--information about option #5--**
9505 scnclr
9510 char 0,4,7,"This option will sort and display the entire " + sm$
9515 print:print
9520 print "    by item name in alphabetical order."
9545 gosub 15000:rem pause routine
9595 goto 500:rem return to menu
9598 :
9599 :
9600 rem **--information about option #6--**
```

```
 9605 scnclr
 9610 char 0,4,7,"This option will sort and display the entire " + sm$
 9615 print:print
 9620 print "    by serial number."
 9635 gosub 15000:rem pause routine
 9695 goto 500:rem return to menu
 9698 :
 9699 :
 9700 rem **--information about option #7--**
 9705 scnclr
 9710 char 0,4,7,"This option allows you to return to the main program menu."
 9715 print:print:
 9735 gosub 15000:rem pause routine
 9740 goto 500:rem return to menu
 9798 :
 9799 :
10000 rem **--display subroutine--**
10010 color chtr,yellow
10012 char 0,4,row + 17
10014 print chr$(27)+chr$(81);:rem clear line
10015 print "Would you like a paper print out? ";
10020 gosub 18000:rem y/n input subroutine
10025 if yes$ = "y" then 10040:rem printer
10030 if yes$ = "n" then 10115:rem screen
10035 :
10040 rem *--print out is desired--*
10045 scnclr
10050 char 0,0,3
10055 print "Please make sure the printer is on and ready to use."
10060 print
10065 print "Are you ready to begin printing? ";
10070 gosub 18000:rem y/n input subroutine
10075 if yes$ = "n" then 10000:rem ask again
10080 :
10085 rem *--printer display--*
10090 file = 4
10095 dvic = 4
10100 cmnd = 7
10105 goto 10150:rem open instruction
10110 :
10115 rem *--screen display--*
10120 file = 3
10125 dvic = 3
10130 cmnd = 1
10135 goto 10150:rem open instruction
10140 :
10145 :
10150 rem *--open instruction--*
10155 open file,dvic,cmnd
10160 return
10165 :
10170 :
10175 :
11000 rem **--line up numbers--**
11005 if i < 10 then ta = 4
11010 if i > 9 and i < 100 then ta = 3
11015 if i > 99 and i < 1000 then ta = 2
11020 if i > 999 then ta = 1
11025 return
11030 :
11035 :
11040 :
12000 rem **--strip $ sign--**
12010 lc = len(c$)
12020 a$ = right(c$,lc - 1)
12030 c$ = a$
12040 return
12050 :
```

```
12060 :
12070 :
13000 rem **--common search routine--**
13005 color chtr,cyan
13010 char 0,4,row + 17
13015 print chr$(27)+chr$(81);:rem clear line
13020 print "Search for which ";b$;"? ";
13025 gosub 19000:rem getkey subroutine
13030 srch$ = t$:t$ = ""
13035 gosub 10000:rem display routine
13040 color chtr,red
13045 char 0,4,row + 17
13050 print chr$(27)+chr$(81);:rem clear line
13055 print chr$(15);:print "Searching......Please Wait! "
13060 print chr$(143):rem turn flash off
13065 :
13070 rem *--convert to lowercase--*
13075 cv$ = srch$
13080 gosub 17000:rem convert routine
13085 srch$ = cv$
13090 :
13095 rem *--file input routine--*
13100 :
13105 rem *--data file--*
13110 dopen#2,"Inventory Record"
13115 ln = len(srch$)
13120 :
13125 for rec = 1 to ptr
13130 record#2,(rec),(pztn)
13135 input#2,find$
13140 :
13145 rem *--convert to lowercase--*
13150 cv$ = left$(find$,ln)
13155 gosub 17000:rem convert routine
13160 if srch$ <> cv$ then 13315:rem next record
13165 :
13170 rem *--matched, now get rest of info.--*
13175 for k = 1 to 5
13180 if k = 1 then byte = 1
13185 if k = 2 then byte = 25
13190 if k = 3 then byte = 40
13195 if k = 4 then byte = 50
13200 if k = 5 then byte = 70
13205 record#2,(rec),(byte)
13210 input#2,info$(k)
13215 next k
13220 :
13225 :
13230 :
13235 rem *--display information--*
13240 scnclr
13245 color chtr,white
13250 char 0,col,row,"HOME INVENTORY INFORMATION"
13255 color chtr,drkyel
13260 print
13265 :
13270 print:print#file,""tab(col) "1.  Item: ";info$(1)
13275 print:print#file,""tab(col) "2.  Ser#: ";info$(2)
13280 print:print#file,""tab(col) "3.  Cost: ";info$(3)
13285 print:print#file,""tab(col) "4.  Room: ";info$(4)
13290 print:print#file,""tab(col) "5.  Desc: ";info$(5)
13295 print:print#file,""tab(col) " "
13300 gosub 15000:rem pause routine
13305 :
13310 :
13315 next rec
13320 :
13325 :
```

```
13330 dclose#2
13335 scnclr
13340 color chtr,white
13345 char 0,col,row,"Search Completed!"
13350 color chtr,drkyel
13355 gosub 15000:rem pause routine
13360 close file
13365 return
13370 :
13375 :
13380 :
14000 rem *--incorrect choice message--*
14005 print:print
14010 color chtr,white:print tab(col) "Incorrect Choice!"
14015 color chtr,yellow:print:print tab(col) "Press ";
14020 color chtr,white:print "RETURN";
14025 color chtr,yellow:print " to continue:";
14030 gosub 19000:rem getkey subroutine
14035 return
14040 :
14045 :
14050 :
15000 rem **--pause routine--**
15005 color chtr,purple
15010 char 0,col,22,"Press the "
15015 color chtr,white
15020 print "RETURN";
15025 color chtr,purple
15030 print " key to continue: ";
15035 gosub 19000:rem getkey subroutine
15040 scnclr
15045 return
15050 :
15055 :
15060 :
17000 rem **--convert to lowercase--**
17005 nc$ = ""
17010 for cv = 1 to len(cv$)
17015 x = asc(mid$(cv$,cv,1))
17020 if x > 192 then x = x - 128
17025 nc$ = nc$ + chr$(x)
17030 next cv
17035 :
17040 cv$ = nc$
17045 f = asc(left$(cv$,1))
17050 f = f + 128
17055 nc$ = chr$(f) + right$(cv$,len(cv$)- 1)
17060 return
17065 :
17070 rem na$ = originally typed name
17075 rem cv$ = converted to lowercase
17080 rem nc$ = 1st letter/uppercase
17085 :
17090 :
17095 :
17500 rem **--convert to uppercase--**
17510 nc$ = ""
17520 for cv = 1 to len(cvt$)
17530 x = asc(mid$(cvt$,cv,1))
17540 if x > 64 and x < 91 then x = x + 128
17550 nc$ = nc$ + chr$(x)
17560 next cv
17570 :
17580 cvt$ = nc$
17590 return
17600 :
17610 :
17620 :
```

```
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 print
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
18090 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19007 if a$ = chr$(27) then 7000
19008 if a$ = chr$(72) then 9000:rem help key
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print a$;
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
19070 :
19075 :
21000 rem **--underline routine--**
21005 for i = 1 to sp
21010 print chr$(2);" ";:rem underline on
21015 next i
21020 print chr$(130):rem underline off
21025 return
21030 :
21035 :
21040 :
23000 rem **--printer display--**
23005 serl$ = left$(a$(i),j - 1)
23010 print#file,""spc(25 - len(serl$)) mid$(a$(i),j + 1,la)
23025 goto 6275
23030 :
23035 :
23040 :
24000 rem **--clear variables--**
24010 item$ = ""
24020 serl$ = ""
24030 cst$ = ""
24040 room$ = ""
24050 desc$ = ""
24055 scf = 0
24060 return
24070 :
24080 :
24090 :
25000 rem **--shell-metzner sort--**
25010 m = n
25020 m = int (m / 2)
25030 if m = 0 then 25150
```

```
25040 j = 1:k = n - m
25050 i = j
25060 z = i + m
25070 if a$(i) < a$(z) then 25120
25080 t$ = a$(i):a$(i) = a$(z):a$(z) = t$
25090 i = i - m
25100 if i < 1 then 25120
25110 goto 25060
25120 j = j + 1
25130 if j > k then 25020
25140 goto 25050
25150 return
```

The homeInv.correct Program

```
  100 rem ***--home inventory correction--***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white = 16
  130 ::::rd =  3
  135 :green =  6
  140 ::blue =  7
  145 yellow = 10
  150 purple =  9
  155 drkyel = 10
  160 ::cyan = 12
  165 ::gray = 16
  170 ::chtr =  5
  175 bckgnd =  0
  195 :
  200 blank$ = chr$(10):rem blank line
  205 print chr$(14):rem upper/lower case mode
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 fast
  225 ttl$ = "CORRECT/DELETE MENU"
  230 ln = len(ttle$)
  235 center = (80 - ln) / 2:rem center title
  245 :
  250 :
  300 rem **--variable list--**
  305 rem item$ = name of item inventoried
  310 rem serl$ = serial # of item
  315 rem  cst$ = cost of item
  320 rem room$ = location of item
  322 rem desc$ = description of item
  325 rem   cv$ = convert to lower case
  330 rem srch$ = search for information
  335 rem find$ = information in data file
  340 rem info$ = associated srch$ information
  345 rem  pztn = position within record
  350 rem  byte = position within record
  355 rem   ptr = pointer for number of records
  360 rem   tp$ = full type message
  365 rem   col = column number
  370 rem   row = row number
  375 rem other variables are descriptive
  380 :
  385 :
  400 rem **--pointer file--**
  405 dopen#1,"Inv.Rec.Ptr"
  410 input#1,ptr
  415 dclose#1
  420 :
  425 :
  500 rem **--correct/delete menu--**
  505 scnclr
```

```
 510 color chtr,purple
 515 :
 520 rem *--draw top line--*
 525 for i = 2 to 79
 530 print chr$(2);" ";
 535 next i
 540 print chr$(130)
 545 :
 550 rem *--display choices--*
 555 col = center:row = 2
 560 color chtr,white
 565 char 0,4,row,"Inventory"
 570 char 0,col,row,(ttle$)
 572 char 0,60,row,"Select Action"
 575 color chtr,yellow
 580 col = 24:row = 5
 585 char 0,col,row +  2,"1.  Correct Inventory Record"
 590 char 0,col,row +  4,"2.  Delete  Inventory Record"
 595 char 0,col,row +  6,"3.  Return to Home Inv. Menu"
 625 col = 4:row = 4
 630 char 0,col,row + 14
 635 print "Type your Selection ";
 640 color chtr,white:print "(1-7)";
 645 color chtr,yellow:print " and press ";
 650 color chtr,white:print "RETURN";
 655 color chtr,yellow:print ": ---> ";
 660 x = pos(0)
 665 color chtr,purple:char 0,0,row + 15
 670 :
 675 rem *--draw bottom line--*
 680 for i = 2 to 79
 685 print chr$(2);" ";
 690 next i
 695 print chr$(130)
 700 :
 705 rem *--options--*
 710 color chtr,yellow
 715 char 0, 4,row + 17,"Options: "
 720 color chtr,white:print "ESC";:
 725 color chtr,yellow:print " key to leave.  ";
 730 color chtr,white:print "HELP";
 735 color chtr,yellow:print " key for information."
 740 :
 745 rem *--wait for keypress--*
 750 char 0,x,row + 14
 755 gosub 19000:rem getkey routine
 760 number$ = t$:t$ = ""
 765 if number$ = chr$(13) then 750
 770 number = val(number$)
 775 :
 780 if number = 1 then 1000
 785 if number = 2 then 2000
 790 if number = 3 then 3000
 820 :
 825 rem *--incorrect choice message--*
 830 gosub 14000:rem incorrect choice
 835 goto 500:rem menu--check again
 840 :
 845 :
1000 rem **--correct record routine--**
1002 flag$ = "no":rem info not yet changed
1005 color chtr,cyan
1010 char 0,4,row + 17
1015 print chr$(27) + chr$(81);:rem clear line
1020 print "Correct which record? (";
1025 color chtr,white:print "1";
1030 color chtr,cyan:print " to";
1035 color chtr,white:print ptr;
1040 color chtr,cyan:print "--Type a ";
```

```
1045 color chtr,white:print "0";
1050 color chtr,cyan:print " when finished.) ";
1055 gosub 19000:rem getkey routine
1060 rc$ = t$:t$ = ""
1065 rc = val(rc$)
1070 if rc = 0 then 500:rem menu
1075 if rc > ptr then gosub 14000:goto 1000:rem ask again
1080 :
1085 gosub 5000:rem file input routine
1090 :
1500 rem *--display for correction--*
1510 scnclr
1515 col = 17:row = 2
1520 color chtr,white
1525 char 0,col,row,"ENTERED INFORMATION"
1530 color chtr,drkyel
1535 char 0,col,row +  2,"1.  Item: " + item$
1540 char 0,col,row +  4,"2.  Ser#: " + serl$
1545 char 0,col,row +  6,"3.  Cost: " + cst$
1550 char 0,col,row +  8,"4.  Room: " + room$
1555 char 0,col,row + 10,"5.  Desc: " + desc$
1575 char 0,col,row + 12,"Is this correct? "
1580 gosub 18000:rem y/n input routine
1585 if yes$ = "y" and flag$  = "yes" then 6000:rem file output routine
1587 if yes$ = "y" and flag$  = "no" then 1000:rem info has not been changed
1588 if yes$ = "n" then 1595
1589 char 0,col,row + 12
1590 print chr$(27) + chr$(81);
1591 goto 1575:rem ask again
1592 :
1595 char 0,col,row + 14,"Which number is wrong? "
1600 gosub 19000:rem getkey subroutine
1605 nb$ = t$:t$ = ""
1610 nb = val(nb$)
1615 if nb < 1 or nb > 5 then gosub 14000:goto 1595
1616 if nb = 1 then sp = 23
1617 if nb = 2 then sp = 14
1618 if nb = 3 then sp =  9
1619 if nb = 4 then sp = 19
1620 if nb = 5 then sp = 29
1621 color chtr,rd
1622 char 0,col,row + 16,"Do not go beyond the end of the line!"
1623 color chtr,yellow
1625 char 0,col,row + 18,"Type in the correct information: "
1626 char 0,col,row + 20
1627 gosub 21000:rem underline routine
1628 print chr$(2);
1629 char 0,col,row + 20
1630 gosub 19000:rem getkey subroutine
1631 print chr$(130)
1635 cinfo$ = t$:t$ = ""
1637 flag$ = "yes"
1640 if nb = 1 then item$ = left$(cinfo$,23):goto 1500:rem ask again
1645 if nb = 2 then serl$ = left$(cinfo$,14):goto 1500:rem ask again
1650 if nb = 3 then  cst$ = left$(cinfo$,9) :goto 1500:rem ask again
1655 if nb = 4 then room$ = left$(cinfo$,19):goto 1500:rem ask again
1660 if nb = 5 then desc$ = left$(cinfo$,29):goto 1500:rem ask again
1690 :
1695 :
2000 rem **--delete record routine--**
2005 color chtr,cyan
2010 char 0,4,row + 17
2015 print chr$(27) + chr$(81);:rem clear line
2020 print "Delete which record? (";
2025 color chtr,white:print "1";
2030 color chtr,cyan:print " to";
2035 color chtr,white:print ptr;
2040 color chtr,cyan:print "--Type a ";
2045 color chtr,white:print "0";
```

```
2050 color chtr,cyan:print " when finished.) ";
2055 gosub 19000:rem getkey routine
2060 rc$ = t$:t$ = ""
2065 rc = val(rc$)
2070 if rc = 0 then 500:rem menu
2075 if rc > ptr then gosub 14000:goto 2000:rem ask again
2080 :
2085 gosub 5000:rem file input routine
2090 :
2500 rem *--display for deletion--*
2510 scnclr
2515 col = 17:row = 2
2520 color chtr,white
2525 char 0,col,row,"FILE INFORMATION"
2530 color chtr,drkyel
2535 char 0,col,row +  2,"1.  Item: " + item$
2540 char 0,col,row +  4,"2.  Ser#: " + serl$
2545 char 0,col,row +  6,"3.  Cost: " + cst$
2550 char 0,col,row +  8,"4.  Room: " + room$
2555 char 0,col,row + 10,"5.  Desc: " + desc$
2575 char 0,col,row + 12,"Delete this record? "
2580 gosub 18000:rem y/n input routine
2585 if yes$ = "y" then 2595
2587 if yes$ = "n" then 2000:rem start again
2589 char 0,col,row + 12
2590 print chr$(27) + chr$(81);
2591 goto 2575:rem ask again
2592 :
2595 char 0,col,row + 14,"Are you sure? Type "
2596 color chtr,white:print "YES";
2597 color chtr,yellow:print " to delete this record: ";
2600 gosub 19000:rem getkey subroutine
2605 yes$ = t$:t$ = ""
2615 if yes$ = "YES" or yes$ = "yes" then 2630
2620 goto 2000:rem ask again
2625 :
2630 rem *--info deleted--*
2635 item$ = "DELETED"
2640 serl$ = "DELETED"
2645 :cst$ = "DELETED"
2650 room$ = "DELETED"
2655 desc$ = "DELETED"
2660 goto 6000:rem file output routine
2665 :
2670 :
2675 :
3000 rem **--return to program menu--**
3005 goto 7000:rem program menu
3010 :
3015 :
3020 :
5000 rem **--file input routine--**
5005 :
5010 rem *--data file--*
5015 dopen#2,"Inventory Record"
5020 rec = rc
5025 :
5030 record#2,(rec),1
5035 input#2,item$
5040 :
5045 record#2,(rec),25
5050 input#2,serl$
5055 :
5060 record#2,(rec),40
5065 input#2, cst$
5070 :
5075 record#2,(rec),50
5080 input#2,room$
```

