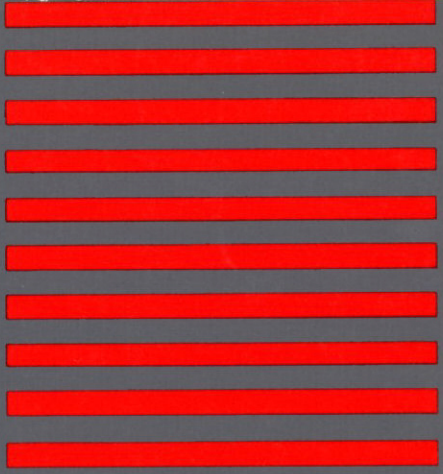


# COMPUTE!'s Guide to Extended BASIC Home Applications on the TI-99/4A

Turn your TI-99/4A into a powerful home management system with seven sophisticated programs, from file organization to financial planning. Includes clear, easy-to-understand explanations of useful Extended BASIC techniques.

Christopher Flynn

A **COMPUTE! Books** Publication

$12.95

# COMPUTE!'s Guide to Extended BASIC Home Applications on the TI-99/4A

Christopher Flynn

COMPUTE! Publications, Inc.
One of the ABC Publishing Companies
Greensboro, North Carolina

Printed in the United States of America

ISBN 0-942386-41-8

10 9 8 7 6 5 4 3 2 1

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is one of the ABC Publishing Companies and is not associated with any manufacturer of personal computers. TI-99/4A is a trademark of Texas Instruments.

# Contents

# Foreword

Before computers found their way into the home, most of them were management tools in business. The business computer was, and still is, used to organize and analyze. A home computer like the TI-99/4A can be used for much the same thing. This book can help you turn your TI computer into a home management system.

*COMPUTE!'s Guide to Extended BASIC Home Applications on the TI-99/4A* contains a series of useful home applications that will turn your computer into an efficient management tool. The programs in this book will help you organize your files, as well as analyze your finances, helping you make decisions about your money. These programs will do many of the repetitious tasks that people hate to do, but computers do so well.

But this book is more than just a series of home applications. You'll also learn how to use your TI-99/4A to get more from your Extended BASIC cartridge. Whether you're an experienced programmer, or just beginning, you'll find this book full of powerful techniques that you can use in your own programming. Each chapter is designed to make these techniques accessible to any TI-99/4A computer user. If you're only interested in using the programs, for instance, just type them in and follow the clear and simple directions. If you want to learn more about Extended BASIC, however, you'll appreciate the program explanations. There are even a number of subprograms that can be easily added to your own programs.

You'll use this book not just once, when you type in the home application programs, but again and again, as you explore the expanded capabilities of your computer. As with all COMPUTE! books, the programs are ready to type in and enjoy.

# Chapter 1

# Introduction

# Introduction

The story of home computers is a familiar one. Microcomputers emerged from the engineering laboratories in January 1975. That was when the magazine *Popular Electronics* featured the Altair computer. The computer was a kit with a whopping 256 bytes (yes, bytes) of memory standard. Programs were entered through toggle switches on the computer's front panel. All programming was done in machine language.

Though the computers of that day seem almost laughable, the contributions made by those technology pioneers were and are still significant. An entire microcomputer industry was born. The market suddenly appeared enormous and potentially profitable. Large companies such as Tandy and Texas Instruments announced their own products. Computers in the home became a reality. Consumers no longer needed a soldering iron for a debugging tool.

Brands of computers can be purchased for under $100. Most households can afford a home computer. These low prices indicate the tremendous progress that has been made in microelectronics. Literally millions of low-cost home computers have been sold.

When home computers were a rarity, friends and relatives expressed amazement at the whole idea. They consistently asked the question "What do you do with it?" The pioneering programmer-technicians were proud of their accomplishments. That one question, though, always evoked a blush and a somewhat incomprehensible response.

Today, the amazement is gone. Personal computers permeate our offices, schools, and neighborhoods. Office work and computers go hand in hand, and it doesn't require a great deal of imagination to see the benefits of computers in schools. But consider computers at home. Has that nagging question "What do you do with it?" ever been answered satisfactorily? The question doesn't seem to cause much embarrassment now, but it doesn't mean we've answered it.

This book is dedicated to all of you who have a TI-99/4A

and would like to put it to practical use around the home. The purpose of this book is twofold.

First, this book will guide you through a quick review of TI's Extended BASIC programming language. You will learn the tips and techniques that you will need to do your own programming.

Second, this book will provide you with many practical programs that you can actually put to use.

Chapters 2 and 3 contain many small programs and examples that illustrate various programming techniques. Chapters 4–8 each describe a specific application program. These programs build on the techniques described in Chapters 2 and 3. Yes, the program listings look quite long. But the time that you spend typing in the programs will be well worth it. These are programs that you can really use.

## Why Extended BASIC?

Extended BASIC is an advanced programming language. It comes in a command module that plugs right into your TI-99/4A. You do not have to send your computer back to the factory to install Extended BASIC. So you can obtain this language whenever your budget or inclination permits.

All of the programs in this book are written in Extended BASIC. Why use another programming language? Although it is limited in some respects, why isn't TI BASIC good enough?

Let's look at a few of the features of Extended BASIC. Then you can decide for yourself.

**Multiple statements per line.** Extended BASIC allows you to put more than one statement on a line. This conserves memory and helps programs run a little faster.

**Improved IF-THEN statements.** In TI BASIC, a line number must follow a THEN or ELSE clause. In Extended BASIC, you can put one or more statements after the THEN or ELSE clauses. Programs are much easier to read because there are fewer line number references to sort out. You can also combine tests with the AND, OR, and NOT relational operators.

**Subprograms.** You can write your own general-purpose subprograms. (Some subprograms such as COLOR or SCREEN are built into the 99/4A.) You can use your homemade subprograms over and over again for different applications.

**Screen formatting.** Extended BASIC has a set of commands for displaying and accepting either strings or numbers from any location on the screen. These commands are extremely useful for any kind of data entry application.

**Error trapping.** Extended BASIC programs can tell when a program or data input error has occurred. Rather than automatically letting the TI stop your program, you can design corrective steps.

**Access to machine resources.** The 99/4A uses some rather clever hardware. Besides the sound generator, Extended BASIC can make the Speech Synthesizer talk. Extended BASIC can also define and manipulate movable graphics blocks, commonly called *sprites.*

This gives you an idea of what you can expect from Extended BASIC. Extended BASIC uses the same extensive file-handling commands that TI BASIC does. All of these features result in a well-designed language that compares favorably to any commercially available BASIC.

Now, if you would really like an idea of what Extended BASIC can do, try Program 1-1. Are you suitably impressed by this little six-line program? It defines 26 sprites with a random color and random initial velocity. As you can imagine, sprites are very handy for game programs. Sprites also find their way into application programs. The "Appointment Calendar" program uses sprites.

### Program 1-1. Extended BASIC Demonstration

```
100 REM EXTENDED BASIC DEMO
110 DISPLAY AT(12,6)ERASE ALL:"C O M P U T E !"
120 FOR I=1 TO 26 :: CHARNO=64+I
130 CALL SPRITE(#I,CHARNO,3+13*RND,89,113,(10*RND+
    1)+20*(.5>RND),(10*RND+1)+20*(.5>RND))
140 NEXT I
150 GOTO 150
```

### Expansion Possibilities

A 99/4A console, Extended BASIC, and a cassette tape recorder form the nucleus of a very capable system. Many practical and entertaining programs can be written and used with just this equipment. In fact, most of the programs in this book are designed for just that configuration. There comes a time, however, when every computer owner would like to

expand the system. Hardware expansion opens up new software capabilities.

### Peripheral Expansion System

One way of expanding your 99/4A centers on the Peripheral Expansion System (PES) from Texas Instruments. The PES consists of an expansion box and various printed circuit boards or *cards* which plug into it. The expansion box is almost literally an empty box. It does contain a power supply for the optional cards. It also has sockets (the *motherboard)* for the cards. A long, slightly unwieldy cable connects the expansion box to the connector on the right side of the 99/4A. By itself, the expansion box adds absolutely no new capabilities to your TI. The chief virtue of the expansion box is that it is unbelievably solid. If you have little children, you need not worry about damage to the expansion box or its contents.