```
5085 :
5090 record#2,(rec),70
5095 input#2,desc$
5100 :
5105 dclose#2
5110 return
5115 :
5120 :
6000 rem **--file output routine--**
6005 :
6010 rem *--data file--*
6015 dopen#2,"Inventory Record",l100
6017 rec = rc
6019 record#2,(rec)
6020 :
6025 record#2,(rec),1
6030 print#2,item$
6035 :
6040 record#2,(rec),25
6045 print#2,serl$
6050 :
6055 record#2,(rec),40
6060 print#2,cst$
6065 :
6070 record#2,(rec),50
6075 print#2,room$
6080 :
6085 record#2,(rec),70
6090 print#2,desc$
6095 :
6097 record#2,(rec)
6100 dclose#2
6105 :
6110 goto 500:rem menu
6115 :
6120 :
7000 rem **--return to program menu--**
7005 scnclr
7010 char 0,15,9
7015 color chtr,red
7020 print "LOADING THE HOMEINV.MENU PROGRAM"
7025 color chtr,yellow
7030 run "homeinv.menu"
7035 :
7040 :
9000 rem **--information section--**
9002 sm$ = "Inventory Record File"
9005 color chtr,cyan
9010 getkey b$,c$,d$
9012 char 0,4,row + 17
9013 print chr$(27)+chr$(81);
9015 input "Which Selection do you need information about";e$
9017 getkey e$
9018 e = val(e$)
9020 color chtr,yellow
9021 if e = 1 then 9100
9022 if e = 2 then 9200
9023 if e = 3 then 9300
9028 gosub 14000:rem incorrect choice
9029 char 0,4,row + 17
9030 print chr$(27) + chr$(81);
9031 color chtr,cyan
9032 print "Which Selection do you need information about?";
9033 goto 9017
9098 :
9099 :
9100 rem **--information about option #1--**
9105 scnclr
```

```
9110 char 0,4,7,"This option allows you to correct or change any piece of"
9115 print:print
9120 print "    information in the ";sm$;"."
9155 gosub 15000:rem pause routine
9195 goto 500:rem return to menu
9198 :
9199 :
9200 rem **--information about option #2--**
9205 scnclr
9210 char 0,4,7,"This option allows you to delete any record (set of"
9215 print:print
9220 print "    information-lines) in the ";sm$;"."
9245 gosub 15000:rem pause routine
9295 goto 500:rem return to menu
9298 :
9299 :
9300 rem **--information about option #3--**
9305 scnclr
9310 char 0,4,7,"This option allows you to return to the main program menu."
9315 print:print:
9335 gosub 15000:rem pause routine
9340 goto 500:rem return to menu
9398 :
9399 :
14000 rem *--incorrect choice message--*
14005 char 0,4,23
14010 color chtr,white:print tab(col) "Incorrect Choice! ";
14015 color chtr,yellow:print tab(col) "Press ";
14020 color chtr,white:print "RETURN";
14025 color chtr,yellow:print " to continue:";
14030 gosub 19000:rem getkey subroutine
14035 char 0,4,23
14040 print chr$(27) + chr$(81);:rem erase line
14045 return
14050 :
15000 rem **--pause routine--**
15005 color chtr,purple
15010 char 0,col,22,"Press the "
15015 color chtr,white
15020 print "RETURN";
15025 color chtr,purple
15030 print " key to continue: ";
15035 gosub 19000:rem getkey subroutine
15040 scnclr
15045 return
15050 :
15055 :
15060 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 gosub 14000:return:rem ask again
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
18090 :
19000 rem **--getkey subroutine--**
```

```
19005 getkey a$
19007 if a$ = chr$(27) then 7000
19008 if a$ = chr$(72) then 9000:rem help key
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print chr$(157) " ";:rem cursor left
19052 print chr$(157);
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
19070 :
19075 :
21000 rem **--underline routine--**
21005 for i = 1 to sp
21010 print chr$(2);" ";:rem underline on
21015 next i
21020 print chr$(130):rem underline off
21025 return
```

The homeInv.copy Program

```
  100 rem ***--home inventory copy--***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white = 16
  130 ::::rd =  3
  135 :green =  6
  140 ::blue =  7
  145 yellow = 10
  150 purple =  9
  155 drkyel = 10
  160 ::cyan = 12
  165 ::gray = 16
  170 ::chtr =  5
  175 bckgnd =  0
  195 :
  200 blank$ = chr$(10):rem blank line
  205 print chr$(14):rem upper/lower case mode
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 fast
  245 :
  250 :
  300 rem **--variable list--**
  305 rem item$ = name of item inventoried
  310 rem serl$ = serial # of item
  315 rem  cst$ = cost of item
  320 rem room$ = location of item
  322 rem desc$ = description of item
  325 rem   cv$ = convert to lower case
  330 rem srch$ = search for information
  335 rem find$ = information in data file
  340 rem info$ = associated srch$ information
  345 rem  pztn = position within record
  350 rem  byte = position within record
  355 rem   ptr = pointer for number of records
  360 rem   tp$ = full type message
  365 rem   col = column number
  370 rem   row = row number
  375 rem other variables are descriptive
  380 :
```

```
  385 :
  400 rem **--pointer file--**
  405 :
  410 msg$ = "Insert the diskette that contains the Inventory Record File."
  415 gosub 15000
  420 :
  425 dopen#1,"Inv.Rec.Ptr"
  430 input#1,ptr
  435 dclose#1
  440 :
  445 dim item$(ptr),serl$(ptr),cst$(ptr)
  450 dim room$(ptr),desc$(ptr)
  455 :
  460 :
 5000 rem **--file input routine--**
 5002 gosub 10000:rem message
 5003 print
 5005 :
 5010 rem *--data file--*
 5015 dopen#2,"Inventory Record"
 5020 for k = 1 to ptr
 5022 rec = k:print ".";
 5025 :
 5030 record#2,(rec),1
 5035 input#2,item$(k)
 5040 :
 5045 record#2,(rec),25
 5050 input#2,serl$(k)
 5055 :
 5060 record#2,(rec),40
 5065 input#2, cst$(k)
 5070 :
 5075 record#2,(rec),50
 5080 input#2,room$(k)
 5085 :
 5090 record#2,(rec),70
 5095 input#2,desc$(k)
 5100 :
 5105 next k
 5110 :
 5115 dclose#2
 5117 msg$ ="Insert the diskette that will receive the copy of the file."
 5120 gosub 15000:rem pause routine
 5125 :
 5130 :
 6000 rem **--file output routine--**
 6002 gosub 12000:rem message
 6003 print
 6005 :
 6010 rem *--data file--*
 6015 dopen#2,"Inventory Record",l100
 6017 for k = 1 to ptr
 6018 rec = k:
 6019 record#2,(rec)
 6020 :
 6025 record#2,(rec),1
 6030 print#2,item$(k)
 6035 :
 6040 record#2,(rec),25
 6045 print#2,serl$(k)
 6050 :
 6055 record#2,(rec),40
 6060 print#2,cst$(k)
 6065 :
 6070 record#2,(rec),50
 6075 print#2,room$(k)
 6080 :
 6085 record#2,(rec),70
 6090 print#2,desc$(k)
```

```
 6095 :
 6097 record#2,(rec)
 6100 next k
 6105 dclose#2
 6115 :
 6120 :
 6200 rem *--pointer file--*
 6205 dopen#1,"Inv.Rec.Ptr",w
 6210 print#1,ptr
 6215 dclose#1
 6220 :
 6225 :
 6230 :
 7000 rem **--return to program menu--**
 7002 msg$ = "Insert the Home Inventory System Program Diskette."
 7003 gosub 15000:rem pause
 7005 scnclr
 7010 char 0,15,9
 7015 color chtr,rd
 7020 print "LOADING THE HOMEINV.MENU PROGRAM"
 7025 color chtr,yellow
 7030 run "homeinv.menu"
 7035 :
 7040 :
10000 rem **--reading file message--**
10005 scnclr
10010 color chtr,purple
10015 print chr$(15);:rem flash on
10020 char 0,25,7,"READING FILE INFORMATION. PLEASE WAIT!"
10025 print chr$(143):rem flash off
10030 color chtr,yellow
10035 return
10040 :
10045 :
10050 :
12000 rem **--writing file message--**
12005 scnclr
12010 color chtr,purple
12015 print chr$(15);:rem flash on
12020 char 0,25,7,"WRITING FILE INFORMATION. PLEASE WAIT!"
12025 print chr$(143):rem flash off
12030 color chtr,yellow
12035 return
12040 :
12045 :
12050 :
15000 rem **--pause routine--**
15002 scnclr
15003 color chtr,rd
15004 char 0,13,7,(msg$)
15005 color chtr,purple
15010 char 0,25,11,"Press the "
15015 color chtr,white
15020 print "RETURN";
15025 color chtr,purple
15030 print " key to continue: ";
15035 gosub 19000:rem getkey subroutine
15040 scnclr
15045 return
15050 :
15055 :
15060 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19007 if a$ = chr$(27) then 7000
19008 if a$ = chr$(72) then 9000:rem help key
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
```

```
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print chr$(157) " ";:rem cursor left
19052 print chr$(157);
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
```

The magazine.menu Program

```
  100 rem ***--magazine article system menu--***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white = 16
  130 ::::rd =  3
  135 :green =  6
  140 ::blue =  7
  145 yellow = 10
  150 purple =  9
  155 drkyel = 10
  160 ::cyan = 12
  165 ::gray = 16
  170 ::chtr =  5
  175 bckgnd =  0
  195 :
  200 blank$ = chr$(10):rem blank line
  205 print chr$(14):rem upper/lower case mode
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 fast
  225 ttle$ = "MAGAZINE ARTICLES SYSTEM"
  230 ln = len(ttle$)
  235 center = (80 - ln) / 2:rem center title
  240 key 1,"goto 9000" + chr$(13)
  245 :
  250 :
  500 rem **--magazine articles menu--**
  505 scnclr
  510 color chtr,purple
  515 :
  520 rem *--draw top line--*
  525 for i = 2 to 79
  530 print chr$(2);" ";
  535 next i
  540 print chr$(130)
  545 :
  550 rem *--display choices--*
  555 col = center:row = 2
  560 color chtr,white
  565 char 0,4,row,"Magazines"
  570 char 0,col,row,"MAGAZINE ARTICLES MENU            Select Action"
  575 color chtr,yellow
  580 col =  8:row = 4
  585 char 0,col,row +  2,"1.  WRITE RECORD"
  590 char 0,col,row +  4,"2.  READ RECORD"
  595 char 0,col,row +  6,"3.  SEARCH RECORD"
  600 char 0,col,row +  8,"4.  CORRECT RECORD"
  605 char 0,col,row + 10,"5.  COPY FILE"
  610 col = 50:row = 4
  615 char 0,col,row +  2,"6.  LIST OF FILES"
  620 char 0,col,row +  4,"7.  END SESSION"
  625 col = 4:row = 4
  630 char 0,col,row + 14
  635 print "Type your Selection ";
  640 color chtr,white:print "(1-7)";
```

```
  645 color chtr,yellow:print " and press ";
  650 color chtr,white:print "RETURN";
  655 color chtr,yellow:print ": ---> ";
  660 x = pos(0)
  665 color chtr,purple:char 0,0,row + 15
  670 :
  675 rem *--draw bottom line--*
  680 for i = 2 to 79
  685 print chr$(2);" ";
  690 next i
  695 print chr$(130)
  700 :
  705 rem *--options--*
  710 color chtr,yellow
  715 char 0, 4,row + 17,"Options: "
  720 color chtr,white:print "ESC";:
  725 color chtr,yellow:print " key to leave.  ";
  730 color chtr,white:print "HELP";
  735 color chtr,yellow:print " key for information."
  740 :
  745 rem *--wait for keypress--*
  750 char 0,x,row + 14
  755 gosub 19000:rem getkey routine
  760 number$ = t$:t$ = ""
  765 if number$ = chr$(13) then 750
  770 number = val(number$)
  775 :
  780 if number = 1 then 1000
  785 if number = 2 then 2000
  790 if number = 3 then 3000
  795 if number = 4 then 4000
  800 if number = 5 then 5000
  805 if number = 6 then 6000
  810 if number = 7 then 7000
  820 :
  825 rem *--incorrect choice message--*
  830 gosub 14000:rem incorrect choice
  835 goto 500:rem menu--check again
  840 :
  845 :
 1000 rem **--write record program--**
 1005 file$ = "MAGAZINE.WRITE"
 1010 gosub 17000:rem new program routine
 1015 run "magazine.write"
 1020 :
 1025 :
 2000 rem **--read record program--**
 2005 file$ = "MAGAZINE.READ"
 2010 gosub 17000:rem new program routine
 2015 run "magazine.read"
 2020 :
 2025 :
 3000 rem **--search record program--**
 3005 file$ = "MAGAZINE.SEARCH"
 3010 gosub 17000:rem new program routine
 3015 run "magazine.search"
 3020 :
 3025 :
 4000 rem **--correct record program--**
 4005 file$ = "MAGAZINE.CORRECT"
 4010 gosub 17000:rem new program routine
 4015 run "magazine.correct"
 4020 :
 4025 :
 5000 rem **--copy file program--**
 5005 file$ = "MAGAZINE.COPY"
 5010 gosub 17000:rem new program routine
 5015 run "magazine.copy"
 5020 :
```

```
5025 :
6000 rem **--list of files routine--**
6005 scnclr
6010 directory
6015 print:print
6020 print "Are you ready to return to the menu? ";
6025 gosub 18000:rem y/n input routine
6030 if yes$ = "y" then 500:rem menu
6035 goto 6000:rem check again
6040 :
6045 :
7000 rem **--end routine--**
7005 scnclr
7010 color chtr,grey
7015 char 0,18,9,"That's all for this session! See you next time."
7020 color chtr,drkyel
7025 char 0,0,22
7030 slow
7035 end
7040 :
7045 :
9000 rem **--information section--**
9002 sm$ = "Magazine Article System"
9005 color chtr,cyan
9010 getkey b$,c$,d$
9012 char 0,4,row + 17
9013 print chr$(27)+chr$(81);
9015 input "Which Selection do you need information about";e$
9017 getkey e$
9018 e = val(e$)
9020 color chtr,yellow
9021 if e = 1 then 9100
9022 if e = 2 then 9200
9023 if e = 3 then 9300
9024 if e = 4 then 9400
9025 if e = 5 then 9500
9026 if e = 6 then 9600
9027 if e = 7 then 9700
9028 gosub 14000:rem incorrect choice
9098 :
9099 :
9100 rem **--information about option #1--**
9105 scnclr
9110 char 0,4,7,"This option allows you to enter information about Magazine"
9115 print:print
9120 print "    Articles. You will be asked to provide the article's name, author,"
9125 print
9130 print "    magazine name, date, and page # for each item you wish to track."
9135 print
9140 print "    Each entry allows only a certain amount of space. Please do not"
9145 print
9150 print "    exceed the alloted number of characters."
9155 gosub 15000:rem pause routine
9195 goto 500:rem return to menu
9198 :
9199 :
9200 rem **--information about option #2--**
9205 scnclr
9210 char 0,4,7,"This option displays the contents of the " + sm$
9215 print:print
9220 print"    in a very elementry form. The entire file is read and displayed."
9225 print
9230 print "    You are given the choice of displaying the information either on"
9235 print
9240 print "    the screen or on paper with a printer."
9245 gosub 15000:rem pause routine
9295 goto 500:rem return to menu
9298 :
9299 :
```