The cards that plug in the expansion box are what give us additional capabilities. We'll discuss the memory expansion, RS232, and disk controller cards. These can all be utilized by Extended BASIC.

### Memory Expansion

The memory expansion card is advertised as a 32K memory card. The TI already has 16K. Usually, adding 32K to 16K would give 48K of memory for Extended BASIC. This is not exactly the case here. The 32K card is really divided into a 24K section and an 8K section. The 8K section is for machine language subroutines. Extended BASIC itself cannot use that area of memory. Furthermore, Extended BASIC partitions the remaining memory into a 24K section and a 16K section. The program and numeric variables stay in the 24K section and string variables are confined to the 16K section.

These divisions cannot be altered. They are due to the hardware design of the TI. So keep in mind that your program and numeric variables must stay under a 24K limit. Unless you are working with very large numeric arrays, you will probably find that there is ample memory available.

### RS232 Interface Card

Adding a printer to your TI opens up many possibilities. You can obtain program listings, print reports, and even do word processing. The RS232 card provides the electronics that are needed for the TI to send the proper signals to the printer.

RS232 is actually the name of an industry communications standard. The RS232 standard describes the electrical signals between two pieces of equipment. Since RS232 is an industry standard, you should have little trouble in obtaining and connecting a printer.

The RS232 card actually contains provisions for the connection of two RS232 devices. You can hook up a printer and a plotter or a printer and a modem (a device for transmitting and receiving signals over the telephone).

In addition, the RS232 card has what is called a parallel port. This is another way of connecting the TI to an external device. A parallel port transmits and receives data eight bits at a time. The RS232 port is a serial port and works with a stream of data one bit at a time. The parallel port, on the surface, appears to have the advantage of speed. However, mechanical devices such as printers operate at speeds much slower than the computer. So the potential speed advantage may not be realized.

There is an industry-wide informal standard for parallel port connections. However, there is the possibility of running across some variations. If you are planning on using a printer with a parallel connection, make sure that it is guaranteed as TI compatible.

So, one RS232 card gives you two RS232 serial ports and one parallel port. This should suffice for most applications. If not, a second RS232 Interface Card can be put in the expansion box. The second card requires a few simple hardware jumper changes. This is so the computer will recognize it as RS232 card number two. (In the past, TI was happy to make these changes; but now it would be wise to see if the service is still available before making any purchases.)

### Disk Controller Card

The Disk Controller contains the electronics that are necessary for the storing and retrieving of data with a disk system. If you are not familiar with disks, you can think of them as a cross between a record and a tape. The diskette is the storage medium. It is round and enclosed in a nonremovable square jacket. The diskette rotates, like a record, in the disk drive. Like a tape, data is stored as magnetic impulses. These magnetic impulses are placed on the diskette by a read-write head which comes in contact with the diskette. The read-write head

moves, as does a tone arm, so that it can access different portions of the diskette.

There are single-sided and double-sided disk drives and diskettes. As the name implies, single-sided drives and media can store data on only one side of a diskette. There is enough room for about 90,000 characters of data. Double-sided drives and media store data on both sides of a diskette. Double-sided diskettes can handle about 180,000 characters of data. However, TI has sold only single-sided drives. Double-sided drives must be purchased from various other sources.

The expansion box will accommodate only one disk drive. It is possible to add two additional drives, but these extra drives must have their own power supply and cabinet.

A vital piece of software is supplied with the disk controller. The software is the Disk Manager Command Module. With it, you can see what files are stored on a particular diskette, copy files, rename files, delete files, and format diskettes (get them ready for recording).

The Disk Manager Command Module also contains a set of very extensive test programs so you can test diskettes to make sure that they are good. Disk Manager 2 (sold after January 1983) provides the software for double-sided drives. (Disk Manager 1 can read and write double-sided diskettes, but because of a software problem, cannot format them.)

An operating system is a set of programs (software) that almost always comes with disk systems. Operating systems handle all of the minute programming details of the equipment connected to a computer. This is so you don't have to worry about such things every time you write your own programs. Operating systems work as traffic cops. They make sure that all parts of the computer work together smoothly.

Many operating systems come on diskette. The software is loaded into the computer's memory when the computer is turned on. Such operating systems take up space on both the diskette and the computer's memory. The TI's disk operating system is actually stored permanently in Read Only Memory (ROM) chips on the disk controller card. This software handles disk-related tasks. Most of the operating system is stored permanently in the 99/4A console itself. Very little memory space is consumed by the operating system. A total of about 4.5K of the 16K section of memory is used for various purposes. Less than 1K of the available space per disk drive is

used by the operating system. And this space we would gladly give up! This is where the TI keeps a list of the files on each diskette. This list or directory is pretty important for proper disk operation.

### Getting Help

When you opened your computer for the first time, you probably came across several manuals. These are well written and contain a lot of useful information. If you have a question, you can often find the answer in these manuals.

Consider joining the International 99/4 Users-Group. It is an independent organization specializing in TI hardware and software. It also maintains an extensive library of programs contributed by people like you. Its address is:

International 99/4 Users-Group, Inc.
P.O. Box 67
Bethany, OK 73008

Best of all, join your local TI users group. If you haven't heard about it, call TI. They will tell you if there is one near you. You will find that local groups have a lot to offer. They sponsor guest speakers, they arrange group purchases of equipment and supplies, and they have classes for all levels of users. They enjoy sharing and exchanging ideas.

# Chapter 2

# Extended BASIC Techniques

# Extended BASIC Techniques

Extended BASIC is a complete programming language in a command module. It is very similar to TI BASIC in many ways. Some of the features present in both languages are:

- 15-character variable names,
- 13- to 14-digit calculation accuracy,
- strings up to 255 characters,
- built-in subprograms for sound, color, and custom character definition, and
- a complete file management system.

So virtually everything that you've learned about TI BASIC still applies to Extended BASIC. Extended BASIC is a new language which builds on TI BASIC. Extended BASIC does introduce several new programming concepts. These are explained in a 224-page manual that comes with the command module. The manual gives the complete format or syntax of each Extended BASIC command. In addition, the manual presents many examples. The Extended BASIC manual is a reference manual: It is where you should turn when you have questions about specific features of the language.

This chapter will not be a rewrite of the manual. Rather, it will go over the high points of Extended BASIC and discuss some tips and techniques. Some examples and short programs will be presented that will help you with your programming.

Since most of the examples here are short, you might like to try them as you come to them in the text. This will give you a better understanding of the examples. It will also give you a better *feel* for Extended BASIC.

## Multiple Statement Lines

One big difference between TI BASIC and Extended BASIC is the structure of the program lines. You can spot this right away. At a glance, a TI BASIC program has the appearance of

many short lines. TI BASIC allows only one program statement per line, and it requires that a statement line number follow the THEN and ELSE clauses of an IF statement. Extended BASIC takes a completely different approach. It allows more than one statement per line, and permits statements other than a line number after the THEN and ELSE clauses.

Let's see a few examples. The TI's random number generator generates numbers in the range from zero to one. This range can be divided into the following four quartiles:

```
0.00 <= N < 0.25
0.25 <= N < 0.5
0.50 <= N < 0.75
0.75 <= N
```

N stands for the random number. Programs 2-1, 2-2, and 2-3 will generate random numbers. Then the programs will determine to which quartile each number belongs. Counters will keep track of the number of random numbers that fall into each quartile. Each program will do this 500 times. You can expect to see the numbers more or less evenly distributed.

Program 2-1 shows the TI BASIC version of the program. Program 2-2 is the same program except that it is written in Extended BASIC. What are the differences? Most obviously, the Extended BASIC version is shorter—12 lines versus 21 lines. Look at line 120 in Program 2-2. Four variables, Q1 through Q4, are initialized to zero all in the same statement. This takes four lines with TI BASIC. Another striking difference can be seen in lines 150–170 of Program 2-2. Here, the counters are incremented right after the THEN clause. After incrementing the counters, control is transferred to the NEXT I statement by means of a GOTO. TI BASIC requires several more statements to do the same thing.

You may be wondering how the IF statement works. The statement or statements following the THEN clause are executed only when the condition is true. For example, consider line 150 of Program 2-2. There are two statements following the THEN clause:

```
Q1=Q1+1 :: GOTO 190
```

Both of these statements will be executed when N is less than .25.

Please note that Extended BASIC uses a double colon as the statement separator. Many other versions of BASIC use a

single colon for the same purpose. Be very careful when you are typing in your programs. This is especially true when you are translating programs into Extended BASIC.

Does the Extended BASIC program have any other benefits besides requiring fewer lines and therefore less memory? Well, a strong case can be made that the Extended BASIC is much easier to understand. There are fewer line number references and transfers of control. You can read the program from the beginning to the end without tracing through as many THENs and GOTOs.

### Program 2-1. TI Random Number Quartile

```
100 REM TI BASIC RANDOM NUMBER QUARTILE COUNTER
110 RANDOMIZE
120 Q1=0
130 Q2=0
140 Q3=0
150 Q4=0
160 FOR I=1 TO 500
170 N=RND
180 IF N<.25 THEN 270
190 IF N<.50 THEN 250
200 IF N<.75 THEN 230
210 Q4=Q4+1
220 GOTO 280
230 Q3=Q3+1
240 GOTO 280
250 Q2=Q2+1
260 GOTO 280
270 Q1=Q1+1
280 NEXT I
290 PRINT Q1;Q2;Q3;Q4
300 END
```

### Program 2-2. Extended BASIC Random Number Quartile Counter

```
100 REM EXTENDED BASIC RANDOM NUMBER QUARTILE COUN
    TER
110 RANDOMIZE
120 Q1,Q2,Q3,Q4=0
130 FOR I=1 TO 500
140 N=RND
150 IF N<.25 THEN Q1=Q1+1 :: GOTO 190
160 IF N<.5 THEN Q2=Q2+1 :: GOTO 190
170 IF N<.75 THEN Q3=Q3+1 :: GOTO 190
```

```
180 Q4=Q4+1
190 NEXT I
200 PRINT Q1;Q2;Q3;Q4
210 END
```

### Program 2-3. Another Extended BASIC Random Number Quartile Counter

```
100 REM ANOTHER EXTENDED BASIC RANDOM NUMBER QUART
    ILE COUNTER
110 RANDOMIZE
120 Q1,Q2,Q3,Q4=0
130 FOR I=1 TO 500
140 N=RND
150 IF N<.25 THEN Q1=Q1+1 ELSE IF N<.5 THEN Q2=Q2+
    1 ELSE IF N<.75 THEN Q3=Q3+1 ELSE Q4=Q4+1
160 NEXT I
170 PRINT Q1;Q2;Q3;Q4
180 END
```

### The IF-THEN Statement

Since the IF-THEN statement is used so often, it deserves special emphasis. The form of the IF-THEN statement that is used most often in this book is:

IF *relational expression* THEN *statement 1* ELSE *statement 2*

Refer to the Extended BASIC manual for a complete description of what is meant by a *relational expression.* In general, it is where a comparison is performed. Statement 1 is executed if the condition is true. Otherwise, statement 2, if present, is executed.

Statement 1 and statement 2 can consist of one or more statements. Even another IF statement will work. Look at Program 2-3. The number testing is done on one line—150. It is a series of IFs within IFs or, as they say in the business, nested IFs. What do you think of nested IFs? They did make the program shorter. Do you think it is easier or more difficult to read?

Another aspect of the IF-THEN statement is that tests or comparisons can be combined with the relational operators AND and OR. The following program fragment shows how a number can be checked against a lower and upper limit.

```
100 INPUT N
110 IF N<0 OR N>10 THEN 100
```

Line 110 contains two comparisons. If *either* one or both are true, the program will go back to line 100. Thus, this program fragment will accept numbers in the range from 0 through 10 inclusive.

When using the AND relational operator, *all* comparisons must be true in order for the expression to be true. Consider this statement:

```
100 IF ENERGY=0 AND SHIELDS=0 AND DAMAGE=100
THEN PRINT "KAPUT!"
```

Each of the three conditions must be met before the message is printed.

More complex conditions can be tested by combining AND and OR operators.

```
100 IF DISTANCE<20 AND (VELOCITY>=40 OR FUEL=0)
THEN PRINT "CRASHED"
```

This example shows how parentheses can help straighten things out. First, the tests on VELOCITY and FUEL are done. If VELOCITY is greater than or equal to 40 *or* FUEL equals 0, then the part within the parentheses is true. But in order for the message to be printed, not only must the part in parentheses be true, but DISTANCE must be less than 20 as well.

Let's look at just this one statement as it might appear in TI BASIC.

```
100 IF DISTANCE>=20 THEN 130
110 IF VELOCITY>=40 THEN 140
120 IF FUEL=0 THEN 140
130 REM condition not met
140 PRINT "CRASHED"
```

Again, this points out that Extended BASIC programs can be written with more clarity.

Now it's time for a little fun. Look at Program 2-4. It is called "Diplomatic Hi-Lo." The program is a variation on the familiar number guessing game. What's the variation? Well, sometimes the computer will lie or give misleading information. Sometimes it will even change the number in the middle of the game. Like a good diplomat, you've got a bit of negotiating ahead of you.

When you start the program, it will ask you for a truth percentage. This represents the percent of time that your TI will be honest. The screen will go dark when you type in this

number. This way, nobody can see what you're typing. You wouldn't try this game on unsuspecting folks, would you? It is surprising how difficult the game is even at an 85 or 90 percent truth level. By the way, the computer will never lie if and when you do guess the number.

On the practical side, the program illustrates the points covered so far. It uses multiple statement lines and various forms of the IF statement. The program is short enough for you to experiment with if you like.

Look at line 220 of Program 2-4. Note: The colons in the PRINT statements are used for skipping lines. They are not statement separators. Thus, : : (note the space in between) means skip two lines. Typing :: (no space between the colons) will cause a syntax error.

### Program 2-4. Diplomatic Hi-Lo

```
100 REM DIPLOMATIC HI-LO
110 RANDOMIZE
120 CALL CLEAR
130 PRINT "DIPLOMATIC HI-LO": :"CAN YOU OUTSMART M
    E?"
140 PRINT :: PRINT :: PRINT
150 PRINT "WHAT IS THE TRUTH PERCENTAGE"
160 PRINT :"ENTER A NUMBER FROM 0 TO 100"
170 PRINT
180 FOR I=1 TO 900 :: NEXT I
190 CALL SCREEN(2):: INPUT TRUTH :: CALL CLEAR ::
    CALL SCREEN(8)
200 IF TRUTH<0 OR TRUTH>100 THEN 120
210 TRUTHPCT=1-(TRUTH/100)
220 PRINT "I HAVE A NUMBER": :"IT IS BETWEEN 1 AND
     100": :"CAN YOU GUESS WHAT IT IS?"
230 REM GENERATE A NEW NUMBER
240 TRIES=0 :: NUMB=INT(100*RND+1):: PRINT
250 REM ACCEPT AND COMPARE GUESS
260 PRINT
270 INPUT "YOUR GUESS: ":GUESS
280 TRIES=TRIES+1
290 IF GUESS>NUMB THEN IF RND>TRUTHPCT THEN PRINT
    "  TOO HIGH" ELSE PRINT "  TOO LOW"
300 IF GUESS<NUMB THEN IF RND>+TRUTHPCT THEN PRINT
     "  TOO LOW" ELSE PRINT "  TOO HIGH"
310 IF GUESS=NUMB THEN PRINT :"THAT'S RIGHT!":"IT
    TOOK";TRIES;"TRIES" :: GOTO 360
```

```
320 IF TRIES>5 AND RND>0.7 THEN PRINT "  COME ON!
    THE NUMBER IS";INT(100*RND+1)
330 IF TRIES>10 AND RND>0.6 THEN BIAS=INT(10*RND+1
    +(20*(RND>0.5))):: PRINT "  THE NUMBER IS NEAR
    ";MIN(NUMB+BIAS,100)
340 IF TRIES>15 AND RND>.95 THEN PRINT "  I NOW HA
    VE A NEW NUMBER!" :: NUMB=INT(100*RND+1)
350 GOTO 250
360 PRINT :"PLAY AGAIN (Y OR N)";:: INPUT R$
370 IF R$="Y" THEN 230
380 CALL CLEAR :: PRINT "THANK YOU"
390 END
```

## Subprograms

The 99/4A contains several built-in subprograms. Some of the more familiar ones are COLOR, SCREEN, and SOUND. These subprograms are used with a CALL statement. The CALL statement requires one or more parameters:

100 CALL SOUND(1000,440,2)

In this case, the SOUND subprogram is given three parameters. Subprograms use the parameters in various ways. The three parameters above control the duration (1000 milliseconds), frequency (440 hertz), and volume (2) of a tone. The designers of the TI determined the way that SOUND would interpret the parameters.