```
9300 rem **--information about option #3--**
9305 scnclr
9310 char 0,4,7,"This option is the heart of the " + sm$ + ". You are "
9315 print:print
9320 print "    presented with another menu of seven choices allowing you to"
9325 print
9330 print "    search and/or sort the file information in a number of different"
9335 print
9340 print "    formats. There are four search routines, two sort routines and an"
9345 print
9350 print "    option allowing you to return to this main program menu."
9355 gosub 15000:rem pause routine
9395 goto 500:rem return to menu
9398 :
9399 :
9400 rem **--information about option #4--**
9405 scnclr
9410 char 0,4,7,"This option allows you to correct or delete information in"
9415 print:print
9420 print "    the ";sm$;" file. You may change information as often"
9425 print
9430 print "    as you want. Deleted information is actually removed, not just"
9435 print
9440 print "    flagged. Therefore, be certain that you do not need the"
9445 print
9450 print "    information before deleting it."
9455 gosub 15000:rem pause routine
9495 goto 500:rem return to menu
9498 :
9499 :
9500 rem **--information about option #5--**
9505 scnclr
9510 char 0,4,7,"This option allows you to make a copy of the " + sm$
9515 print:print
9520 print "    file. You may make the copy on the same diskette or on a"
9525 print
9530 print "    different diskette as long as the new diskette has previously"
9535 print
9540 print "    been formatted. The file copy is given the added letters, 'bak'."
9545 gosub 15000:rem pause routine
9595 goto 500:rem return to menu
9598 :
9599 :
9600 rem **--information about option #6--**
9605 scnclr
9610 char 0,4,7,"This option simply displays a directory of the current diskette."
9615 gosub 15000:rem pause routine
9695 goto 500:rem return to menu
9698 :
9699 :
9700 rem **--information about option #7--**
9705 scnclr
9710 char 0,4,7,"This option allows you to stop using the " + sm$ + "."
9715 print:print:
9720 print "    You are returned to BASIC and can then safely remove the diskette"
9725 print
9730 print "    from the disk drive."
9735 gosub 15000:rem pause routine
9795 goto 500:rem return to menu
9798 :
9799 :
14000 rem *--incorrect choice message--*
14005 char 0,4,23
14010 color chtr,white:print tab(col) "Incorrect Choice! ";
14015 color chtr,yellow:print "Press the ";
14020 color chtr,white:print "RETURN";
14025 color chtr,yellow:print " key to continue:";
14030 gosub 19000:rem getkey subroutine
```

```
14035 return
14040 :
14045 :
14050 :
15000 rem **--pause routine--**
15005 color chtr,purple
15010 char 0,col,22,"Press the "
15015 color chtr,white
15020 print "RETURN";
15025 color chtr,purple
15030 print " key to continue: ";
15035 gosub 19000:rem getkey subroutine
15040 scnclr
15045 return
15050 :
15055 :
15060 :
17000 rem **--new program routine--**
17005 scnclr:char 0,15,5
17010 print "You have selected the ";
17015 color chtr,red
17020 print file$;
17025 color chtr,yellow
17030 print " program."
17035 char 0,15,9
17040 print "Is this the program you want? ";
17045 gosub 18000:rem y/n input routine
17050 if yes$ = "n" then 500:rem menu
17055 :
17060 rem *--loading message--*
17065 scnclr
17070 color chtr,purple:char 0,15,5
17075 print "Please wait! I'm loading....";
17080 color chtr,red
17085 print file$
17090 color chtr,purple
17095 return
17100 :
17105 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 print
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
18090 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19007 if a$ = chr$(27) then 7000
19008 if a$ = chr$(72) then 9000:rem help key
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
```

```
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print a$;
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
19070 :
19075 :
```

The magazine.write Program

```
  100 rem ***--write magazines record--***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white = 16
  130 ::::rd =  3
  135 :green =  6
  140 ::blue =  7
  145 yellow = 10
  150 purple =  9
  155 drkyel = 10
  160 ::cyan = 12
  165 ::gray = 16
  170 ::chtr =  5
  175 bckgnd =  0
  195 :
  200 blank$ = chr$(10):rem blank line
  205 print chr$(14):rem upper/lower case mode
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 fast
  225 :
  230 :
  300 rem **--variable list--**
  305 rem  ttl$ = name of article
  310 rem auth$ = author of article
  315 rem magz$ = name of magazine
  320 rem  dte$ = date of article
  322 rem page$ = page # of article
  323 rem code$ = description of article
  325 rem   cv$ = convert to lower case
  330 rem srch$ = search for information
  335 rem find$ = information in data file
  340 rem info$ = associated srch$ information
  345 rem  pztn = position within record
  350 rem  byte = position within record
  355 rem   ptr = pointer for number of records
  365 rem   col = column number
  370 rem   row = row number
  375 rem other variables are descriptive
  380 :
  385 :
 1000 rem **--write record routine--**
 1005 scnclr:col = 10
 1010 color chtr,white
 1015 char 0,25,2,"ENTER INFORMATION"
 1020 color chtr,blue
 1025 char 0,col,5,"Please do not go beyond the end of the line!"
 1030 color chtr,drkyel
 1035 :
 1040 :
 1100 rem *--name of article--*
 1105 print:print:print
 1110 char 0,col,8
 1115 msg$ = "Type in the article's name please: "
 1117 lm = len(msg$)
 1118 print msg$;
```

```
1120 sp = 23
1125 gosub 21000:rem underline routine
1130 char 0,lm + col,8
1135 print chr$(2);
1140 gosub 19000:rem getkey subroutine
1145 print chr$(130)
1150 ttl$ = t$:t$ = ""
1155 if len( ttl$) > 23 then  ttl$ = left$( ttl$,23)
1160 :
1165 :
1200 rem *--author of article--*
1205 char 0,col,10
1210 msg$ = "Type in the article's author please: "
1215 lm = len(msg$)
1220 print msg$;
1225 sp = 14
1230 gosub 21000:rem underline routine
1235 char 0,lm + col,10
1240 print chr$(2);
1245 gosub 19000:rem getkey subroutine
1250 print chr$(130)
1255 auth$ = t$:t$ = ""
1260 if len(auth$) > 14 then auth$ = left$(auth$,14)
1265 :
1270 :
1300 rem *--name of magazine--*
1305 char 0,col,12
1310 msg$ = "Type in the magazine's name please: "
1315 lm = len(msg$)
1320 print msg$;
1325 sp = 19
1330 gosub 21000:rem underline routine
1335 char 0,lm + col,12
1340 print chr$(2);
1345 gosub 19000:rem getkey subroutine
1350 print chr$(130)
1355 magz$ = t$:t$ = ""
1360 if len(magz$) > 19 then magz$ = left$(magz$,19)
1365 :
1370 :
1400 rem *--date of magazine--*
1405 char 0,col,14
1410 msg$ = "Type in the magazine's date please: "
1415 lm = len(msg$)
1420 print msg$;
1425 sp =  9
1430 gosub 21000:rem underline routine
1435 char 0,lm + col,14
1440 print chr$(2);
1445 gosub 19000:rem getkey subroutine
1450 print chr$(130)
1455 dte$ = t$:t$ = ""
1460 if len( dte$) >  9 then  dte$ = left$( dte$, 9)
1465 :
1470 :
1500 rem *--page number of article--*
1505 char 0,col,16
1510 msg$ = "Type in the article's page number please: "
1515 lm = len(msg$)
1520 print msg$;
1525 sp =  4
1530 gosub 21000:rem underline routine
1535 char 0,lm + col,16
1540 print chr$(2);
1545 gosub 19000:rem getkey subroutine
1550 print chr$(130)
1555 page$ = t$:t$ = ""
1560 if len(page$) >  4 then page$ = left$(page$, 4)
1565 :
```

```
1570 :
1600 rem *--category of article--*
1605 char 0,col,18
1610 msg$ = "Type in the article's code/category please: "
1615 lm = len(msg$)
1620 print msg$;
1625 sp =  9
1630 gosub 21000:rem underline routine
1635 char 0,lm + col,18
1640 print chr$(2);
1645 gosub 19000:rem getkey subroutine
1650 print chr$(130)
1655 code$ = t$:t$ = ""
1660 if len(code$) >  9 then code$ = left$(code$, 9)
1665 :
1670 :
1800 rem *--display for correction--*
1810 scnclr
1815 col = 17:row = 2
1820 color chtr,white
1825 char 0,col,row,"ENTERED INFORMATION"
1830 color chtr,drkyel
1835 char 0,col,row +  2,"1.  Title: " +  ttl$
1840 char 0,col,row +  4,"2.  Authr: " + auth$
1845 char 0,col,row +  6,"3.  Magzn: " + magz$
1850 char 0,col,row +  8,"4.  Date : " +  dte$
1855 char 0,col,row + 10,"5.  Page : " + page$
1860 char 0,col,row + 12,"6.  Code : " + code$
1875 char 0,col,row + 14,"Is this correct? "
1880 gosub 18000:rem y/n input routine
1885 if yes$ = "y" then 2000:rem file output routine
1890 :
1895 char 0,col,row + 16,"Which number is wrong? "
1900 gosub 19000:rem getkey subroutine
1905 nb$ = t$:t$ = ""
1910 nb = val(nb$)
1915 if nb < 1 or nb > 6 then gosub 14000:goto 1800
1925 char 0,col,row + 18,"Type in the correct information: "
1930 gosub 19000:rem getkey subroutine
1935 cinfo$ = t$:t$ = ""
1940 if nb = 1 then  ttl$ = left$(cinfo$,23):goto 1800:rem ask again
1945 if nb = 2 then auth$ = left$(cinfo$,14):goto 1800:rem ask again
1950 if nb = 3 then magz$ = left$(cinfo$,19):goto 1800:rem ask again
1955 if nb = 4 then  dte$ = left$(cinfo$, 9):goto 1800:rem ask again
1960 if nb = 5 then page$ = left$(cinfo$, 4):goto 1800:rem ask again
1965 if nb = 6 then code$ = left$(cinfo$, 9):goto 1800:rem ask again
1990 :
1995 :
2000 rem **--file output routine--**
2005 :
2010 rem *--get # of records from pointer file--*
2015 dopen#1,"Mag.Rec.Ptr"
2020 input#1,ptr
2025 dclose#1
2027 if status = 16 then 3000
2030 ptr = ptr + 1
2035 :
2040 rem *--data file--*
2045 dopen#2,"Magazine Record",185
2047 record#2,(ptr)
2050 :
2055 record#2,(ptr),1
2060 print#2,ttl$
2065 :
2070 record#2,(ptr),25
2075 print#2,auth$
2080 :
2085 record#2,(ptr),40
2090 print#2,magz$
```

```
2095 :
2100 record#2,(ptr),60
2105 print#2,dte$
2110 :
2115 record#2,(ptr),70
2120 print#2,page$
2125 :
2126 record#2,(ptr),75
2127 print#2,code$
2128 :
2129 record#2,(ptr)
2130 dclose#2
2135 :
2140 rem *--update pointer file--*
2145 dopen#1,"@Mag.Rec.Ptr",w
2150 print#1,ptr
2155 dclose#1
2160 :
2165 :
2170 goto 4000:rem repeat routine
2175 :
2180 :
3000 rem **--first time use only--**
3005 dclose#1
3010 dopen#1,"Mag.Rec.Ptr",w
3015 print#1,"0"
3020 dclose#1
3025 :
3030 goto 2000:rem begin file routine again
3035 :
3040 :
4000 rem **--repeat routine--**
4005 scnclr
4010 char 0,col,row,"Do you want to add more items? "
4015 gosub 18000:rem y/n input routine
4020 if yes$ = "y" then 1000:rem get more info
4025 if yes$ = "n" then 6000:rem return to main menu
4030 :
4035 :
6000 rem **--return to program menu--**
6005 scnclr
6010 char 0,15,9
6015 color chtr,red
6020 print "LOADING THE MAGAZINE.MENU PROGRAM"
6025 color chtr,yellow
6030 run "magazine.menu"
6035 :
6040 :
14000 rem *--incorrect choice message--*
14005 print:print
14010 color chtr,white:print tab(col) "Incorrect Choice!"
14015 color chtr,yellow:print:print tab(col) "Press ";
14020 color chtr,white:print "RETURN";
14025 color chtr,yellow:print " to continue:";
14030 gosub 19000:rem getkey subroutine
14035 return
14040 :
14045 :
14050 :
15000 rem **--pause routine--**
15005 color chtr,purple
15010 char 0,col,22,"Press the "
15015 color chtr,white
15020 print "RETURN";
15025 color chtr,purple
15030 print " key to continue: ";
15035 gosub 19000:rem getkey subroutine
15040 scnclr
```