The 99/4A does have its own set of built-in subprograms. They perform often-needed tasks related to controlling system features.

In the course of doing your own programming, you have probably found yourself writing some of the same code over and over again. Extended BASIC provides a very nice capability. You can write your own subprograms. And these subprograms can be written in Extended BASIC; you do not have to resort to machine language.

Subprograms form a special part of your program. Extended BASIC insists that all subprograms be placed at the end of the main program. Generally, subprograms will be written using high line numbers—25000 and above, for example. When the program is finished, the resequence command RES can tidy up the line number gaps.

You can build your own library of subprograms. If you wish, you can type the subprograms into each program as you

need them. On the other hand, disk users can save each subprogram in MERGE format so that subprograms can be merged right in with the other program lines. This method requires a bit of planning as to the use of line numbers. Be careful that subprograms use line numbers that are not already used by the main program. Also, make sure that the line numbers are such that the subprogram will be added at the end of the main program.

Program 2-5 is an example of a main program and a subprogram. The main program dimensions three arrays. The arrays are assigned arbitrary values. The contents of all three arrays are then printed. Next a CALL is done to the subprogram ARRAYINIT. This should alter the contents of array B. The three arrays are printed again. Indeed, array B was changed while arrays A and C were not.

Notice the relationship between the CALL statement and the SUB statement.

```
CALL ARRAYINIT(B(),5,0)
SUB ARRAYINIT(ARRAY(),DIMENSION,VALUE)
```

SUB defines the subprogram. The name of the subprogram is ARRAYINIT. It has three parameters since there are three items in the list following the subprogram name. The first parameter is a one-dimensional array. This is indicated by the name of the array and a left and right parentheses pair—ARRAY(). The next two parameters are numeric variables.

The CALL statement must agree with the subprogram definition in terms of the number and types of parameters. Notice that the names of the parameters don't match. They do not have to. Values are transferred according to a parameter's position in the parameter list. Here is how the values are transferred:

| | | |
|---|---|---|
| B() | to | ARRAY() |
| 5 | to | DIMENSION |
| 0 | to | VALUE |

How do values get back to the main program? The correspondence rules are the same. Notice that ARRAY appears in an assignment statement. VALUE is moved to elements of ARRAY. In the example, array B matches ARRAY. So, changes made to ARRAY actually show up in array B. This is borne out when we print array B after the CALL statement.

You will notice that the variable I is used in both the

main program and the subprogram. Believe it or not, these are really two separate variables. This is because any variables used in a subprogram are local variables. The subprogram knows about them, but the main program does not. Local variables are initialized to zero when a subprogram is first called. Thereafter, local variables retain their values even if the subprogram is called repeatedly.

Like most things, local variables have advantages and disadvantages. On the plus side, a subprogram cannot alter any variables in the main program even if they happen to have the same name. On the minus side, if you want a subprogram to alter such variables, they will have to be passed through the subprogram as parameters. This requires a bit of forethought and effort. However, the protection that is achieved from local variables far outweighs this minor disadvantage.

### Program 2-5. Array Initialization Subprogram

```
100 REM ARRAY INITIALIZATION SUBPROGRAM
110 CALL CLEAR
120 DIM A(5),B(5),C(5)
130 FOR I=1 TO 5 :: A(I),B(I),C(I)=I :: NEXT I
140 PRINT "BEFORE"
150 FOR I=1 TO 5 :: PRINT A(I);B(I);C(I):: NEXT I
160 CALL ARRAYINIT(B(),5,0)
170 PRINT "AFTER"
180 FOR I=1 TO 5 :: PRINT A(I);B(I);C(I):: NEXT I
190 END
200 SUB ARRAYINIT(ARRAY(),DIMENSION,VALUE)
210 FOR I=1 TO DIMENSION
220 ARRAY(I)=VALUE
230 NEXT I
240 SUBEND
```

### More Subprograms

Two other sample subprograms are included below. Each of the samples includes a main routine for testing purposes. If you would like to include these subprograms in your own programs, use the lines beginning with SUB and ending with SUBEND.

Program 2-6 illustrates how a string array is passed to a subprogram. Numeric arrays are handled the same as string arrays. The subprogram sorts the elements in the array in ascending order. A very simple, but slow, bubble sort

algorithm is used. Therefore, the subprogram is best suited for alphabetizing short lists.

Program 2-7 shows a very interesting get key subprogram. Many programs require at times a single keystroke reply. This can, for example, be a Y or N to signal if the player wants another game or not. Program 2-7 displays a message on row 20 of the screen. Next, a beep is sounded. TI's KEY subprogram checks for a pressed key. If a key is not pressed within a certain interval, the beep is sounded again.

Notice that there are two CALL KEY statements. The first calls key unit 3. This puts the keyboard in an uppercase only mode. So even if the ALPHA LOCK key is released, only uppercase characters can be typed. The call to key unit 5 puts the keyboard back into a full upper- and lowercase mode.

In general, these sample subprograms show how subprograms will be used in this book. There are some additional details, such as passing multidimensional arrays and call by value, which will not be covered. Consult the Extended BASIC manual.

### Program 2-6. String Sort Subprogram

```
100 REM STRING SORT SUBPROGRAM
110 DIM NAMES$(5)
120 FOR I=1 TO 5
130 INPUT "ENTER A NAME ":NAME$(I)
140 NEXT I
150 CALL CLEAR
160 PRINT "*** BEFORE ***"
170 FOR I=1 TO 5 :: PRINT NAME$(I):: NEXT I
180 CALL SORT(NAME$(),5)
190 PRINT "*** AFTER ***"
200 FOR I=1 TO 5 :: PRINT NAME$(I):: NEXT I
210 END
220 SUB SORT(LIST$(),N)
230 EXCHANGE=0
240 FOR I=1 TO N-1
250 IF LIST$(I)<=LIST$(I+1)THEN 300
260 T$=LIST$(I)
270 LIST$(I)=LIST$(I+1)
280 LIST$(I+1)=T$
290 EXCHANGE=1
300 NEXT I
310 IF EXCHANGE<>0 THEN 230
320 SUBEND
```

### Program 2-7. Get Key Subprogram

```
100 REM GET KEY SUBPROGRAM
110 CALL CLEAR
120 CALL GETKEY(R,"PRESS ANY KEY")
130 IF R<>13 THEN DISPLAY AT(1,1):CHR$(R):: GOTO 1
    20
140 END
150 SUB GETKEY(R,MSG$)
160 DISPLAY AT(20,1):MSG$
170 CALL SOUND(200,262,0):: K=0
180 CALL KEY(3,R,S):: IF S<>1 THEN K=K+1 :: IF K<2
    50 THEN 180 ELSE 170
190 CALL KEY(5,R1,S):: DISPLAY AT(20,1):" "
200 SUBEND
```

### Screen Formatting

Some of the most notable features of Extended BASIC are its screen formatting capabilities. 99/4A screen formatting is implemented with the DISPLAY and ACCEPT statements. These two statements have a broad application. They are ideal for any program that uses the video display for communicating.

You can think of the DISPLAY and ACCEPT statements as fancy versions of the PRINT and INPUT statements respectively. Unlike the PRINT and INPUT statements, DISPLAY and ACCEPT will not perform file input and output. Instead, DISPLAY and ACCEPT are specifically designed for the video display and keyboard.

The DISPLAY statement will print information at any location on the screen. Locations are identified by their row and column numbers. Since there are 24 rows, the row numbers range from 1 to 24. Column numbers range from 1 to 28. Row 1, column 1 is at the upper left corner of the screen.