```
15045 return
15050 :
15055 :
15060 :
17000 rem **--convert to lowercase--**
17005 nc$ = ""
17010 for cv = 1 to len(cv$)
17015 x = asc(mid$(cv$,cv,1))
17020 if x > 192 then x = x - 128
17025 nc$ = nc$ + chr$(x)
17030 next cv
17035 :
17040 cv$ = nc$
17045 f = asc(left$(cv$,1))
17050 f = f + 128
17055 nc$ = chr$(f) + right$(cv$,len(cv$)- 1)
17060 return
17065 :
17070 rem na$ = originally typed name
17075 rem cv$ = converted to lowercase
17080 rem nc$ = 1st letter/uppercase
17085 :
17090 :
17095 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 gosub 14000:goto 1800
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
18090 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print chr$(157) " ";:rem cursor left
19052 print chr$(157);
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
19070 :
19075 :
21000 rem **--underline routine--**
21005 for i = 1 to sp
21010 print chr$(2);" ";:rem underline on
21015 next i
21020 print chr$(130):rem underline off
21025 return
```

Chapter 11

Planning A File System

Rather than present another chapter explaining the programming of another system, this chapter covers the procedures involved in conceiving and creating a file system. A Stock Market System will be used as the example. Versions of the basic Stock Market System can be used to keep track of individual player statistics (Sports System), information relating to video tapes and video discs (Video System), and personal library information (Library System). One version forms a fairly sophisticated relative-access C-128 Mailing List System. Although shorter than previous chapters, this chapter is no less important.

There are five main steps involved in conceiving and creating a specific database system.

1. Know your subject.
2. Plan carefully and organize your thoughts.
3. Make preliminary decisions on the number of main variables, the length of each record, and if necessary, the lengths of fields within each record.
4. Roughly plan the sequence of the database operation and the code for each part of that operation.
5. Begin on the code for the first part of the system.

Some programmers will argue with either the sequence of the steps or the steps themselves. Some might say that such an outline is too limited. This might be true, but I am merely trying to give a limited amount of guidance in the development of a specific file system. Some systems analysts are carried away with the precode procedures, but one thing is clear: a certain amount of planning before coding is absolutely necessary.

THE STOCK MARKET SYSTEM

Results of the previous day's trading activity are printed in most daily newspapers. Normally, these results include such things as the 52-week high and low stock price, stock symbol, latest dividend, a yield figure, P/E ratio, sales, daily high, low, and closing price, and, possibly, the amount of price change from the previous day. This information is available for active issues on the New York and American Stock Exchanges. Less information is given for Over the Counter or NASDAQ issues, option, commodity, and bond trading. There are figures for the various averages: NYSE Index, Dow, Standard & Poors, American Index, NASDAQ Composite and/or Industrials, to name just a few. There are other key items to watch: gold, interest rates, value of the dollar overseas, money supply, the President's daily intake of vitamins, and so on. As you might guess, the list is limitless.

Although an investment record might contain more variables than a record of information on your own library does, the principle is the same. You must thoroughly know your subject in order to be able to make decisions concerning the information to be saved. The first step in planning your database should be deciding which information is of value (i.e., what information to save).

In this Stock Market System, I am going to severely limit the amount of information saved. You might wish to keep additional information that you believe to be important. For each issue, this system will save the daily high, low, and closing price, plus the day's volume, P/E ratio, and date. In addition, any price that makes either a new high or new low for that issue will be saved. The information to be saved is listed below:

1. P/E ratio
2. Day's volume
3. Day's high
4. Day's low
5. Day's close

Steps two and three of the planning process blend together somewhat at this point. In the planning, decisions are made. Most stock prices are under $1000 per share, so this system will allow a maximum of three places before the decimal point. Prices are usually given in terms of eighths of a dollar (i.e., 3/8 or 1/2).

With a little extra planning and coding, significant diskette space can be saved on each issue. If the extreme right figure is always viewed as the fractional portion of the stock price, then four digits will represent all stock prices up to $999 and 7/8 per share. (This price would be saved on the diskette as 9997.) Saving the high, low, and close each day already means 15 bytes per issue per day (4 bytes for the number, plus one byte for the delimiter, for a total of 5 bytes for each high, low, and closing price).

Item	# of Bytes	Example	Entered As
3. Day's high	5	65 7/8	657
4. Day's low	5	64 1/2 (4/8)	644
5. Day's close	5	65 1/4 (2/8)	652

Volume can be handled in a similar way. Most newspapers indicate the sales volume only in hundreds of shares sold per day. A volume of 2000 shares would be displayed as 20. Since virtually every stock trades under 9,999,999 shares in one day, you can limit

the number of places to six (99999, plus one for the carriage return delimiter). All P/E ratios are under 999 for any issue. This necessitates another four bytes for each issue.

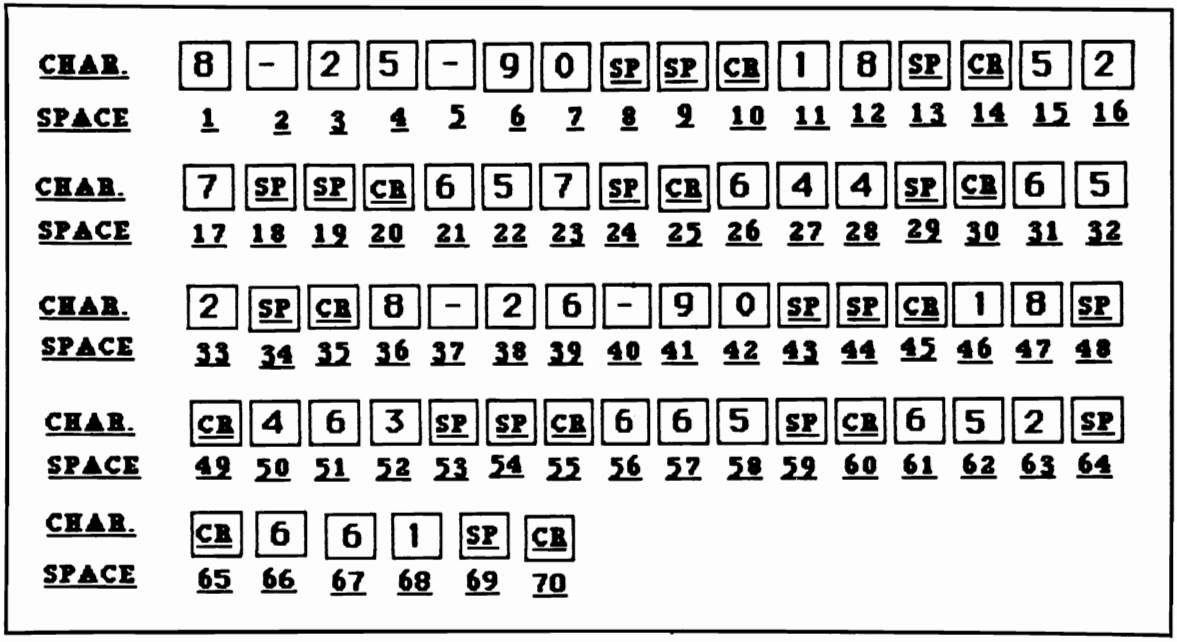
Item	# of Bytes	Example	Entered As
1. P/E ratio	4	18	18
2. Day's volume	6	52,700	527

Finally, the trading date with each record (ten more bytes) will be saved. That brings the total number of bytes for each issue to 35: 4 for the P/E ratio, 6 for the volume, 5 for the high, 5 for the low, 5 for the close, and 10 for the date. Figure 11-1 shows the layout of two records. Any new high or new low price will be saved in a separate sequential file.

Item	# of Bytes	Example	Entered As
1. P/E ratio	4	18	18
2. Day's volume	6	52,700	527
3. Day's high	5	65 7/8	657
4. Day's low	5	64 1/2 (4/8)	644
5. Day's close	5	65 1/4 (2/8)	652
6. *Day's date*	*10*	*July 25, 1990*	*8-25-90*
Total # of Bytes	35		

Next, you must decide on the number of issues to follow on a daily basis. This is an individual choice and often depends on the time available for closely following the mar-

Fig. 11-1. The arrangement of two records in the stock market file.



ket. A reasonable figure to start out with is 10 issues. If a number greater than 10 is used, the string array variable chosen for the stock names will need to be dimensioned to the proper number. With the number of stocks determined, you can calculate the maximum number of bytes used for each day's transactions.

Ten stocks, each requiring 35 bytes, mean a length of 350 bytes per trading day. Based on approximately 340,000 available bytes on a diskette, you can store just under 1000 days of trading information. That is about three years of stock market activity for 10 issues on a single data diskette. If the Stock Market System program is included on the same diskette, the number of trading days decreases.

Next, you need a rough plan of the sequence of programs and the code within each program. Following the procedures used in the previous examples, you need programs that will create the necessary files and daily add to those files. Second, you need programs to display, in a variety of ways, the information either stored in the file or derived from the information stored. Finally, you must have correction programs.

Another method of creating relative files has been included—a program that will only be used once for each set of stocks followed. The stock.menu program indicates that, when the user chooses the first option, the stock.update program is run. The first time (and only the first time) the stock.menu program is run and the number one selection is made, the computer will load a program that is called stock.update. In reality, this is a program used one time to create the stock.ptr file and the hi/low data file. It also provides the user with some information on the general operation of the Stock Market System. When these files have been created and the user has indicated that the information has been absorbed, this program renames itself on the diskette so that it now has the program file name old.update. Then, it renames a second program from its original real.update file name to the necessary stock.update file name. Finally, the user is given the choice to either return to the main Stock Menu or to add the first day's trading activity. In this manner, the stock.menu program always runs a program called stock.update. The first time, it runs one program (originally called stock.update and renamed to old.update). Every time after that, however, it runs another program (originally called real.update and renamed to stock.update). The different programs share the same file name, stock.update, but are assigned the name at different times.

In summary, the first program is designed to create the necessary files and then rename both itself and the real stock.update program. The real stock.update (originally real.update program) is designed to update each individual stock's file daily. The explanation might sound complicated, but the operation itself is surprisingly simple.

Within the file addition program (stock.update), the sequence of operation is fairly standard. You need to read in the value of the file pointer (and add one to that value), the symbols for the various stocks, and their current high and low price. Next, the day's trading figures must be entered for each issue and checked for accuracy. The previous day's trading figures must also be compared to the current high and low, and, if they exceed those figures, they need to replace the appropriate figure. Finally, all the new information must be written to the diskette.

The editing programs should follow a similar pattern, but with a few exceptions. The pointer should be determined by an input from the keyboard. The high, low, close, volume, and P/E should come from the diskette instead of the keyboard, with corrections coming from the keyboard.

The display programs are more difficult to structure in an absolute manner. The only certain structure is that information comes from the diskette and is displayed either on

the screen or through the printer. In between, a variety of steps can take place. The information can be used to evaluate a particular situation and project the price movement of the stock, or the diskette information can simply be formated for display on either the screen or the printer. The information might be used to graph the price movement of the stock, or compare its movement against that of another stock average.

The display portion of the Stock Market System is the core of the system, and is usually not a fixed set of programs. User needs change, and require that the display programs change. In the system presented, the display programs will be limited in their scope. Individual stock histories will be displayed, along with some calculated figures, averages, price and volume changes, and the like. These programs will not get into great detail concerning graphic representation of stock performance in this book.

The final step is the coding of the programs according to the plans established. In Fig. 11-2 at the end of this chapter, you will find minimum programs designed according to the structure outlined in this chapter. You are encouraged to take these minimum programs and expand them to fit your own interests, or alter them to cover a topic of your own design. It is only by practical experience that you will learn to create Commodore 128 BASIC files.

CHAPTER 11 QUESTIONS

1. True or False: Good programmers can just sit down and start writing code.
2. What is the first step in planning a database system?
3. Give the three main parts to any database system.
4. Which part must be flexible as user needs change?

Fig. 11-2. The Stock Market System programs.

The stock.menu Program

```
100 rem ***--stock market menu--***
105 :
110 :
115 rem **--initialization--**
120 :black =  1
125 :white = 16
130 :::red =  3
135 :green =  6
140 ::blue =  7
145 yellow = 10
150 purple =  9
155 drkyel = 10
160 ::cyan = 12
165 ::gray = 16
170 ::chtr =  5
175 bckgnd =  0
195 :
200 blank$ = chr$(10):rem blank line
205 print chr$(14):rem upper/lower case mode
210 print chr$(8):rem disable char.set change
215 color bckgnd,black
220 fast
225 ttle$ = "STOCK MARKET SYSTEM"
230 ln = len(ttle$)
235 center = (80 - ln) / 2:rem center title
240 key 1,"goto 9000" + chr$(13)
245 :
250 :
```

```
500 rem **--stock market menu--**
505 scnclr
510 color chtr,purple
515 :
520 rem *--draw top line--*
525 for i = 2 to 79
530 print chr$(2);" ";
535 next i
540 print chr$(130)
545 :
550 rem *--display choices--*
555 col = center:row = 2
560 color chtr,white
565 char 0,4,row,"Stock Market"
570 char 0,col,row,(ttle$)
572 char 0,60,row,"Select Action"
575 color chtr,yellow
580 col =  4:row = 4
585 char 0,col,row +  2,"1.  Daily Update Stock Market Data"
590 char 0,col,row +  4,"2.  Display Stock Market Information"
595 char 0,col,row +  6,"3.  Display Hi/Low Values"
600 char 0,col,row +  8,"4.  Create/Correct Hi/Low File"
605 col = 50:row = 4
610 char 0,col,row +  2,"5.  Correct Stock Market Data"
615 char 0,col,row +  4,"6.  List Diskette Files"
620 char 0,col,row +  6,"7.  End the Session"
625 col = 4:row = 4
630 char 0,col,row + 14
635 print "Type your Selection ";
640 color chtr,white:print "(1-7)";
645 color chtr,yellow:print " and press ";
650 color chtr,white:print "RETURN";
655 color chtr,yellow:print ": ---> ";
660 x = pos(0)
665 color chtr,purple:char 0,0,row + 15
670 :
675 rem *--draw bottom line--*
680 for i = 2 to 79
685 print chr$(2);" ";
690 next i
695 print chr$(130)
700 :
705 rem *--options--*
710 color chtr,yellow
715 char 0, 4,row + 17,"Options: "
720 color chtr,white:print "ESC";:
725 color chtr,yellow:print " key to leave.  ";
730 color chtr,white:print "HELP";
735 color chtr,yellow:print " key for information."
740 :
745 rem *--wait for keypress--*
750 char 0,x,row + 14
755 gosub 19000:rem getkey routine
760 number$ = t$:t$ = ""
765 if number$ = chr$(13) then 750
770 number = val(number$)
775 :
780 if number = 1 then 1000
785 if number = 2 then 2000
790 if number = 3 then 3000
795 if number = 4 then 4000
800 if number = 5 then 5000
805 if number = 6 then 6000
810 if number = 7 then 7000
820 :
825 rem *--incorrect choice message--*
830 gosub 14000:rem incorrect choice
835 goto 500:rem menu--check again
```

```
840 :
845 :
1000 rem **--stock update program--**
1005 file$ = "STOCK.UPDATE"
1010 gosub 17000:rem new program routine
1015 run "stock.update"
1020 :
1025 :
2000 rem **--stock display program--**
2005 file$ = "STOCK.DISPLAY"
2010 gosub 17000:rem new program routine
2015 run "stock.display"
2020 :
2025 :
3000 rem **--display hi/low program--**
3005 file$ = "STOCK.DSP HI/LOW"
3010 gosub 17000:rem new program routine
3015 run "stock.dsp hi/low"
3020 :
3025 :
4000 rem **--correct hi/low program--**
4005 file$ = "STOCK.CRT HI/LOW"
4010 gosub 17000:rem new program routine
4015 run "stock.crt hi/low"
4020 :
4025 :
5000 rem **--correct stock data--**
5005 file$ = "STOCK.CORRECT"
5010 gosub 17000:rem new program routine
5015 run "stock.correct"
5020 :
5025 :
6000 rem **--list of files routine--**
6005 scnclr
6010 directory
6015 print:print
6020 print "Are you ready to return to the menu? ";
6025 gosub 18000:rem y/n input routine
6030 if yes$ = "y" then 500:rem menu
6035 goto 6000:rem check again
6040 :
6045 :
7000 rem **--end routine--**
7005 scnclr
7010 color chtr,grey
7015 char 0,18,9,"That's all for this session! See you next time."
7020 color chtr,drkyel
7025 char 0,0,22
7030 slow
7035 end
7040 :
7045 :
9000 rem **--information section--**
9005 color chtr,cyan
9010 getkey b$,c$,d$
9012 char 0,4,row + 17
9013 print chr$(27)+chr$(81);
9015 input "Which Selection do you need information about";e$
9017 getkey e$
9018 e = val(e$)
9020 color chtr,yellow
9021 if e = 1 then 9100
9022 if e = 2 then 9200
9023 if e = 3 then 9300
9024 if e = 4 then 9400
9025 if e = 5 then 9500
9026 if e = 6 then 9600
9027 if e = 7 then 9700
```

```
9028 gosub 14000:rem incorrect choice
9098 :
9099 :
9100 rem **--information about option #1--**
9105 scnclr
9110 char 0,4,7,"This option allows you to daily update the Stock Market System"
9115 print:print
9120 print "    Data File. You will be asked to provide the P/E ratio, Volume, and"
9125 print
9130 print "    daily high, low, and closing price for each stock you are tracking."
9135 print
9140 print "    Each entry allows only a certain amount of space. Please do not"
9145 print
9150 print "    exceed the alloted number of characters."
9155 gosub 15000:rem pause routine
9195 goto 500:rem return to menu
9198 :
9199 :
9200 rem **--information about option #2--**
9205 scnclr
9210 char 0,4,7,"This option displays the contents of the Stock Market file."
9215 print:print
9220 print"    The entire file is read and displayed along with some calculations."
9225 print
9230 print "    You are given the choice of displaying the information either on"
9235 print
9240 print "    the screen or on paper with a printer."
9245 gosub 15000:rem pause routine
9295 goto 500:rem return to menu
9298 :
9299 :
9300 rem **--information about option #3--**
9305 scnclr
9310 char 0,4,7,"This option displays the current High and Low price for each"
9315 print:print
9320 print "    stock over the last twelve months."
9355 gosub 15000:rem pause routine
9395 goto 500:rem return to menu
9398 :
9399 :
9400 rem **--information about option #4--**
9405 scnclr
9410 char 0,4,7,"This option allows you to correct or change information in"
9415 print:print
9420 print "    the Stock Market High/Low file. You may change information"
9425 print
9430 print "    as often as you want."
9435 print
9455 gosub 15000:rem pause routine
9495 goto 500:rem return to menu
9498 :
9499 :
9500 rem **--information about option #5--**
9505 scnclr
9510 char 0,4,7,"This option allows you to correct or change information in"
9515 print:print
9520 print "    the Stock Market Data file. You may change information as often"
9525 print
9530 print "    as you want. Deleted information is actually removed, not just"
9535 print
9540 print "    flagged. Therefore, be certain that you do not need the"
9545 print
9550 print "    information before deleting it."
9590 gosub 15000:rem pause routine
9595 goto 500:rem return to menu
9598 :
9599 :
9600 rem **--information about option #6--**
```

```
9605 scnclr
9610 char 0,4,7,"This option simply displays a directory of the current diskette."
9615 gosub 15000:rem pause routine
9695 goto 500:rem return to menu
9698 :
9699 :
9700 rem **--information about option #7--**
9705 scnclr
9710 char 0,4,7,"This option allows you to stop using the Home Inventory System."
9715 print:print:
9720 print "    You are returned to BASIC and can then safely remove the diskette"
9725 print
9730 print "    from the disk drive."
9735 gosub 15000:rem pause routine
9795 goto 500:rem return to menu
9798 :
9799 :
14000 rem *--incorrect choice message--*
14005 char 0,4,23
14010 color chtr,white:print tab(col) "Incorrect Choice! ";
14015 color chtr,yellow:print "Press the ";
14020 color chtr,white:print "RETURN";
14025 color chtr,yellow:print " key to continue:";
14030 gosub 19000:rem getkey subroutine
14035 return
14040 :
14045 :
14050 :
15000 rem **--pause routine--**
15005 color chtr,purple
15010 char 0,col,22,"Press the "
15015 color chtr,white
15020 print "RETURN";
15025 color chtr,purple
15030 print " key to continue: ";
15035 gosub 19000:rem getkey subroutine
15040 scnclr
15045 return
15050 :
15055 :
15060 :
17000 rem **--new program routine--**
17005 scnclr:char 0,15,5
17010 print "You have selected the ";
17015 color chtr,red
17020 print file$;
17025 color chtr,yellow
17030 print " program."
17035 char 0,15,9
17040 print "Is this the program you want? ";
17045 gosub 18000:rem y/n input routine
17050 if yes$ = "n" then 500:rem menu
17055 :
17060 rem *--loading message--*
17065 scnclr
17070 color chtr,purple:char 0,15,5
17075 print "Please wait! I'm loading....";
17080 color chtr,red
17085 print file$
17090 color chtr,purple
17095 return
17100 :
17105 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
```

```
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 print
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
18090 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19007 if a$ = chr$(27) then 7000
19008 if a$ = chr$(72) then 9000:rem help key
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print a$;
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
```

The Original stock.update Program

```
  100 rem ***--initialize stock market files-***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white = 16
  130 ::::rd =  3
  135 :green =  6
  140 ::blue =  7
  145 yellow = 10
  150 purple =  9
  155 drkyel = 10
  160 ::cyan = 12
  165 ::gray = 16
  170 ::chtr =  5
  175 bckgnd =  0
  195 :
  200 blank$ = chr$(10):rem blank line
  205 print chr$(14):rem upper/lower case mode
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 fast
  225 :
  230 :
  500 rem **--information--**
  505 scnclr
  510 color chtr,yellow
  515 print "Before you can enter any daily stock information, you must decide"
  520 print
  525 print "which stocks you will be following. This program will ask you to"
  530 print
  535 print "enter the name or symbol of ten stocks and their respective values"
  540 print
  545 print "for the high and low prices within the last 12 months. If the high"
  550 print
  555 print "or low values are not known, simply enter a '0'. The high and low"
  560 print
  565 print "values will automatically be updated when daily prices exceed or"
```