More than one item may be displayed on any given line. DISPLAY has an option (the SIZE parameter) which can limit the number of characters shown. This prevents the DISPLAY from wiping out information that may already be on a line.

To get decimal points to line up nicely, employ the PRINT USING option. The PRINT USING option can reference the line number of an IMAGE statement. Or a format string may be included in the USING option itself. (For more information about PRINT USING and IMAGE statements,

refer to the manual that came with the cartridge, *TI Extended BASIC*, pages 97–100 and 150.)

Information may be read from any screen location by means of the ACCEPT statement. ACCEPT follows the same row and column addressing scheme as does DISPLAY. ACCEPT can also limit the number of characters typed in. The SIZE parameter performs this function.

The ACCEPT statement has an option called VALIDATE. VALIDATE performs editing tasks as the input field is typed in. You can specify a list of allowable characters. VALIDATE checks each key, as it is typed, to make sure that the character is valid. If not, VALIDATE will not accept the character. A tone will sound to alert the operator that an invalid key has been pressed. Imagine the programming that this saves.

Now let's see some of the screen formatting statements in action. Programs 2-8 and 2-9 represent two versions of a checkbook program. The processing is the same in each program. Program 2-8 uses conventional PRINT and INPUT statements. Program 2-9 shows what DISPLAY and ACCEPT can do.

The checkbook program is a simple one that helps you with checkbook arithmetic. As such, the program only does subtraction. Here is how it works. First, you type in the beginning balance. Then the program takes you through a series of steps. Type in a check amount. A new balance will be shown. Type in a negative number for deposits. Type in zero when you are finished.

When you run these two programs, you will notice some difference in how the screen is formatted. Program 2-8 scrolls the screen up every time it comes to a PRINT or INPUT statement. This is visually distracting even for this little program. DISPLAY and ACCEPT eliminate the scrolling. Program 2-9 has a cleaner overall operation. Now, try this. Type in some garbage, such as #$G!J, when you're supposed to type in a check amount. What does Program 2-8 do? How about Program 2-9? We used the VALIDATE option in Program 2-9, which will not accept such input.

### Program 2-8. Checkbook Adder with PRINT and INPUT

```
100 REM CHECK BOOK WITH PRINT AND INPUT
110 CALL CLEAR :: PRINT "BEGINNING BALANCE"
120 INPUT BALANCE
```

```
130 CALL CLEAR
140 INPUT "CHECK AMOUNT ":CHECK
150 IF CHECK=0 THEN 190
160 BALANCE=BALANCE-CHECK
170 PRINT :"NEW BALANCE ";BALANCE
180 PRINT :: GOTO 140
190 END
```

### Program 2-9. Checkbook Adder with DISPLAY and ACCEPT

```
100 REM CHECK BOOK WITH DISPLAY AND ACCEPT
110 DISPLAY AT(12,1)ERASE ALL:"BEGINNING BALANCE"
120 ACCEPT AT(14,1)VALIDATE(NUMERIC)BEEP:BALANCE
130 DISPLAY AT(12,1):"CHECK AMOUNT"
140 ACCEPT AT(14,1)VALIDATE(NUMERIC)BEEP:CHECK
150 IF CHECK=0 THEN 200
160 BALANCE=BALANCE-CHECK
170 DISPLAY AT(23,1):"NEW BALANCE"
180 DISPLAY AT(23,14):BALANCE
190 GOTO 130
200 END
```

There are additional features of the ACCEPT and DISPLAY. Data that may already be on the screen can be reused by the ACCEPT statement. You do not necessarily have to retype the data when the same data is repeated. Try out the short program shown below.

```
100 DATE$="00/00/00" :: CALL CLEAR
110 DISPLAY AT(2,1) SIZE(8):DATE$
120 ACCEPT AT(2,1) SIZE(-8) VALIDATE(DIGIT,"/
    ")BEEP:DATE$
130 DISPLAY AT(4,1):"EXPENSE AMOUNT"
140 ACCEPT AT(6,1)VALIDATE(NUMERIC):AMT
150 GOTO 110
```

This program fragment shows how you can reuse the value of DATE$. Line 110 displays the contents of DATE$. Line 120 reads DATE$ back in again. When you run the program, you'll notice that DATE$ is not erased from the screen. You could just hit ENTER. Whatever eight-character field was at row 2, column 1 was assigned to DATE$. This works because we specified a negative number in the ACCEPT's SIZE parameter. The negative sign tells ACCEPT not to erase the place on the screen before reading in another value.

Rather, ACCEPT reads what's there already unless, of course, you type in something different.

This is a very handy feature. You will often need to type in lots of data. Some information changes less frequently than the rest. Suppose you're entering expenses: Type in an expense date first, then the category, and then the amount. Generally, you'll have more than one expense per date. Why type the date over each time? Why introduce the risk of data entry errors? When designing your own programs, consider the techniques that have just been illustrated.

Many times when you're displaying numeric values, it's necessary to define the number of digits before or after the decimal point. Figures dealing with money should have two digits after the decimal point. In other situations, the number of digits just depends on the particular computations.

The USING option can be included in DISPLAY statements. Although the USING option is most frequently used with numeric variables, it can also be used with character strings. In the latter case, it just limits the number of characters that will be shown much as the SIZE option does.

Program 2-10 is a miles per gallon calculation program. It shows the computed results two ways. First, it shows the miles per gallon without a USING statement. Then it displays the miles per gallon with a USING statement. The particular format used is ##.##. This means that there can be a maximum of two digits before the decimal point and two digits after the decimal point. Experiment with this format. See what effects it has. You will notice that fractional numbers are rounded rather than simply truncated.

### Program 2-10. Miles Per Gallon

```
100 REM MILES PER GALLON
110 DISPLAY AT(2,1)ERASE ALL:"BEGINNING MILE READI
    NG"
120 ACCEPT AT(4,1)VALIDATE(NUMERIC)BEEP:BEGINMILES
130 DISPLAY AT(2,1):"MILE READING"
140 ACCEPT AT(4,1)VALIDATE(NUMERIC)BEEP:ENDMILES
150 IF ENDMILES=0 THEN 250
160 DISPLAY AT(6,1):"GALLONS USED"
170 ACCEPT AT(8,1)VALIDATE(NUMERIC)BEEP:GALLONS
180 DISTANCE=ENDMILES-BEGINMILES
190 MPG=DISTANCE/GALLONS
200 BEGINMILES=ENDMILES
210 DISPLAY AT(15,1):"MPG"
```

```
220 DISPLAY AT(17,1):MPG
230 DISPLAY AT(19,1):USING "##.##":MPG
240 GOTO 130
250 END
```

## Error Trapping

Extended BASIC provides three statements that are used for intercepting various conditions that may arise when a program is running:

- ON BREAK,
- ON WARNING, and
- ON ERROR.

Look at your keyboard for a minute. The CLEAR key is right next to the ERASE key. The ERASE key is handy for erasing a field from the screen prior to keying in the new information. Have you ever hit the CLEAR key by mistake? (To press the CLEAR key, you must hold down the FCTN key and the 4 key at the same time.) CLEAR interrupts and halts a BASIC program. This is nice for debugging purposes, but a real inconvenience when you're running an application.

Try the following small program:

```
100 GOTO 100
```

Of course, this program does nothing except run in circles. After you tire of watching the program, press CLEAR (FCTN-4). The program stops, as you would expect.

Now, try this variation:

```
100 ON BREAK NEXT
110 GOTO 110
```

Start up the program. Press CLEAR (FCTN-4). Nothing happens. The CLEAR key has been disabled. There are only two ways of stopping the program now. You can press QUIT (FCTN-=) or turn the power off. In either case, the program will be lost.

The CLEAR key can be enabled at a strategic point in the program. Try this:

```
100 ON BREAK NEXT
110 COUNT=COUNT+1
120 IF COUNT=200 THEN ON BREAK STOP
130 GOTO 110
```

This program cannot be stopped until its critical processing is finished. Once that is done, the program will let itself be interrupted. The ON BREAK STOP enables the CLEAR key again.

Most of the programs in this book use the ON BREAK NEXT statement at the beginning of the program. The main reason for this is to make sure the program keeps running even if CLEAR is pushed. These programs also give you a way of returning to Extended BASIC short of pulling the plug.