```
 570 print
 575 print "are less than the values in the 'Hi/Low' file. If you need to change"
 580 print
 585 print "any of the information in the Hi/Low file, choose #4 Correct Hi/Low"
 590 print
 595 print "from the main program menu. Once you have finished entering the"
 600 print
 605 print "information in this program, you will be given the choice to enter"
 610 print
 615 print "the first day's prices or to return to the main program menu."
 620 print
 625 gosub 15000:rem pause
 630 scnclr
 635 print "After this first time, the Update Stock Information selection will not"
 640 print
 645 print "show all this text or require you to update the Hi/Low file."
 650 print
 655 print "If you want to re-read this information, type a 'Y', if not, type"
 660 print
 665 print "an 'N'. ";
 670 gosub 18000:rem y/n input routine
 675 if yes$ = "y" then 500:rem begin again
1000 rem **--write record routine--**
1002 for k = 1 to 10
1005 scnclr:col = 10
1010 color chtr,white
1015 char 0,25,2,"ENTER INFORMATION"
1020 color chtr,blue
1025 char 0,col,5,"Please do not go beyond the end of the line!"
1030 color chtr,drkyel
1035 :
1040 :
1100 rem *--name of item--*
1105 k$ = str$(k)
1107 k$ = str$(k)
1110 char 0,col,8
1115 msg$ = "Type in Stock #" + k$ + "'s NAME or SYMBOL please: "
1117 lm = len(msg$)
1118 print msg$;
1120 sp = 16
1125 gosub 21000:rem underline routine
1130 char 0,lm + col,8
1135 print chr$(2);
1140 gosub 19000:rem getkey subroutine
1145 print chr$(130)
1150 stk$(k) = t$:t$ = ""
1155 if len( stk$(k)) > 16 then  stk$(k) = left$( stk$(k),16)
1160 :
1165 :
1200 rem *--stock's 12 mo. high price--*
1202 gosub 16000:rem reminder routine
1203 color chtr,cyan
1204 char 0,col,20,"If you are not sure, enter a '0'."
1205 color chtr,yellow
1207 char 0,col,22
1210 msg$ = "Type in Stock #" + k$ + "'s 12 month HIGH price please: "
1215 lm = len(msg$)
1220 print msg$;
1225 sp =  5
1230 gosub 21000:rem underline routine
1235 char 0,lm + col,22
1240 print chr$(2);
1245 gosub 19000:rem getkey subroutine
1250 print chr$(130)
1255 hi$(k) = t$:t$ = ""
1260 if len(  hi$(k)) >  5 then   hi$(k) = left$(  hi$(k), 5)
1265 :
1270 :
1300 rem *--stock's 12 mo. low price--*
```

```
1302 gosub 16000:rem reminder routine
1303 color chtr,cyan
1304 char 0,col,20,"If you are not sure, enter a '0'."
1305 color chtr,yellow
1307 char 0,col,22
1310 msg$ = "Type in Stock #" + k$ + "'s 12 month LOW price please: "
1315 lm = len(msg$)
1320 print msg$;
1325 sp =  5
1330 gosub 21000:rem underline routine
1335 char 0,lm + col,22
1340 print chr$(2);
1345 gosub 19000:rem getkey subroutine
1350 print chr$(130)
1355 low$(k) = t$:t$ = ""
1360 if len( low$(k)) >  5 then  low$(k) = left$( low$(k), 5)
1365 :
1370 :
1800 rem *--display for correction--*
1810 scnclr
1815 col = 17:row = 2
1820 color chtr,white
1825 char 0,col,row,"ENTERED INFORMATION"
1830 color chtr,drkyel
1835 char 0,col,row +  2,"1.  Name       :  " + stk$(k)
1837 m = val(hi$(k)):gosub 27000:rem convert to fraction
1840 char 0,col,row +  4,"2.  12 Mo. High: " + str$(m) + " " + f$
1842 m = val(low$(k)):gosub 27000:rem convert to fraction
1845 char 0,col,row +  6,"3.  12 Mo. Low : " + str$(m) + " " + f$
1875 char 0,col,row +  8,"Is this correct? "
1880 gosub 18000:rem y/n input routine
1885 if yes$ = "y" then 1975:rem next stock data
1890 :
1895 char 0,col,row + 10,"Which number is wrong? "
1900 gosub 19000:rem getkey subroutine
1905 nb$ = t$:t$ = ""
1910 nb = val(nb$)
1915 if nb < 1 or nb > 3 then gosub 14000:goto 1895
1917 if nb = 1 then cur$ = stk$(k)
1920 if nb = 2 then gosub 16000:row = 10:cur$ = hi$(k)
1922 if nb = 3 then gosub 16000:row = 10:cur$ = low$(k)
1923 char 0,col,row + 10,"Current information is: " + cur$
1925 char 0,col,row + 12,"Type in the correct information: "
1930 gosub 19000:rem getkey subroutine
1935 cinfo$ = t$:t$ = ""
1940 if nb = 1 then stk$(k) = left$(cinfo$,16):goto 1800:rem ask again
1945 if nb = 2 then  hi$(k) = left$(cinfo$, 5):goto 1800:rem ask again
1950 if nb = 3 then low$(k) = left$(cinfo$, 5):goto 1800:rem ask again
1965 :
1970 :
1975 next k
1990 :
1995 :
2000 rem **--file output routine--**
2005 :
2010 rem *--stock market pointer file--*
2015 dopen#1,"Stock.Ptr",w
2020 print#1,"0"
2025 dclose#1
2035 :
2040 rem *--hi/low data file--*
2045 dopen#2,"Hi/Low Data",w
2050 :
2055 for k = 1 to 10
2060 print#2,stk$(k)
2065 print#2,hi$(k)
2070 print#2,low$(k)
2075 next k
2080 :
```

```
2085 dclose#2
2090 :
2095 :
5000 rem **--rename files routine--**
5005 rename "stock.update" to "old.update"
5010 rename "real.update" to "stock.update"
5015 scnclr
5020 char 0,0,5,"Do you want to add the first day's stock prices? "
5025 gosub 18000:rem y/n input routine
5030 if yes$ = "y" then run "stock.update"
5035 :
5040 :
6000 rem **--return to program menu--**
6005 scnclr
6010 char 0,15,9
6015 color chtr,rd
6020 print "LOADING THE STOCK MARKET.MENU PROGRAM"
6025 color chtr,yellow
6030 run "stock.menu"
6035 :
6040 :
14000 rem *--incorrect choice message--*
14005 print:print
14010 color chtr,white:print tab(col) "Incorrect Choice!"
14015 color chtr,yellow:print:print tab(col) "Press ";
14020 color chtr,white:print "RETURN";
14025 color chtr,yellow:print " to continue:";
14030 gosub 19000:rem getkey subroutine
14035 return
14040 :
14045 :
14050 :
15000 rem **--pause routine--**
15005 color chtr,purple
15010 char 0,col,24,"Press the "
15015 color chtr,white
15020 print "RETURN";
15025 color chtr,purple
15030 print " key to continue: ";
15035 gosub 19000:rem getkey subroutine
15040 color chtr,yellow
15045 return
15050 :
15055 :
15060 :
16000 rem **--reminder routine--**
16005 scnclr:col = 12
16010 print tab(col + 8)
16015 color chtr,white
16030 print "*****---REMEMBER---*****"
16035 color chtr,yellow
16040 print
16050 print tab(col) "You must add the fraction as the last"
16060 print tab(col) "digit! 21 5/8 should be entered as: 215"
16070 print
16080 print tab(col) "If the number has no fraction, please"
16090 print tab(col) "enter a '0' after the number! 21 even"
16100 print tab(col) "should be entered as: 210"
16110 print:print
16115 color chtr,rd
16120 print tab(col + 11) "1/8 --------- = 1"
16130 print tab(col + 11) "1/4 --------- = 2"
16140 print tab(col + 11) "3/8 --------- = 3"
16150 print tab(col + 11) "1/2 --------- = 4"
16160 print tab(col + 11) "5/8 --------- = 5"
16170 print tab(col + 11) "3/4 --------- = 6"
16180 print tab(col + 11) "7/8 --------- = 7"
16190 print tab(col + 11) "EVEN -------- = 0"
```

```
16195 color chtr,yellow
16200 print
16210 return
16215 :
16220 :
16225 :
17000 rem **--convert to lowercase--**
17005 nc$ = ""
17010 for cv = 1 to len(cv$)
17015 x = asc(mid$(cv$,cv,1))
17020 if x > 192 then x = x - 128
17025 nc$ = nc$ + chr$(x)
17030 next cv
17035 :
17040 cv$ = nc$
17045 f = asc(left$(cv$,1))
17050 f = f + 128
17055 nc$ = chr$(f) + right$(cv$,len(cv$)- 1)
17060 return
17065 :
17070 rem na$ = originally typed name
17075 rem cv$ = converted to lowercase
17080 rem nc$ = 1st letter/uppercase
17085 :
17090 :
17095 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 print
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
18090 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print chr$(157) " ";:rem cursor left
19052 print chr$(157);
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
19070 :
19075 :
21000 rem **--underline routine--**
21005 for i = 1 to sp
21010 print chr$(2);" ";:rem underline on
21015 next i
21020 print chr$(130):rem underline off
21025 return
```

```
21030 :
21035 :
21040 :
27000 rem **--convert to fraction--**
27010 f = m - int(m/10) * 10
27020 m = int(m/10)
27030 if f = 0 then f$ = ""
27040 if f = 1 then f$ = "1/8"
27050 if f = 2 then f$ = "1/4"
27060 if f = 3 then f$ = "3/8"
27070 if f = 4 then f$ = "1/2"
27080 if f = 5 then f$ = "5/8"
27090 if f = 6 then f$ = "3/4"
27100 if f = 7 then f$ = "7/8"
27110 return
```

The real.update Program

```
  100 rem ***--update stock market files-***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white = 16
  130 ::::rd =  3
  135 :green =  6
  140 ::blue =  7
  145 yellow = 10
  150 purple =  9
  155 drkyel = 10
  160 ::cyan = 12
  165 ::gray = 16
  170 ::chtr =  5
  175 bckgnd =  0
  195 :
  200 blank$ = chr$(10):rem blank line
  205 print chr$(14):rem upper/lower case mode
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 fast
  225 :
  230 :
  400 rem **--input pointer file--**
  405 dopen#1,"Stock.Ptr"
  410 input#1,ptr
  415 dclose#1
  420 ptr = ptr + 1
  425 :
  430 :
  500 rem **--input hi/low file--**
  502 dopen#2,"Hi/Low Data"
  505 for k = 1 to 10
  510 input#2,stk$(k)
  515 input#2,hi$(k)
  520 input#2,low$(k)
  525 next k
  530 dclose#2
  535 :
  540 :
  600 rem **--date information--**
  605 scnclr:col = 10:row = 7:color chtr,yellow
  610 msg$ = "Type in the date of the stock information please: "
  615 lm = len(msg$)
  620 char 0,col,row
  625 print msg$;
  630 sp = 10
  635 gosub 21000:rem underline routine
  640 char 0,lm + col,row
  645 print chr$(2);
  650 gosub 19000:rem getkey subroutine
```

```
 655 print chr$(130)
 660 dte$ = t$:t$ = ""
 662 scnclr
 665 char 0,col,9,"The date is " + dte$
 670 char 0,col,11,"Is this correct? "
 675 gosub 18000:rem y/n input routine
 680 if yes$ = "y" then 1000
 685 msg$ = "Type in the correct date: "
 690 row = 13
 695 goto 615
 700 :
 705 :
1000 rem **--stock information--**
1002 for k = 1 to 10
1005 scnclr:col = 10
1010 color chtr,white
1015 char 0,25,2,"ENTER INFORMATION"
1020 color chtr,blue
1025 char 0,col,5,"Please do not go beyond the end of the line!"
1030 color chtr,drkyel
1035 :
1040 :
1100 rem *--stock's p/e ratio--*
1105 k$ = str$(k)
1110 char 0,col,8
1115 msg$ = "Type in " + stk$(k) +  "'s P/E RATIO for " + dte$ + " please: "
1117 lm = len(msg$)
1118 print msg$;
1120 sp =  3
1125 gosub 21000:rem underline routine
1130 char 0,lm + col,8
1135 print chr$(2);
1140 gosub 19000:rem getkey subroutine
1145 print chr$(130)
1150 pe = val(t$):t$ = ""
1160 :
1165 :
1200 rem *--stock's volume--*
1210 char 0,col,10
1215 msg$ = "Type in " + stk$(k) +  "'s VOLUME for " + dte$ + " please: "
1217 lm = len(msg$)
1218 print msg$;
1220 sp =  5
1225 gosub 21000:rem underline routine
1230 char 0,lm + col,10
1235 print chr$(2);
1240 gosub 19000:rem getkey subroutine
1245 print chr$(130)
1250 v = val(t$):t$ = ""
1260 :
1265 :
1300 rem *--stock's daily high price--*
1302 gosub 16000:rem reminder routine
1305 char 0,col,22
1310 msg$ = "Type in " + stk$(k) +  "'s HIGH price for " + dte$ + " please: "
1315 lm = len(msg$)
1320 print msg$;
1325 sp =  4
1330 gosub 21000:rem underline routine
1335 char 0,lm + col,22
1340 print chr$(2);
1345 gosub 19000:rem getkey subroutine
1350 print chr$(130)
1355 h = val(t$):t$ = ""
1365 :
1370 :
1400 rem *--stock's daily low price--*
1402 gosub 16000:rem reminder routine
1405 char 0,col,22
```

```
1410 msg$ = "Type in " + stk$(k) +  "'s LOW price for " + dte$ + " please: "
1415 lm = len(msg$)
1420 print msg$;
1425 sp =  4
1430 gosub 21000:rem underline routine
1435 char 0,lm + col,22
1440 print chr$(2);
1445 gosub 19000:rem getkey subroutine
1450 print chr$(130)
1455 l = val(t$):t$ = ""
1465 :
1470 :
1500 rem *--stock's daily close price--*
1502 gosub 16000:rem reminder routine
1505 char 0,col,22
1510 msg$ = "Type in " + stk$(k) +  "'s CLOSE price for " + dte$ + " please: "
1515 lm = len(msg$)
1520 print msg$;
1525 sp =  4
1530 gosub 21000:rem underline routine
1535 char 0,lm + col,22
1540 print chr$(2);
1545 gosub 19000:rem getkey subroutine
1550 print chr$(130)
1555 c = val(t$):t$ = ""
1565 :
1570 :
1800 rem *--display for correction--*
1810 scnclr
1815 col = 20:row = 1
1820 color chtr,white
1825 char 0,col,row,(stk$(k)) + "'s data for " + dte$
1830 color chtr,drkyel
1832 row = 2
1835 char 0,col,row +  2,"1.  P/E ratio ---- :  " + str$(pe)
1836 char 0,col,row +  4,"2.  Volume ------- :  " + str$( v)
1837 m = h:gosub 27000:rem convert to fraction
1840 char 0,col,row +  6,"3.  High --------- :  " + str$(m) + " " + f$
1842 m = l:gosub 27000:rem convert to fraction
1845 char 0,col,row +  8,"4.  Low ---------- :  " + str$(m) + " " + f$
1850 m = c:gosub 27000:rem convert to fraction
1855 char 0,col,row + 10,"5.  Close -------- :  " + str$(m) + " " + f$
1875 char 0,col,row + 12,"Is this correct? "
1880 gosub 18000:rem y/n input routine
1885 if yes$ = "y" then 2000:rem exchange hi/low routine
1890 :
1895 char 0,col,row + 14,"Which number is wrong? "
1900 gosub 19000:rem getkey subroutine
1905 nb$ = t$:t$ = ""
1910 nb = val(nb$)
1915 if nb < 1 or nb > 5 then gosub 14000:goto 1895
1917 if nb = 1 then cur = pe
1918 if nb = 2 then cur = v
1919 if nb = 3 then gosub 16000:row =  6:cur = h
1920 if nb = 4 then gosub 16000:row =  6:cur = l
1921 if nb = 5 then gosub 16000:row =  6:cur = c
1922 cur$ = str$(cur)
1923 char 0,col,row + 14,"Current information is: " + cur$
1925 char 0,col,row + 16,"Type in the correct information: "
1930 gosub 19000:rem getkey subroutine
1935 cinfo$ = t$:t$ = ""
1940 if nb = 1 then  pe = val(cinfo$):goto 1800:rem ask again
1945 if nb = 2 then   v = val(cinfo$):goto 1800:rem ask again
1950 if nb = 3 then   h = val(cinfo$):goto 1800:rem ask again
1955 if nb = 4 then   l = val(cinfo$):goto 1800:rem ask again
1960 if nb = 5 then   c = val(cinfo$):goto 1800:rem ask again
1965 :
1970 :
1990 :
```

```
1995 :
2000 rem **--exchange hi/low--**
2010 if h > val(hi$(k)) then hi$(k) = str$(h):nh$ = "*"
2020 if l < val(low$(k)) then low$(k) = str$(l):nl$ = "*"
2030 if val(low$(k)) = 0 then low$(k) = str$(l)
2040 :
2050 :
2500 rem **--convert to strings--**
2505 ss$ = str$(pe):gosub 20000: pe$ = ss$
2510 ss$ = str$(v ):gosub 20000:  v$ = ss$
2515 ss$ = str$(h ):gosub 20000:  h$ = ss$
2520 ss$ = str$(l ):gosub 20000:  l$ = ss$
2525 ss$ = str$(c ):gosub 20000:  c$ = ss$
2530 :
2535 :
3000 rem **--data file output routine--**
3005 scnclr:col = 31:row = 7
3010 color chtr,white
3015 char 0,col,row,"ONE MOMENT PLEASE!"
3020 color chtr,yellow
3025 char 0,col,row + 2,"Updating " + stk$(k) + "'s file."
3027 :
3030 cv$ = stk$(k):gosub 17000:stk$ = cv$
3035 :
3040 rem *--data file--*
3042 stk$ = stk$ + ".data"
3045 dopen#3,(stk$),l35
3047 record#3,(ptr)
3050 :
3055 record#3,(ptr),1
3060 print#3,dte$
3065 :
3070 record#3,(ptr),11
3075 print#3,pe$
3080 :
3085 record#3,(ptr),15
3090 print#3,v$
3095 :
3100 record#3,(ptr),21
3105 print#3,h$
3110 :
3115 record#3,(ptr),26
3120 print#3,l$
3125 :
3130 record#3,(ptr),31
3135 print#3,c$
3140 :
3142 record#3,(ptr)
3145 dclose#3
3150 :
3155 nh$ = "":nl$ = ""
3160 :
3165 :
3170 next k:rem repeat routine
3175 :
3180 :
5000 rem **--pointer & hi/low file output routine--**
5005 :
5010 rem *--stock market pointer file--*
5015 dopen#1,"@Stock.Ptr",w
5020 print#1,ptr
5025 dclose#1
5035 :
5040 rem *--hi/low data file--*
5045 dopen#2,"@Hi/Low Data",w
5050 :
5055 for k = 1 to 10
5060 print#2,stk$(k)
5065 print#2,hi$(k)
```

```
 5070 print#2,low$(k)
 5075 next k
 5080 :
 5085 dclose#2
 5090 :
 5095 :
 6000 rem **--return to program menu--**
 6005 scnclr
 6010 char 0,15,9
 6015 color chtr,rd
 6020 print "LOADING THE STOCK MARKET.MENU PROGRAM"
 6025 color chtr,yellow
 6030 run "stock.menu"
 6035 :
 6040 :
14000 rem *--incorrect choice message--*
14005 print:print
14010 color chtr,white:print tab(col) "Incorrect Choice!"
14015 color chtr,yellow:print:print tab(col) "Press ";
14020 color chtr,white:print "RETURN";
14025 color chtr,yellow:print " to continue:";
14030 gosub 19000:rem getkey subroutine
14035 return
14040 :
14045 :
14050 :
15000 rem **--pause routine--**
15005 color chtr,purple
15010 char 0,col,24,"Press the "
15015 color chtr,white
15020 print "RETURN";
15025 color chtr,purple
15030 print " key to continue: ";
15035 gosub 19000:rem getkey subroutine
15040 color chtr,yellow
15045 return
15050 :
15055 :
15060 :
16000 rem **--reminder routine--**
16005 scnclr:col = 12
16010 print tab(col + 8)
16015 color chtr,white
16030 print "*****---REMEMBER---*****"
16035 color chtr,yellow
16040 print
16050 print tab(col) "You must add the fraction as the last"
16060 print tab(col) "digit! 21 5/8 should be entered as: 215"
16070 print
16080 print tab(col) "If the number has no fraction, please"
16090 print tab(col) "enter a '0' after the number! 21 even"
16100 print tab(col) "should be entered as: 210"
16110 print:print
16115 color chtr,rd
16120 print tab(col + 11) "1/8 --------- = 1"
16130 print tab(col + 11) "1/4 --------- = 2"
16140 print tab(col + 11) "3/8 --------- = 3"
16150 print tab(col + 11) "1/2 --------- = 4"
16160 print tab(col + 11) "5/8 --------- = 5"
16170 print tab(col + 11) "3/4 --------- = 6"
16180 print tab(col + 11) "7/8 --------- = 7"
16190 print tab(col + 11) "EVEN -------- = 0"
16195 color chtr,yellow
16200 print
16210 return
16215 :
16220 :
16225 :
```

```
17000 rem **--convert to lowercase--**
17005 nc$ = ""
17010 for cv = 1 to len(cv$)
17015 x = asc(mid$(cv$,cv,1))
17020 if x > 192 then x = x - 128
17025 nc$ = nc$ + chr$(x)
17030 next cv
17035 :
17040 cv$ = nc$
17045 f = asc(left$(cv$,1))
17050 f = f + 128
17055 nc$ = chr$(f) + right$(cv$,len(cv$)- 1)
17060 return
17065 :
17070 rem na$ = originally typed name
17075 rem cv$ = converted to lowercase
17080 rem nc$ = 1st letter/uppercase
17085 :
17090 :
17095 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 print
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
18090 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print chr$(157) " ";:rem cursor left
19052 print chr$(157);
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
19070 :
19075 :
20000 rem **--strip sign routine--**
20005 lgth = len(ss$)
20010 s2$  = right$(ss$,lgth - 1)
20015 ss$  = s2$
20020 return
20025 :
20030 :
20035 :
21000 rem **--underline routine--**
21005 for i = 1 to sp
21010 print chr$(2);" ";:rem underline on
21015 next i
```

```
21020 print chr$(130):rem underline off
21025 return
21030 :
21035 :
21040 :
27000 rem **--convert to fraction--**
27010 f = m - int(m/10) * 10
27020 m = int(m/10)
27030 if f = 0 then f$ = ""
27040 if f = 1 then f$ = "1/8"
27050 if f = 2 then f$ = "1/4"
27060 if f = 3 then f$ = "3/8"
27070 if f = 4 then f$ = "1/2"
27080 if f = 5 then f$ = "5/8"
27090 if f = 6 then f$ = "3/4"
27100 if f = 7 then f$ = "7/8"
27110 return
```

The stock.crt hi/low Program

```
  100 rem ***--correct hi/low data--***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white = 16
  130 ::::rd =  3
  135 :green =  6
  140 ::blue =  7
  145 yellow = 10
  150 purple =  9
  155 drkyel = 10
  160 ::cyan = 12
  165 ::gray = 16
  170 ::chtr =  5
  175 bckgnd =  0
  195 :
  200 blank$ = chr$(10):rem blank line
  205 print chr$(14):rem upper/lower case mode
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 fast
  225 flag$ = "off":rem info unchanged
  230 :
  235 :
  400 rem **--file input routine--**
  405 :
  410 dopen#1,"Hi/Low Data"
  415 for k = 1 to 10
  420 input#1,stk$(k)
  425 input#1,hi$(k)
  430 input#1,low$(k)
  435 next k
  440 dclose#1
  445 :
  450 :
  500 rem **--hi/low correct menu--**
  505 col = 24:row = 2
  510 scnclr
  515 color chtr,white
  520 char 0,col,row,"CORRECT HI/LOW INFO.MENU":print
  525 color chtr,yellow
  530 char 0,col,row +  2,"1.  Display Hi/Low Information"
  535 char 0,col,row +  4,"2.  Correct Hi/Low Information"
  540 char 0,col,row +  6,"3.  Return to Main Stock Menu"
  560 char 0,col,row +  8
  565 input "Which Number Please ";number$
  570 number = val(number$)
  575 :
  580 if number = 1 then 1000
```

```
 585 if number = 2 then 2000
 590 if number = 3 and flag$ = "off" then 6000
 595 if number = 3 and flag$ = "on" then 5000
 620 :
 625 rem *--incorrect choice message--*
 630 gosub 14000:rem incorrect choice
 635 goto 500:rem menu--check again
 640 :
 645 :
1000 rem **--display routine--**
1005 :
1010 rem *-titles-*
1015 scnclr
1020 color chtr,white
1025 print "Stock Symbol";
1030 print tab(28) "Hi Value";
1035 print tab(58) "Low Value"
1040 print
1045 :
1050 rem *-stock name-*
1052 color chtr,yellow
1055 for k = 1 to 10
1060 if k < 10 then print tab(1)
1062 k$ = str$(k)
1065 print k$;". ";
1070 print stk$(k);
1075 :
1080 rem *-high value-*
1085 m = val(hi$(k))
1090 gosub 27000:rem convert to fraction
1095 print tab(28)
1100 print str$(m);" ";f$;
1105 :
1110 rem *-low value-*
1115 m = val(low$(k))
1120 gosub 27000:rem convert to fraction
1125 print tab(58)
1130 print str$(m);" ";f$
1135 :
1140 next k
1145 :
1150 gosub 15000:rem pause routine
1155 goto 500
2000 rem **--display stock names--**
2005 scnclr:col = 31
2010 color chtr,white
2015 char 0,25,2,"CORRECT STOCK HI/LOW HISTORY"
2017 print:print
2020 color chtr,yellow
2025 for k = 1 to 10
2027 k$ = str$(k)
2030 if k < 10 then print tab(col + 1) k$;". ";stk$(k):goto 2040
2035 print tab(col)k$;". ";stk$(k)
2040 next k
2045 print tab(col) " 11. Menu"
2047 print
2050 print tab(col) " Which Number? ";
2055 gosub 19000:rem getkey routine
2060 w$ = t$:t$ = ""
2065 w = val(w$)
2070 if w < 1 or w > 11 then gosub 14000:goto 2000
2075 if w = 11 then 500:rem menu
2080 :
2085 :
3000 rem *--display for correction--*
3010 scnclr
3015 col = 17:row = 2:k = w
3020 color chtr,white
3025 char 0,col,row,"ENTERED INFORMATION"
```

```
 3030 color chtr,drkyel
 3035 char 0,col,row +  2,"1.  Name       :  " + stk$(k)
 3037 m = val(hi$(k)):gosub 27000:rem convert to fraction
 3040 char 0,col,row +  4,"2.  12 Mo. High: " + str$(m) + " " + f$
 3042 m = val(low$(k)):gosub 27000:rem convert to fraction
 3045 char 0,col,row +  6,"3.  12 Mo. Low : " + str$(m) + " " + f$
 3075 char 0,col,row +  8,"Is this correct? "
 3080 gosub 18000:rem y/n input routine
 3085 if yes$ = "y" then 2000:rem next stock data
 3090 :
 3095 char 0,col,row + 10,"Which number is wrong? "
 3100 gosub 19000:rem getkey subroutine
 3105 nb$ = t$:t$ = ""
 3110 nb = val(nb$)
 3115 if nb < 1 or nb > 3 then gosub 14000:goto 3095
 3117 if nb = 1 then cur$ = stk$(k)
 3120 if nb = 2 then gosub 16000:row = 10:cur$ = hi$(k)
 3122 if nb = 3 then gosub 16000:row = 10:cur$ = low$(k)
 3123 char 0,col,row + 10,"Current information is: " + cur$
 3125 char 0,col,row + 12,"Type in the correct information: "
 3130 gosub 19000:rem getkey subroutine
 3135 cinfo$ = t$:t$ = ""
 3137 flag$ = "on":rem info changed
 3140 if nb = 1 then stk$(k) = left$(cinfo$,16):goto 3000:rem ask again
 3145 if nb = 2 then  hi$(k) = left$(cinfo$, 5):goto 3000:rem ask again
 3150 if nb = 3 then low$(k) = left$(cinfo$, 5):goto 3000:rem ask again
 3165 :
 3170 :
 3190 :
 3195 :
 5000 rem **--file output routine--**
 5005 :
 5040 rem *--hi/low data file--*
 5045 dopen#2,"@Hi/Low Data",w
 5050 :
 5055 for k = 1 to 10
 5060 print#2,stk$(k)
 5065 print#2,hi$(k)
 5070 print#2,low$(k)
 5075 next k
 5080 :
 5085 dclose#2
 5090 :
 5095 :
 6000 rem **--return to program menu--**
 6005 scnclr
 6010 char 0,15,9
 6015 color chtr,rd
 6020 print "LOADING THE STOCK MARKET.MENU PROGRAM"
 6025 color chtr,yellow
 6030 run "stock.menu"
 6035 :
 6040 :
14000 rem *--incorrect choice message--*
14005 print:print
14010 color chtr,white:print tab(col) "Incorrect Choice!"
14015 color chtr,yellow:print:print tab(col) "Press ";
14020 color chtr,white:print "RETURN";
14025 color chtr,yellow:print " to continue:";
14030 gosub 19000:rem getkey subroutine
14035 return
14040 :
14045 :
14050 :
15000 rem **--pause routine--**
15005 color chtr,purple
15010 char 0,col,24,"Press the "
15015 color chtr,white
15020 print "RETURN";
```

```
15025 color chtr,purple
15030 print " key to continue: ";
15035 gosub 19000:rem getkey subroutine
15040 color chtr,yellow
15045 return
15050 :
15055 :
15060 :
16000 rem **--reminder routine--**
16005 scnclr:col = 12
16010 print tab(col + 8)
16015 color chtr,white
16030 print "*****---REMEMBER---*****"
16035 color chtr,yellow
16040 print
16050 print tab(col) "You must add the fraction as the last"
16060 print tab(col) "digit! 21 5/8 should be entered as: 215"
16070 print
16080 print tab(col) "If the number has no fraction, please"
16090 print tab(col) "enter a '0' after the number! 21 even"
16100 print tab(col) "should be entered as: 210"
16110 print:print
16115 color chtr,rd
16120 print tab(col + 11) "1/8 --------- = 1"
16130 print tab(col + 11) "1/4 --------- = 2"
16140 print tab(col + 11) "3/8 --------- = 3"
16150 print tab(col + 11) "1/2 --------- = 4"
16160 print tab(col + 11) "5/8 --------- = 5"
16170 print tab(col + 11) "3/4 --------- = 6"
16180 print tab(col + 11) "7/8 --------- = 7"
16190 print tab(col + 11) "EVEN -------- = 0"
16195 color chtr,yellow
16200 print
16210 return
16215 :
16220 :
16225 :
17000 rem **--convert to lowercase--**
17005 nc$ = ""
17010 for cv = 1 to len(cv$)
17015 x = asc(mid$(cv$,cv,1))
17020 if x > 192 then x = x - 128
17025 nc$ = nc$ + chr$(x)
17030 next cv
17035 :
17040 cv$ = nc$
17045 f = asc(left$(cv$,1))
17050 f = f + 128
17055 nc$ = chr$(f) + right$(cv$,len(cv$)- 1)
17060 return
17065 :
17070 rem na$ = originally typed name
17075 rem cv$ = converted to lowercase
17080 rem nc$ = 1st letter/uppercase
17085 :
17090 :
17095 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
```

```
18055 print
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
18090 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print chr$(157) " ";:rem cursor left
19052 print chr$(157);
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
19070 :
19075 :
21000 rem **--underline routine--**
21005 for i = 1 to sp
21010 print chr$(2);" ";:rem underline on
21015 next i
21020 print chr$(130):rem underline off
21025 return
21030 :
21035 :
21040 :
27000 rem **--convert to fraction--**
27010 f = m - int(m/10) * 10
27020 m = int(m/10)
27030 if f = 0 then f$ = ""
27040 if f = 1 then f$ = "1/8"
27050 if f = 2 then f$ = "1/4"
27060 if f = 3 then f$ = "3/8"
27070 if f = 4 then f$ = "1/2"
27080 if f = 5 then f$ = "5/8"
27090 if f = 6 then f$ = "3/4"
27100 if f = 7 then f$ = "7/8"
27110 return
27120 :
27130 :
```

The stock.dsp hi/low Program

```
  100 rem ***--display hi/low data-***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white = 16
  130 ::::rd =  3
  135 :green =  6
  140 ::blue =  7
  145 yellow = 10
  150 purple =  9
  155 drkyel = 10
  160 ::cyan = 12
  165 ::gray = 16
  170 ::chtr =  5
  175 bckgnd =  0
  195 :
  200 blank$ = chr$(10):rem blank line
  205 print chr$(14):rem upper/lower case mode
```

```
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 fast
  225 :
  230 :
  400 rem **--file input routine--**
  405 :
  410 dopen#1,"Hi/Low Data"
  415 for k = 1 to 10
  420 input#1,stk$(k)
  425 input#1,hi$(k)
  430 input#1,low$(k)
  435 next k
  440 dclose#1
  445 :
  450 :
 1000 rem **--display routine--**
 1005 :
 1010 rem *-titles-*
 1015 scnclr
 1020 color chtr,white
 1025 print "Stock Symbol";
 1030 print tab(28) "Hi Value";
 1035 print tab(58) "Low Value"
 1040 print
 1045 :
 1050 rem *-stock name-*
 1052 color chtr,yellow
 1055 for k = 1 to 10
 1060 if k < 10 then print tab(1)
 1062 k$ = str$(k)
 1065 print k$;". ";
 1070 print stk$(k);
 1075 :
 1080 rem *-high value-*
 1085 m = val(hi$(k))
 1090 gosub 27000:rem convert to fraction
 1095 print tab(28)
 1100 print str$(m);" ";f$;
 1105 :
 1110 rem *-low value-*
 1115 m = val(low$(k))
 1120 gosub 27000:rem convert to fraction
 1125 print tab(58)
 1130 print str$(m);" ";f$
 1135 :
 1140 next k
 1145 :
 1150 gosub 15000:rem pause routine
 6000 rem **--return to program menu--**
 6005 scnclr
 6010 char 0,15,9
 6015 color chtr,rd
 6020 print "LOADING THE STOCK MARKET.MENU PROGRAM"
 6025 color chtr,yellow
 6030 run "stock.menu"
 6035 :
 6040 :
15000 rem **--pause routine--**
15005 color chtr,purple
15010 char 0,col,24,"Press the "
15015 color chtr,white
15020 print "RETURN";
15025 color chtr,purple
15030 print " key to continue: ";
15035 gosub 19000:rem getkey subroutine
15040 color chtr,yellow
15045 return
15050 :
```