In the section on screen formatting, we looked at the VALIDATE option of the ACCEPT statement. VALIDATE checks characters as they are typed in. Here is another experiment.

```
100 CALL CLEAR
110 ACCEPT AT(12,1)VALIDATE(NUMERIC):NUMB
120 DISPLAY AT(14,1):NUMB
130 GOTO 110
```

When you run this program, you will not be able to key in nonnumeric characters. Try just hitting ENTER. A warning message indicating an input error appears on the screen. Type 123.45.67. It is obvious that this is not a valid number because it has two decimal points. However, this type of error is not caught by VALIDATE. Instead, the TI displays a warning message. This message indicates that a string-number mismatch has occurred. In the case of either of these errors, your program keeps running and you can type in the number again. However, the warning messages have probably made a mess of your nicely formatted screen.

Here is a way around this.

```
100 CALL CLEAR
110 ON WARNING NEXT
120 ACCEPT AT(12,1)VALIDATE(NUMERIC):NUMB
130 DISPLAY AT(14,1):NUMB
140 GOTO 120
```

Run this program. Type ENTER, and then try typing 123.45.67. The warning messages are gone. The cursor stays right at the input field. The input is not accepted until a valid number is typed.

Try entering this number—123.E7. Notice that the VALIDATE(NUMERIC) will accept scientific notation.

There are two other variations of the ON WARNING

statement. ON WARNING PRINT will cause the warning messages to be printed as they normally are. ON WARNING STOP will make the program stop when a warning condition arises.

Sometimes serious errors can crop up when a program is running. An example of this is giving the VAL function a non-numeric argument. Extended BASIC stops the program when errors occur. There is a way to prevent Extended BASIC from stopping the program when such an error occurs.

ON ERROR line number

When an error occurs, control will be passed to the given line number. The program can analyze the error with the CALL ERR subprogram. Where possible, corrective action can be taken.

The error-handling subroutine must issue a RETURN in order for the program to resume. After an error is detected, Extended BASIC defaults to ON ERROR STOP. If you want the program to continue intercepting errors, another ON ERROR statement must be executed.

ON BREAK and ON WARNING can improve the operation of a program. They eliminate the effects of common warning conditions, and they make a program easier to use because the program becomes very forgiving of common data entry problems. If you want to add ON BREAK and ON WARNING to a program it is best to add them *after* you are completely satisfied that the program is working as it should. ON BREAK and ON WARNING remove valuable debugging capabilities and information.

Even though ON BREAK and ON WARNING appear in the program listings in this book, it is best to enter them only after you are sure the program runs properly.

## Program Design

When you're designing a program, the first thing you must decide is what specific results you're aiming for. The result of a program is its *output*. The outputs may be a printed report, a display screen, or even a file on cassette tape. Every output has some particular arrangement of data.

By analyzing the output, a program design can be developed. If a program achieves the desired output, the program works. The desired output is your objective when you design

a program. If the design of the program achieves these objectives, you know the design is correct. Once the actual program does what you aimed for, you know it's finished.

You may be wondering about *input* to the program. Well, input comes about as a natural result of the design. Only when you know what the program must do, will you know what inputs are necessary.

This way of thinking is not so far-fetched. Consider the task of building a house. What style of house do we want? Will it be a split-level or a colonial? Will it have an attached garage? Once these overall decisions are made, the architect can concentrate on the finer details such as the number of rooms, the layout of the kitchen, and so on. Eventually a point is reached when the lumber and nails can be ordered.

Imagine building a house without a plan. One day a truck shows up with a pile of lumber and a tin of nails. The carpenters begin their work and complete the house. Will anyone be pleased with the results? Since the desired results were not established beforehand, how will anyone know whether the builders built what was desired?

A floor plan or blueprint documents the internal structure of the house. A flowchart does the same thing for programs. Both types of documents are prepared as part of the design process.

Figure 2-1 shows a sample flowchart. Each box on the chart represents a group of related program statements. The statements are related in that they are working together to perform one particular function. In practice, the statements may actually be a subroutine or a subprogram.

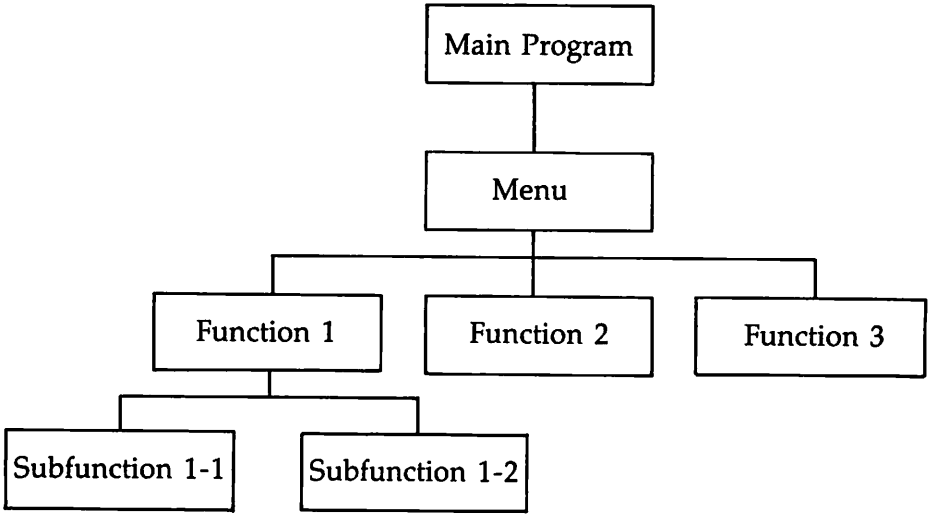
The flowchart in Figure 2-1 is typical of the programs in this book. The program begins by dimensioning arrays and intializing variables. Many programs are menu-driven. This means that the program gives the user a choice of functions that can be performed. A box on the chart shows the menu display.

The person using the program can choose any one of the functions from the menu screen. Thus, several boxes are shown below the menu box. These boxes represent the various functions. Some of these functions may be very simple and self-contained. Other functions, however, may be more involved. Perhaps they require subfunctions. If so, the subfunctions are shown as a box or boxes below the function.

So the flowchart performs a function similar to the floor plan: The floor plan shows the interior architecture of a house, and the flowchart shows the interior architecture of a program. Neither the floor plan nor the flowchart shows every little detail.

Since menu screens are used so often, a sample menu subprogram has been included below (Program 2-11). The menu subprogram is a very general one. The title of the menu and the list and number of functions are passed as parameters. The menu program gives you back a function code. The function code is the number of the selection that was chosen. The function code will be set to zero if the operator typed ENTER instead of a selection number.

**Figure 2-1. Sample Program Flowchart**




**Program 2-11. Menu Subprograms**

```
100 REM MENU SUBPROGRAM
110 DIM SELECT$(5)
120 DATA 3
130 DATA "FUNCTION 1","FUNCTION 2","FUNCTION 3"
140 RESTORE 120
150 READ N :: FOR I=1 TO N :: READ SELECT$(I):: NE
    XT I
160 CALL MENU("MAIN MENU",SELECT$(),N,FUNC)
170 IF FUNC=0 THEN 190 ELSE IF FUNC=1 THEN GOSUB 2
    00
```

```
180 GOTO 140
190 END
200 REM SUB FUNCTION
210 DATA 2
220 DATA "SUB FUNCTION 1","SUB FUNCTION 2"
230 RESTORE 210
240 READ N :: FOR I=1 TO N :: READ SELECT$(I):: NE
    XT I
250 CALL MENU("SUB FUNCTION MENU",SELECT$(),N,FUNC
    )
260 RETURN
270 SUB MENU(TITLE$,CHOICE$(),N,FUNC)
280 C=(28-LEN(TITLE$))/2
290 DISPLAY AT(1,C)ERASE ALL:TITLE$
300 DISPLAY AT(4,1):"DO YOU WANT:"
310 R=6 :: FOR I=1 TO N
320 DISPLAY AT(R,1):CHOICE$(I):: R=R+2
330 NEXT I
340 R=R+1
350 DISPLAY AT(R,1)BEEP:"TYPE YOUR SELECTION -> "
360 CALL KEY(0,R2,S):: IF S<>1 THEN 360
370 IF R2=13 THEN FUNC=0 :: GOTO 400
380 IF R2<49 OR R2>48+N THEN 350
390 FUNC=R2-48 :: DISPLAY AT(R,24)SIZE(1):CHR$(R2)
400 SUBEND
```

# Chapter 3

# File Management

# File Management

File Management applications can be developed for a system that consists of the 99/4A console, a TV or video monitor, and a tape recorder for program storage. Such applications accept input from the keyboard, do some calculations, and then display the results. The gas mileage program in the last chapter is a good example.

However, the more useful home applications take advantage of the computer's ability to store data and subsequently retrieve it. In this way, the tedious task of typing in data every time it is needed is avoided. Moreover, a permanent record of important information can be retained.

The 99/4A can use a variety of hardware devices for data storage. A cassette recorder or disk drive is the device most commonly used for data storage. In addition, a printed report can also be thought of as a storage medium. The main limitation of a printed report is that the computer cannot automatically read a report—at least not yet. So a printed report works in only one direction. It is a *write only* storage medium.

This chapter will look at ways that you can use Extended BASIC and the 99/4A for data storage and retrieval, or *data management* for short. The data management capabilities of TI BASIC and Extended BASIC are almost identical. Extended BASIC adds only a few new statements to the 99/4A's file management repertoire. So the basis of what is covered in this chapter can be found in the TI BASIC manual.

## Fields, Records, and Files

Before discussing the details, let's briefly define these three important terms.

**Field.** A field is an item of data which cannot be further subdivided. In a program, a field will consist of a numeric or string variable. Examples of fields are expense category and expense amount. Sometimes the term *data element* is used in place of field.

**Record.** A record is a group of related fields. Generally, a record is created by a PRINT statement. An INPUT statement reads the record. A record may also be viewed as that unit of

information transferred from or to the computer with an output or input operation.

**File.** A file is a collection of records. Typically, a file is stored on an external device such as a disk. Before any records can be read from or written to the file, the file must be opened with an OPEN statement. This establishes the software connection between the program and the device. Similarly, a file must be closed when the program is finished with it. The CLOSE statement does this. It breaks the connection between the device and the program.

### Program Files

The simplest kinds of files on the TI are program files. A program file is created with a SAVE statement:

```
SAVE CS1
or
SAVE DSK1.PROGRAM
```

The SAVE command writes a copy of the program to the specified output device. The program remains intact in memory. The 99/4A takes care of opening and closing the output device.

The OLD command reads a program from the specified device into the console's memory. The program is then set to run. Again, there is no need for an OPEN or CLOSE statement.

Program files are stored in a special format that makes sense only to the 99/4A. It is difficult to print program files.

The LIST command converts a program back into a format that is easily understood. The LIST command can be used as a way of sending the program to an external device:

```
LIST "RS232.BA=1200"
LIST "CS1"
LIST "DSK1.PGMLIST"
```

The first LIST sends the program to the RS232 device. If there is a printer attached to the RS232 device, the program in memory will be printed. The second and third LISTs make a permanent copy of the program on magnetic media. In all three cases, the program has been converted to ASCII (DISPLAY) format. This is the form of the program that is easily understood by people.

Why LIST (rather than SAVE) a program on tape or

diskette? A LISTed program file can be mainpulated as though it were an ordinary data file. For example, a LISTed program can be accessed by a word processing program such as TI-Writer. Thus, a program listing can be included as part of a document.

The RUN statement can also be used with program files. Extended BASIC allows a program to be RUN from another program. This is very handy when working with programs that will not fit in memory all at once. Unfortunately, one program's variables cannot easily be passed to another program. The variables must be passed by means of a data file.

Here is an example of this.

```
100 REM PROGRAM-A
110 PRINT "IN PROGRAM-A"
120 RUN "DSK1.PROGRAM-B"
```

```
100 REM PROGRAM-B
110 PRINT "IN PROGRAM-B"
120 RUN "DSK1.PROGRAM-C"
```

```
110 REM PROGRAM-C
110 PRINT "IN PROGRAM-C"
120 RUN "DSK1.PROGRAM-A"
```

There are three programs—PROGRAM-A, PROGRAM-B, and PROGRAM-C. Save each of them separately on disk. Then run PROGRAM-A. It will start up PROGRAM-B which will start up PROGRAM-C which will start up PROGRAM-A. This will continue until you stop it with a FCTN-4 (CLEAR) or FCTN-= (QUIT).

You can, under program control, run a program from tape as well.

```
100 RUN "CS1"
```

When line 100 is executed in a program, the familiar tape load instructions will appear on the screen. Position the tape to the program that you want. Then follow the rest of the instructions.

### Data Files

The 99/4A's built-in data management software will work with a variety of devices. The data management facilities also offer several different ways of organizing files and records. All these features add up to a very flexible data management system.

# File Management

You can set up data files for practically any kind of home application.

One disadvantage, though, of such flexibility is that there are so many options that it is difficult to know where to begin. The rest of this section will be devoted to several aspects of data management and will present programs that illustrate file management techniques.

The OPEN statement in a TI BASIC or Extended BASIC program establishes a link between a program and a file. Many of the characteristics of the file are specified by means of the OPEN statement.

**File organization.** TI files may be either SEQUENTIAL or RELATIVE. Records in a sequential file are always accessed one after another; therefore, they must be read in order. If you are interested only in the twenty-first record, you still must read records 1–20 first. Records in a relative file may be accessed directly. If you want the twenty-first record, you can retrieve it alone. There is no need to read through preceding records.

**File type.** Records may be stored in a file that is either in binary or ASCII format. An INTERNAL file is one in which all the records are in binary format. Since the records are in binary format, they are readily understood by the computer. Thus, INTERNAL files may be processed a little more quickly. A DISPLAY file is a file that contains records that are in ASCII format. This is the format that is readable by people. DISPLAY format files are suitable for printing or for being displayed on a TV.

**Open mode.** There are four ways of treating a file.

- INPUT. If it is necessary only to read records from a file, the file should be open with INPUT. This tells the computer that the file will not be changed.
- OUTPUT. An OUTPUT file is used to write records. There will be no reading or changing of records. Generally, a file is opened as OUTPUT when it is first created.
- APPEND. Opening a file as APPEND allows you to add records at the end of an existing file. Records at the front of the file cannot be accessed in any way in the APPEND mode.
- UPDATE. In the UPDATE mode, records can be read, changed, and written. The type of file processing intended will determine the open mode that should be used.

**Record type.** If all the records in a file are the same length, the file is said to consist of FIXED length records. On the other hand, the records may be of different lengths. The file then consists of VARIABLE length records.

Variable length records generally use storage space more efficiently. Only the exact amount of storage needed is used. Sometimes fixed length records are padded with extra characters, thereby taking up some additional storage space. However, relative files must contain fixed length records.

## Storage Devices

These are the different options that can be specified for a file in the OPEN statement. Not all devices will work with all the open options.

**Disk.** All of the options can be used with the disk system. The choice of options really depends on the design of any given application. If you are writing files that will be used by another program, you must write the file in a format that the program will accept. For example, the Editor/Assembler expects records in DISPLAY (ASCII) format in either fixed or variable length.

**Cassette.** Cassette files have a narrower range of available options. Cassette files must be SEQUENTIAL. They can be opened for either INPUT or OUTPUT. The record length must be FIXED. Furthermore, there are only three choices for the record length. The record length must be 64, 128, or 192.

**RS232.** The only restriction for files using the RS232 interface is that they must be SEQUENTIAL. All the other options are available. Remember though, the device that you connect to the RS232 interface may impose some additional restrictions. For example, a printer should be opened with a file type of DISPLAY in the OUTPUT mode. Since each printed line may contain a different number of characters, printers usually accept variable length records.

The OPEN statement serves another important purpose in addition to establishing the file characteristics. The OPEN statement defines the file number and filename. The file is made known to the program by its file number, a number from 1 to 255. The PRINT, INPUT, and CLOSE statements all refer to the file by file number rather than filename. This has a very important benefit. Suppose a program's function is to print a report. Say the output file is assigned file number 10.

Later on, only the OPEN statement needs to be changed in order that file number 10 be written to a disk. No other line of the program will require change. The program can write to different output devices just by changing the OPEN statement.