```
15055 :
15060 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print chr$(157) " ";:rem cursor left
19052 print chr$(157);
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
19070 :
19075 :
27000 rem **--convert to fraction--**
27010 f = m - int(m/10) * 10
27020 m = int(m/10)
27030 if f = 0 then f$ = ""
27040 if f = 1 then f$ = "1/8"
27050 if f = 2 then f$ = "1/4"
27060 if f = 3 then f$ = "3/8"
27070 if f = 4 then f$ = "1/2"
27080 if f = 5 then f$ = "5/8"
27090 if f = 6 then f$ = "3/4"
27100 if f = 7 then f$ = "7/8"
27110 return
27120 :
27130 :
```

The stock.display Program

```
  100 rem ***--display stock history--***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white = 16
  130 ::::rd =  3
  135 :green =  6
  140 ::blue =  7
  145 yellow = 10
  150 purple =  9
  155 drkyel = 10
  160 ::cyan = 12
  165 ::gray = 16
  170 ::chtr =  5
  175 bckgnd =  0
  195 :
  200 blank$ = chr$(10):rem blank line
  205 print chr$(14):rem upper/lower case mode
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 fast
  225 :
  230 :
  300 rem **--program title and message--**
  305 scnclr
  310 color chtr,white
  315 char 0,31,7,"STOCK HISTORY"
  320 color chtr,yellow
  325 char 0,15,11,"One Moment Please! Loading Stock Information."
  330 :
  335 :
  400 rem **--input pointer file--**
  405 dopen#1,"Stock.Ptr"
```

```
  410 input#1,ptr
  415 dclose#1
  425 :
  430 :
  500 rem **--input hi/low file--**
  502 dopen#2,"Hi/Low Data"
  505 for k = 1 to 10
  507 :
  510 input#2,stk$(k)
  515 input#2,hi$(k)
  520 input#2,low$(k)
  525 :
  530 m = val(hi$(k)):gosub 27000:rem convert to fraction
  535 hi$(k) = str$(m) + " " + f$
  540 :
  545 m = val(low$(k)):gosub 27000:rem convert to fraction
  550 low$(k) = str$(m) + " " + f$
  555 :
  560 next k
  565 dclose#2
  570 :
  575 :
  600 rem **--display stock names--**
  605 scnclr:col = 31
  610 color chtr,white
  615 char 0,31,2,"STOCK HISTORY"
  617 print:print
  620 color chtr,yellow
  625 for k = 1 to 10
  627 k$ = str$(k)
  630 if k < 10 then print tab(col + 1) k$;". ";stk$(k):goto 640
  635 print tab(col)k$;". ";stk$(k)
  640 next k
  645 print tab(col) " 11. Menu"
  647 print
  650 print tab(col) " Which Number? ";
  655 gosub 19000:rem getkey routine
  660 w$ = t$:t$ = ""
  665 w = val(w$)
  670 if w < 1 or w > 11 then gosub 14000:goto 600
  675 if w = 11 then 6000:rem menu
  680 :
  685 :
 1000 rem **--stock information--**
 1002 lgth = len(stk$(w))
 1003 col = (80 - lgth)/2
 1004 cl = col
 1005 scnclr
 1010 color chtr,white
 1015 char 0,col,2,(stk$(w))
 1020 color chtr,yellow
 1025 col = 10
 1035 :
 1040 :
 1100 rem *--titles--*
 1105 print:print
 1110 print "DATE" tab(18) "VOL" tab(35) "HIGH" tab(50) "LOW" tab(65) "CLOSE"
 1115 :
 1120 :
 1200 rem *--stock file input routine--*
 1202 cv$ = stk$(w):gosub 17000:stk$ = cv$
 1203 stk$ = stk$ + ".data"
 1205 dopen#3,(stk$)
 1210 :
 1215 for kx = 1 to ptr
 1220 gosub 9000:rem file input routine
 1225 :
 1230 print dte$ tab(18) v$;
 1235 :
```

```
1240 m = val(h$):gosub 27000:rem convert to fraction
1245 h = m
1250 z = instr(h$,"*")
1255 if z > 0 then nh$ = "*"
1260 print tab(34) h;f$;nh$;
1265 :
1270 m = val(l$):gosub 27000:rem convert to fraction
1275 l = m
1280 z = instr(l$,"*")
1285 if z > 0 then nl$ = "*"
1290 print tab(49) l;f$;nl$;
1295 :
1300 c = val(c$)
1305 if kx = 1 then c1 = c:v1 = val(v$)
1310 if kx = ptr then c2 = c
1315 cv = c
1320 z2 = cv
1325 gosub 22000:rem convert to decimal
1330 cv = z2
1335 m = c:gosub 27000:rem convert to fraction
1340 c = m
1345 print tab(64) c;f$
1350 :
1355 nh$ = "":nl$ = ""
1360 av = av + val(v$)
1365 ap = ap + cv
1370 :
1375 next kx
1380 :
1385 :
1390 dclose#3
1395 gosub 15000:rem pause routine
1400 :
1405 :
1500 rem **--display second page--**
1505 scnclr:col = c1
1510 color chtr,white
1515 char 0,col,2,(stk$(w))
1520 color chtr,yellow
1522 char 0,col,4:print
1525 col = 25
1530 print tab(col) "Current P/E ratio  = ";pe$
1535 print:ss$ = hi$(w):gosub 20000:hi$(w) = ss$
1540 print tab(col) "Current high price = ";hi$(w)
1545 print:ss$ = low$(w):gosub 20000:low$(w) = ss$
1550 print tab(col) "Current low  price = ";low$(w)
1555 print
1560 :
1565 av = av / (kx - 1)
1570 av$ = str$(av)
1572 ss$ = av$:gosub 20000:av$ = ss$
1575 dc = instr(av$,".")
1577 if dc = 0 then dc = len(av$) - 2
1580 print tab(col) "Average volume     = ";left$(av$,dc + 2)
1585 print
1590 :
1595 rem *--average price--*
1600 ap = ap / (kx - 1)
1605 ap$ = str$(ap)
1607 ss$ = ap$:gosub 20000:ap$ = ss$
1610 dc = instr(ap$,".")
1615 print tab(col) "Average price      = ";left$(ap$,dc + 2)
1620 :
1625 rem *--last price--*
1630 print
1635 z2 = c2
1640 gosub 22000:rem convert to decimal
1645 c2 = z2
1650 print tab(col) "Last Price         =";c2
```

```
1655 :
1660 rem *--first price--*
1665 z2 = c1
1670 gosub 22000:rem convert to decimal
1675 c1 = z2
1680 :
1685 rem *--difference in price--*
1690 cd = c2 - c1
1695 print
1700 print tab(col) "Price difference"
1705 print tab(col) "from 1st record    =";cd
1710 :
1720 rem *--another stock--*
1725 av = 0:ap = 0
1730 gosub 15000:rem pause routine
1795 goto 600
1990 :
1995 :
6000 rem **--return to program menu--**
6005 scnclr
6010 char 0,15,9
6015 color chtr,rd
6020 print "LOADING THE STOCK MARKET.MENU PROGRAM"
6025 color chtr,yellow
6030 run "stock.menu"
6035 :
6040 :
9000 rem **--data file input routine--**
9035 :
9040 rem *--data file--*
9050 :
9055 record#3,(kx),1
9060 input#3,dte$
9065 :
9070 record#3,(kx),11
9075 input#3,pe$
9080 :
9085 record#3,(kx),15
9090 input#3,v$
9095 :
9100 record#3,(kx),21
9105 input#3,h$
9110 :
9115 record#3,(kx),26
9120 input#3,l$
9125 :
9130 record#3,(kx),31
9135 input#3,c$
9140 :
9145 return
9175 :
9180 :
14000 rem *--incorrect choice message--*
14005 print:print
14010 color chtr,white:print tab(col) "Incorrect Choice!"
14015 color chtr,yellow:print:print tab(col) "Press ";
14020 color chtr,white:print "RETURN";
14025 color chtr,yellow:print " to continue:";
14030 gosub 19000:rem getkey subroutine
14035 return
14040 :
14045 :
14050 :
15000 rem **--pause routine--**
15005 color chtr,purple
15010 char 0,col,24,"Press the "
15015 color chtr,white
15020 print "RETURN";
15025 color chtr,purple
```

```
15030 print " key to continue: ";
15035 gosub 19000:rem getkey subroutine
15040 color chtr,yellow
15045 return
15050 :
15055 :
15060 :
17000 rem **--convert to lowercase--**
17005 nc$ = ""
17010 for cv = 1 to len(cv$)
17015 x = asc(mid$(cv$,cv,1))
17020 if x > 192 then x = x - 128
17025 nc$ = nc$ + chr$(x)
17030 next cv
17035 :
17040 cv$ = nc$
17045 f = asc(left$(cv$,1))
17050 f = f + 128
17055 nc$ = chr$(f) + right$(cv$,len(cv$)- 1)
17060 return
17065 :
17070 rem na$ = originally typed name
17075 rem cv$ = converted to lowercase
17080 rem nc$ = 1st letter/uppercase
17085 :
17090 :
17095 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 print
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
18090 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print chr$(157) " ";:rem cursor left
19052 print chr$(157);
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
19070 :
19075 :
20000 rem **--strip sign routine--**
20005 lgth = len(ss$)
20010 s2$  = right$(ss$,lgth - 1)
20015 ss$  = s2$
20020 return
20025 :
```

```
20030 :
20035 :
22000 rem **--convert to decimal--**
22005 z2 = z2 / 10:s3 = int(z2):dc = z2 - s3
22010 dc = (dc * 10)/ 8:z2 = s3 + dc:z2 = int(z2 * 1000 + .5) / 1000
22015 return
22020 :
22025 :
22030 :
27000 rem **--convert to fraction--**
27010 f = m - int(m/10) * 10
27020 m = int(m/10)
27030 if f = 0 then f$ = ""
27040 if f = 1 then f$ = "1/8"
27050 if f = 2 then f$ = "1/4"
27060 if f = 3 then f$ = "3/8"
27070 if f = 4 then f$ = "1/2"
27080 if f = 5 then f$ = "5/8"
27090 if f = 6 then f$ = "3/4"
27100 if f = 7 then f$ = "7/8"
27110 return
```

The stock.correct Program

```
  100 rem ***--correct stock information--***
  105 :
  110 :
  115 rem **--initialization--**
  120 :black =  1
  125 :white = 16
  130 ::::rd =  3
  135 :green =  6
  140 ::blue =  7
  145 yellow = 10
  150 purple =  9
  155 drkyel = 10
  160 ::cyan = 12
  165 ::gray = 16
  170 ::chtr =  5
  175 bckgnd =  0
  195 :
  200 blank$ = chr$(10):rem blank line
  205 print chr$(14):rem upper/lower case mode
  210 print chr$(8):rem disable char.set change
  215 color bckgnd,black
  220 fast:flag$ = "off"
  225 :
  230 :
  300 rem **--program title and message--**
  305 scnclr
  310 color chtr,white
  315 char 0,31,7,"CORRECT STOCK HISTORY"
  320 color chtr,yellow
  325 char 0,20,11,"One Moment Please! Loading Stock Information."
  330 :
  335 :
  400 rem **--input pointer file--**
  405 dopen#1,"Stock.Ptr"
  410 input#1,ptr
  415 dclose#1
  425 :
  430 :
  500 rem **--input hi/low file--**
  502 dopen#2,"Hi/Low Data"
  505 for k = 1 to 10
  507 :
  510 input#2,stk$(k)
  515 input#2,hi$(k)
  520 input#2,low$(k)
  525 :
```

```
 530 m = val(hi$(k)):gosub 27000:rem convert to fraction
 535 hi$(k) = str$(m) + " " + f$
 540 :
 545 m = val(low$(k)):gosub 27000:rem convert to fraction
 550 low$(k) = str$(m) + " " + f$
 555 :
 560 next k
 565 dclose#2
 570 :
 575 :
 600 rem **--display stock names--**
 605 scnclr:col = 31
 610 color chtr,white
 615 char 0,31,2,"CORRECT STOCK HISTORY"
 617 print:print
 620 color chtr,yellow
 625 for k = 1 to 10
 627 k$ = str$(k)
 630 if k < 10 then print tab(col + 1) k$;". ";stk$(k):goto 640
 635 print tab(col)k$;". ";stk$(k)
 640 next k
 645 print tab(col) " 11. Menu"
 647 print
 650 print tab(col) " Which Number? ";
 655 gosub 19000:rem getkey routine
 660 w$ = t$:t$ = ""
 665 w = val(w$)
 670 if w < 1 or w > 11 then gosub 14000:goto 600
 675 if w = 11 then 6000:rem menu
 680 :
 685 :
 800 rem **--get record number--**
 802 char 0,col,18:print chr$(27) + chr$(64)
 805 char 0,col,18," Which record for " + stk$(w) + " (1 to" + str$(ptr) + ")? "
 810 gosub 19000:rem getkey routine
 815 rec$ = t$:t$ = ""
 820 rec = val(rec$)
 825 if rec < 1 or rec > ptr then gosub 14000:goto 800
 830 :
 835 :
1000 rem **--stock information--**
1002 lgth = len(stk$(w))
1003 col = (80 - lgth)/2
1004 cl = col
1005 scnclr
1010 color chtr,white
1015 char 0,col,2,(stk$(w))
1020 color chtr,yellow
1025 col = 10
1035 :
1040 :
1200 rem *--stock file input routine--*
1202 cv$ = stk$(w):gosub 17000:stk$ = cv$
1203 stk$ = stk$ + ".data"
1205 dopen#3,(stk$)
1206 record#3,(rec)
1210 :
1215 gosub 9000:rem file input routine
1220 :
1225 flag$ = "off":rem info. has not yet been changed
1230 :
1500 rem **--convert to numbers--**
1505 pe = val(pe$)
1510 :v = val( v$)
1515 :h = val( h$)
1520 :l = val( l$)
1525 :c = val( c$)
1530 :
1535 :
```

```
1800 rem *--display for correction--*
1810 scnclr
1815 col = 20:row = 1
1820 color chtr,white
1825 char 0,col,row,(stk$(w)) + "'s data for " + dte$
1830 color chtr,drkyel
1832 row = 2
1835 char 0,col,row +  2,"1.  P/E ratio ---- :  " + str$(pe)
1836 char 0,col,row +  4,"2.  Volume ------- :  " + str$( v)
1837 m = h:gosub 27000:rem convert to fraction
1840 char 0,col,row +  6,"3.  High --------- :  " + str$(m) + " " + f$
1842 m = l:gosub 27000:rem convert to fraction
1845 char 0,col,row +  8,"4.  Low ---------- :  " + str$(m) + " " + f$
1850 m = c:gosub 27000:rem convert to fraction
1855 char 0,col,row + 10,"5.  Close -------- :  " + str$(m) + " " + f$
1875 char 0,col,row + 12,"Is this correct? "
1880 gosub 18000:rem y/n input routine
1885 if yes$ = "y" and flag$ = "off" then 600:rem ask again
1887 if yes$ = "y" and flag$ = "on" then 2000:rem file correction
1890 :
1892 char 0,0,row + 14:print chr$(27) + chr$(64)
1895 char 0,col,row + 14,"Which number is wrong? "
1900 gosub 19000:rem getkey subroutine
1905 nb$ = t$:t$ = ""
1910 nb = val(nb$)
1915 if nb < 1 or nb > 5 then gosub 14000:goto 1892
1917 if nb = 1 then cur = pe
1918 if nb = 2 then cur = v
1919 if nb = 3 then gosub 16000:row =  6:cur = h
1920 if nb = 4 then gosub 16000:row =  6:cur = l
1921 if nb = 5 then gosub 16000:row =  6:cur = c
1922 cur$ = str$(cur)
1923 char 0,col,row + 14,"Current information is: " + cur$
1925 char 0,col,row + 16,"Type in the correct information: "
1930 gosub 19000:rem getkey subroutine
1935 cinfo$ = t$:t$ = ""
1937 flag$ = "on":rem info has been changed
1940 if nb = 1 then  pe = val(cinfo$):goto 1800:rem ask again
1945 if nb = 2 then   v = val(cinfo$):goto 1800:rem ask again
1950 if nb = 3 then   h = val(cinfo$):goto 1800:rem ask again
1955 if nb = 4 then   l = val(cinfo$):goto 1800:rem ask again
1960 if nb = 5 then   c = val(cinfo$):goto 1800:rem ask again
1965 :
1970 :
1975 :
2000 rem **--convert back to strings--**
2005 ss$ = str$(pe):gosub 20000:pe$ = ss$
2010 ss$ = str$( h):gosub 20000: h$ = ss$
2015 ss$ = str$( l):gosub 20000: l$ = ss$
2020 ss$ = str$( c):gosub 20000: c$ = ss$
2025 ss$ = str$( v):gosub 20000: v$ = ss$
2030 if len(pe$) > 3 then pe$ = left$(pe$,3)
2035 if len( v$) > 5 then  v$ = left$( v$,5)
2040 if len( h$) > 4 then  h$ = left$( h$,4)
2045 if len( l$) > 4 then  l$ = left$( l$,4)
2050 if len( c$) > 4 then  c$ = left$( c$,4)
2055 :
2060 :
3000 rem **--data file output routine--**
3005 scnclr:col = 31:row = 7
3010 color chtr,white
3015 char 0,col,row,"ONE MOMENT PLEASE!"
3020 color chtr,yellow
3025 char 0,col,row + 2,"Updating " + stk$(w) + "'s file."
3026 print
3027 :
3030 cv$ = stk$(w):gosub 17000:stk$ = cv$
3035 :
3040 rem *--data file--*
```

```
3042 stk$ = stk$ + ".data"
3045 dopen#3,(stk$),135
3047 record#3,(rec)
3050 :
3055 record#3,(rec),1
3060 print#3,dte$
3065 :
3070 record#3,(rec),11
3075 print#3,pe$
3080 :
3085 record#3,(rec),15
3090 print#3,v$
3095 :
3100 record#3,(rec),21
3105 print#3,h$
3110 :
3115 record#3,(rec),26
3120 print#3,l$
3125 :
3130 record#3,(rec),31
3135 print#3,c$
3140 :
3142 record#3,(rec)
3145 dclose#3
3150 :
3155 nh$ = "":nl$ = ""
3160 :
3165 :
3175 :
4000 rem **--another stock--**
4005 scnclr
4010 color chtr,white
4015 char 0,col,7,(stk$(w)) + "'s"
4020 color chtr,yellow
4025 print " file has been updated!"
4030 gosub 15000:rem pause routine
4035 goto 600:rem start again
4040 :
4045 :
6000 rem **--return to program menu--**
6005 scnclr
6010 char 0,15,9
6015 color chtr,rd
6020 print "LOADING THE STOCK MARKET.MENU PROGRAM"
6025 color chtr,yellow
6030 run "stock.menu"
6035 :
6040 :
9000 rem **--data file input routine--**
9035 :
9040 rem *--data file--*
9050 :
9055 record#3,(rec),1
9060 input#3,dte$
9065 :
9070 record#3,(rec),11
9075 input#3,pe$
9080 :
9085 record#3,(rec),15
9090 input#3,v$
9095 :
9100 record#3,(rec),21
9105 input#3,h$
9110 :
9115 record#3,(rec),26
9120 input#3,l$
9125 :
9130 record#3,(rec),31
9135 input#3,c$
```

```
9140 :
9145 dclose#3
9150 return
9175 :
9180 :
14000 rem *--incorrect choice message--*
14005 print:print
14010 color chtr,white:print tab(col) "Incorrect Choice!"
14015 color chtr,yellow:print:print tab(col) "Press ";
14020 color chtr,white:print "RETURN";
14025 color chtr,yellow:print " to continue:";
14030 gosub 19000:rem getkey subroutine
14035 return
14040 :
14045 :
14050 :
15000 rem **--pause routine--**
15005 color chtr,purple
15010 char 0,col,24,"Press the "
15015 color chtr,white
15020 print "RETURN";
15025 color chtr,purple
15030 print " key to continue: ";
15035 gosub 19000:rem getkey subroutine
15040 color chtr,yellow
15045 return
15050 :
15055 :
15060 :
16000 rem **--reminder routine--**
16005 scnclr:col = 12
16010 print tab(col + 8)
16015 color chtr,white
16030 print "*****---REMEMBER---*****"
16035 color chtr,yellow
16040 print
16050 print tab(col) "You must add the fraction as the last"
16060 print tab(col) "digit! 21 5/8 should be entered as: 215"
16070 print
16080 print tab(col) "If the number has no fraction, please"
16090 print tab(col) "enter a '0' after the number! 21 even"
16100 print tab(col) "should be entered as: 210"
16110 print:print
16115 color chtr,rd
16120 print tab(col + 11) "1/8 --------- = 1"
16130 print tab(col + 11) "1/4 --------- = 2"
16140 print tab(col + 11) "3/8 --------- = 3"
16150 print tab(col + 11) "1/2 --------- = 4"
16160 print tab(col + 11) "5/8 --------- = 5"
16170 print tab(col + 11) "3/4 --------- = 6"
16180 print tab(col + 11) "7/8 --------- = 7"
16190 print tab(col + 11) "EVEN -------- = 0"
16195 color chtr,yellow
16200 print
16210 return
16215 :
16220 :
16225 :
17000 rem **--convert to lowercase--**
17005 nc$ = ""
17010 for cv = 1 to len(cv$)
17015 x = asc(mid$(cv$,cv,1))
17020 if x > 192 then x = x - 128
17025 nc$ = nc$ + chr$(x)
17030 next cv
17035 :
17040 cv$ = nc$
17045 f = asc(left$(cv$,1))
```

```
17050 f = f + 128
17055 nc$ = chr$(f) + right$(cv$,len(cv$)- 1)
17060 return
17065 :
17070 rem na$ = originally typed name
17075 rem cv$ = converted to lowercase
17080 rem nc$ = 1st letter/uppercase
17085 :
17090 :
17095 :
18000 rem **--y/n input subroutine--**
18005 print "Type a ";
18010 color chtr,white:print "Y";
18015 color chtr,yellow:print " or ";
18020 color chtr,white:print "N";
18025 color chtr,yellow:print " :";
18030 gosub 19000:rem getkey subroutine
18035 yes$ = t$:t$ = ""
18040 print
18045 if yes$ = "y" or yes$ = "Y" then yes$ = "y":return
18050 if yes$ = "n" or yes$ = "N" then yes$ = "n":return
18055 print
18060 color chtr,white:print "Incorrect Choice!"
18065 color chtr,yellow
18070 print
18075 goto 18000:rem ask again
18080 :
18085 :
18090 :
19000 rem **--getkey subroutine--**
19005 getkey a$
19010 if a$ = chr$(20) then 19040:rem delete char.
19015 if a$ = chr$(13) then return:rem return key
19020 print a$;
19025 t$ = t$ + a$
19030 goto 19000
19035 :
19040 lg = len(t$)
19045 if lg < 1 then 19000
19050 print chr$(157) " ";:rem cursor left
19052 print chr$(157);
19055 t$ = left$(t$,lg - 1)
19060 goto 19000
19065 :
19070 :
19075 :
20000 rem **--strip sign routine--**
20005 lgth = len(ss$)
20010 s2$  = right$(ss$,lgth - 1)
20015 ss$  = s2$
20020 return
20025 :
20030 :
20035 :
22000 rem **--convert to decimal--**
22005 z2 = z2 / 10:s3 = int(z2):dc = z2 - s3
22010 dc = (dc * 10)/ 8:z2 = s3 + dc:z2 = int(z2 * 1000 + .5) / 1000
22015 return
22020 :
22025 :
22030 :
27000 rem **--convert to fraction--**
27010 f = m - int(m/10) * 10
27020 m = int(m/10)
27030 if f = 0 then f$ = ""
27040 if f = 1 then f$ = "1/8"
27050 if f = 2 then f$ = "1/4"
27060 if f = 3 then f$ = "3/8"
```

```
27070 if f = 4 then f$ = "1/2"
27080 if f = 5 then f$ = "5/8"
27090 if f = 6 then f$ = "3/4"
27100 if f = 7 then f$ = "7/8"
27110 return
```

Chapter 12

Advanced Database Options

There are a number of additional tasks that are often included with database systems. A detailed explanation of these advanced topics is not included, due to the added complexity and amount of code required. None of these tasks is essential for the type of files covered in this book, but all could be added to the relative file systems to make those systems more complete.

The problem in adding advanced options is that such additions significantly increase the amount and complexity of the programming. Therefore, programmers are often faced with a decision. Increasing a database's power and ease of use increases the time required to create the system (and usually, therefore, the money invested in the system). As indicated, such topics are not necessary for an introductory book, but because the Commodore 128 is at least a second generation computer for many people, I decided to add a final chapter providing a brief explanation of some advanced tasks possible with relative-access files.

INDEXING

Indexing is a powerful tool used to allow very rapid searches through large files or series of files. In its simplest form, indexing is just what its name implies. An index to a book provides a quick way of finding a specific piece of information by providing the exact page number for that information. Indexing a file is very similar to indexing a book. A separate file is maintained that provides an ordered list of a certain field for all records in the file. Using the book example, a separate part of the book is used to keep an ordered list of certain key words or topics used on any page within the book. In a mailing list file, for example, the names probably would be added to the file in no particular order; John

Jones might be added first, Bill Zachary second, and Sam Adams third. An alphabetical index by name would list Adams first, Jones second, and Zachary third. In addition to the name, the index file would list the record number for each name, which is similar to the page number provided in a book index. Therefore, the index file would look something like this:

```
Adams*Sam**3
Jones*John**1
Zachary*Bill**2
```

Subsequent searches would access the index file, find the desired name, obtain the associated record number, and access the correct record within the data file. Such a procedure is usually much faster than sequentially searching through a large data file for the desired information.

The problem with indexing occurs in the creation and maintenance of the index. When a book is indexed, a decision must be made regarding which words and/or topics are to be included in the index. Every word in the book is not indexed. Similarly, every field in the file is usually not indexed (although some files may, in fact, require such indexing).

Unlike books, files frequently increase in size, thus necessitating an update of all index files. Such updating can be handled in a number of different ways. The user might need to specifically instruct the computer to create a new index file after the file has been altered by the addition or correction of information.

Some database systems automatically update the index files whenever information in the file is added to the file or changed. The safest method is the automatic updating process, because a user might not remember to create a new index after altering the file. Regardless, some method must allow for updating all index files after the data file has been changed. Creating and maintaining index files means considerably more programming, but if the application requires near instant access to any piece of information in the file, some form of indexing is often the answer.

FILE UTILITY PROGRAMS

I have included some examples of file utility programs—copying and correcting functions are two such programs. Additional techniques are required for very large files.

Copying Large Files

If the entire file will not fit within the computer's memory, the copy program must keep track of the number of records that have been copied, the number of records that must still be copied, and the number of records currently being copied.

Multiple Data Diskettes with a Single File

Some files might need to extend over more than a single data diskette. The use of multiple data diskettes for a single file creates many new programming challenges. For example, search and sort routines must be structured to prompt the user to insert the additional data diskettes, and if the file is very large, temporary sort or search files must be maintained. Disk-full checks must be made. File copying becomes a major task if it is done from within the program and not from the operating system.

Merging Two or More Files

Merging two or more files into one large file is a task that some applications might require. The need for the reverse of this operation is also very likely with certain applications, that is, a large file might better be broken into smaller files that have common characteristics. A good database system might contain utility programs that allow the user the option of either combining small files into one or more large files, or breaking one or more large files into several small files. A good database system should allow the user as much flexibility as possible. If the user does not need flexibility, then the addition of such utility programs is simply a waste of time, effort, and money.

Adding Fields to an Existing File

Another useful (though hopefully unnecessary) option is the capability of increasing the number of fields within each record. For example, a user might find that more information is needed for each record in the file. If the database system is sophisticated, the system will contain utility programs that allow the definition and addition of other fields to the original file. Other programs are then needed to access each of the previously entered records and enter the additional information. If the file is indexed or cross-indexed, those files must also be changed and updated to include the new information. It is also possible that additional indices might need to be created.

Transferring Information to Other Applications

Some database systems include utility programs that allow information in the file to be translated into a standard format that can then be used by many commercial application programs. There are at least two such standard formats, the DIF (document interchange format) and the SYLK (symbolic link) format. In addition, many application programs include utility programs that read sequential files with a specific format or makeup. The database system must then include utility programs that allow for the creation of such files (DIF, SYLK, or specific sequential) from the information stored in relative file format.

ADDITIONAL DISPLAY PROGRAMS

As indicated, the greatest source of variety in database programs is in the display of information contained in a file. Frequently, the user's needs change, and new or additional display options are necessary. There are two basic media for the display of file information: screen display and paper display. With just these two basic media, the amount of possible formats is enormous.

Graphic representation of file information is growing in popularity. This graphic representation can exist either on screen or paper, in color or black and white (or green or amber), and in 40 column, 80 column, or 132 column format. Information can be grouped and summarized or presented individual record by individual record. *Windowing* can display two or more different sets of information at the same time.

With this brief description, I have not attempted to explain all the possibilities for the display of file information. Instead, I merely want to suggest some of the choices facing both users and programmers when a database system is used.

Appendix

Answers to Chapter Questions

CHAPTER 1 ANSWERS

1. By the three-character file type abbreviation.
2. PRG.
3. SEQ or REL data files.
4. Data files.
5. Press the F3 function key.
6. BASIC or other program-language files.
7. Data files.
8. The size of the file, the file name, and the three-character file-type abbreviation.

CHAPTER 2 ANSWERS

1. True.
2. DIRECTORY, CATALOG, or pressing the F3 key.
3. Disk Operating System.
4. Three.
5. RUN " (with a BASIC program file name).
6. SAVE or DSAVE " (with a BASIC program file name).
7. False; it only takes a copy.
8. The previous version is not erased and replaced with the new version unless you have added the REPLACE (@) character.

9. False, it shows what is in the computer's program memory.
10. HEADER (or use the FORMAT A DISK part of the DOS shell program).
11. False, they are files when stored on diskette.
12. NEW erases whatever is in the computer's memory.

CHAPTER 3 ANSWERS

1. Data files.
2. Sequential-access, direct-access (relative files).
3. Sequential-access.
4. 16.
5. False.
6. 2; immediate and deferred.
7. OPEN, CLOSE, some method of writing to the file (PRINT#), and some method of reading from the file (INPUT# or GET#).
8. REMark.
9. SEQ.
10. It brings information into the computer from the disk drive.
11. $
12. They are names of locations in the computer's memory where values can be temporarily stored.

CHAPTER 4 ANSWERS

1. False; opening a sequential file with the REPLACE character (@) erases the entire contents of any file with the same name. Without the REPLACE character, opening a sequential file that already exists with a normal "dopen#,file number, file name,w" and attempting to write to that file produces an error condition. Opening a sequential file with the APPEND command adds the new information to the end of the designated file.
2. DOPEN.
3. Input, correction, and file creation routines.
4. An array.
5. !
6. DIMension.
7. A FOR-NEXT loop.
8. IF-THEN statements.

CHAPTER 5 ANSWERS

1. True.
2. DELETE.
3. Disk or diskette.
4. GOTO.
5. F2 = DLOAD with the necessary ".
   F3 = DIRECTORY and the equivalent of pressing the RETURN key.
   F5 = DSAVE with the necessary ".
   F6 = RUN and the equivalent of pressing the RETURN key.

F7 = LIST and the equivalent of pressing the RETURN key.

6. APPEND: a sequential file can be opened with the APPEND command.

CHAPTER 6 ANSWERS

1. True.
2. TAB. TAB must be used from within a print statement.
3. GOSUB.
4. RETURN.
5. True.
6. Bubble Sort, Quicksort, and Shell-Metzner Sort.
7. LEFT$, RIGHT$, and MID$.
8. MID$.
9. LEN, STR$, VAL, and DIM.
10. False. The first file is replaced only if the replace character (@) is used.

CHAPTER 7 ANSWERS

1. False.
2. It becomes Mail Data Backup.
3. SCRATCH..
4. RENAME.
5. False.

CHAPTER 8 ANSWERS

1. C.
2. False.
3. False. Variables are often used for file names.
4. String.
5. STR$.
6. VAL.
7. INT.
8. STATUS.

CHAPTER 9 ANSWERS

1. Divided and undivided relative files.
2. False.
3. INPUT#, PRINT#.
4. False.
5. L.
6. True.
7. True.
8. Bytes.
9. False.
10 Record.
11. Offset or field position.

CHAPTER 10 ANSWERS

1. Status = 16.
2. Find$.
3. Shell-Metzner.
4. Concatenate.
5. False.

CHAPTER 11 ANSWERS

1. False.
2. Deciding which information is of value.
3. Creation and addition, display, and correction.
4. Display.

Index

Commodore 128 Data File Programming

If you are intrigued with the possibilities of the programs included in *Commodore 128 Data File Programming* (TAB Book No. 2805), you should definitely consider having the ready-to-run disk containing the software applications. This software is guaranteed free of manufacturer's defects. (If you have any problems, return the disk within 30 days, and we'll send you a new one.) Not only will you save the time and effort of typing the programs, the disk eliminates the possibility of errors that can prevent the programs from functioning. Interested?

Available on disk for the Commodore 128 with an 80-column monitor at $24.95 for each disk plus $1.00 each shipping and handling.

Diskettes containing other application programs and file systems also are available, as well as customized systems to suit particular needs. The following systems can be purchased for $25 each ($27.50 outside of North America):

Sports System—Used to keep track of individual player statistics—child or adult.

Video System—Used to keep track of information relating to video tapes and video discs.

Library System—Used to maintain personal library information.

C-128 Mailing List System—A fairly sophisticated and complete relative-access database system used to store and retrieve names, addresses, and phone numbers; category of code classification is included.

ISBN 0-8306-0205-4