The filename actually consists of the device name, which may or may not be followed by the name of the file. All devices have a TI-assigned device name such as CS1 or DSK1. Only disk files have the name of the file after the device name—DSK1.PAYROLL, for example. Cassette and RS232 files simply use the device name.

### Sequential File Examples

Sequential files are perfectly suited for processing lists of information. A list is first made by adding items one at a time to the end of the list. Later on, it may be alphabetized or some items may be changed. In any case, the entire list is processed from the beginning to the end. A good example is a name and address list. It contains entries for relatives and friends. Each entry will have the person's name, address, and perhaps birthday.

Programs 3-1 and 3-2 show how the 99/4A can be used for maintaining a computerized name and address list. Program 3-1 builds the name and address file. The program is a data entry program. It is designed so that it can be used to help you convert your manual records to computer records. After you have built the initial computer file, you will no longer need Program 3-1. However, people have been known to change their addresses from time to time. You would certainly like to keep your name and address list up-to-date. Program 3-2 is an update program that can be used for correcting entries. It can also add new entries to the file.

These two programs treat the name and address list as a SEQUENTIAL file. The file is stored in INTERNAL format. Each record is organized as follows:

| Item | Maximum Length |
|---|---|
| Record Number | 9 |
| Name | 26 |
| Address Line 1 | 26 |
| Address Line 2 | 26 |
| Address Line 3 | 26 |
| Birthday | 9 |
| | 122 |

Included in each of the field lengths is the one-byte length code that TI adds for INTERNAL format files. For example, the maximum length of a name is 25 characters. The SIZE option on the ACCEPT statement establishes this limit. Then, one byte is added for the length code. Thus, the name field can actually be up to 26 characters long.

Programs 3-1 and 3-2 use cassette storage. This forces us to use FIXED length records. The record length of 128 is specified in the OPEN statements. That is the best fit that can be obtained given a maximum record length of 122. The file management system will add a minimum of six pad characters. This is done in order to bring the actual record length to 128.

What happens if you don't type in all of these items? Suppose you leave out two address lines and a birthday for a particular person. Yes, the actual data will be less than 128 characters. The 99/4A will automatically make up the difference by adding pad characters so that the record length comes up to 128.

If Programs 3-1 and 3-2 had been written for disk operation, we could have used VARIABLE length records. In that case, there would be no padding of the records. The actual record lengths would be used. The space used on a diskette, then, would be strictly that required by the data.

Program 3-1 builds the initial name and address file. At the beginning of the program, an OPEN statement establishes file number 1 as an output file. The individual data items are read from the screen. Then they are written to the output file with a PRINT # statement:

```
PRINT #1 :
RECNO;NAME#;ADDR1$;ADDR2$;ADDR3$;BIRTH$
```

This single PRINT # statement writes 128 characters of data to the cassette output device.

This process of reading from the screen and writing to tape continues until the file has been built. Notice that the last record written is a special one:

```
99999;"ZZZ";"ZZZ";"ZZZ";"ZZZ";"ZZZ"
```

This record is a marker. It marks the end of the file. There are no more records after this one.

When using this program, type in the information for

each field. Once you've finished one complete entry, the program will recycle and you can enter another name. When you come to the end of your list and have no more names to enter, simply type END instead of a name and press RETURN.

Program 3-2 is a little more involved. It begins by reading each record from the file. The information is displayed on the screen. You have the option of changing, deleting, or keeping the information. If you decide to keep the record, the fields are stored in arrays. Then the next record is read. This record-by-record process continues until the entire file has been read. Note the check for the end-of-file marker.

At this point, you have the option of adding more entries to the file. If you choose to do so, the information will be read from the screen and also stored in the arrays in memory.

When you are all finished, the arrays will be written back to tape. This gives you an updated version of the name and address file.

Program 3-2 uses file number 1 as both an input and an output file. This is accomplished by first using file number 1 as an input file. The file is opened, the records are read, and then the file is closed. File number 1 can only be used as an output file after it is finished being used as an input file. Notice how Program 3-2 reads an input record:

```
INPUT #1:R,NAME$;ADDR1$;ADDR2$;ADDR3$;BIRTH$
```

The six variables are read in with the INPUT # statement. The variable list corresponds with the number and type of variables that were used in the PRINT # statement that created the file. This is necessary in order for the program to work properly. Different variable names can be used in the INPUT # statement, but the variable types must agree.

These two programs cover the fundamentals of sequential file processing. You can use these programs as the starting point for your own name and address system. Maybe these programs are fine the way they are. Perhaps you would like to try your hand at manipulating the name and address file. Here are some ideas. Can you write a program that will alphabetize the names? What do you think about printing mailing labels from the files? Do you have a need for expanding the record to include wedding anniversary dates, for example?

## Program 3-1. Build Name List

```
100 REM BUILD NAME LIST
110 RECNO=0
120 CALL CLEAR
130 OPEN #1:"CS1",OUTPUT,INTERNAL,FIXED 128
140 DISPLAY AT(1,8)ERASE ALL:"BUILD NAME LIST"
150 DISPLAY AT(3,1):"NAME":">"
160 DISPLAY AT(7,1):"ADDRESS":">":">":">"
170 DISPLAY AT(13,1):"BIRTHDAY":">"
180 DISPLAY AT(20,1):"RECORD NUMBER"
190 DISPLAY AT(22,1):"O.K. (Y OR N)?"
200 ACCEPT AT(4,4)SIZE(-25)BEEP:NAME$ :: IF NAME$=
    "" THEN 200
210 IF NAME$="END" OR NAME$="end" THEN 340
220 ACCEPT AT(8,4)SIZE(-25)BEEP:ADDR1$
230 ACCEPT AT(9,4)SIZE(-25)BEEP:ADDR2$
240 ACCEPT AT(10,4)SIZE(-25)BEEP:ADDR3$
250 ACCEPT AT(14,4)SIZE(-8)BEEP:BIRTH$
260 DISPLAY AT(20,15):RECNO
270 ACCEPT AT(22,16)SIZE(1)VALIDATE("YN")BEEP:R$ :
    : IF R$="" THEN 270
280 IF R$="N" THEN 200
290 DISPLAY AT(24,1)BEEP:"WORKING ..."
300 PRINT #1:RECNO;NAME$;ADDR1$;ADDR2$;ADDR3$;BIRT
    H$
310 RECNO=RECNO+1
320 DISPLAY AT(24,1):" "
330 GOTO 200
340 PRINT #1:99999;"ZZZ";"ZZZ";"ZZZ";"ZZZ";"ZZZ"
350 CALL CLEAR :: CLOSE #1 :: CALL CLEAR
360 END
```

## Program 3-2. Update Name List

```
100 REM UPDATE NAME LIST
110 MAXN=50 :: RECNO,P=0
120 DIM NAMES$(50),ADDRS$(50,2),BDAYS$(50)
130 CALL CLEAR :: PRINT "** INSERT INPUT TAPE **"
140 OPEN #1:"CS1",INPUT ,INTERNAL,FIXED 128
150 DISPLAY AT(1,7)ERASE ALL:"UPDATE NAME LIST"
160 DISPLAY AT(3,1):"NAME":">"
170 DISPLAY AT(6,1):"ADDRESS":">":">":">"
180 DISPLAY AT(11,1):"BIRTHDAY":">"
190 DISPLAY AT(15,1):"PRESS:"
200 DISPLAY AT(17,2):"C - CHANGE ENTRY"
210 DISPLAY AT(18,2):"D - DELETE ENTRY"
220 DISPLAY AT(19,2):"K - KEEP ENTRY"
230 DISPLAY AT(22,1):"RECORD NUMBER"
```

```
240 REM LOAD ARRAY
250 DISPLAY AT(22,15):RECNO
260 DISPLAY AT(24,1)BEEP:"WORKING ..."
270 INPUT #1:R,NAME$,ADDR1$,ADDR2$,ADDR3$,BIRTH$
280 DISPLAY AT(24,1):" "
290 IF R=99999 THEN 440
300 REM DISPLAY RECORD
310 DISPLAY AT(4,4):NAME$
320 DISPLAY AT(7,4):ADDR1$
330 DISPLAY AT(8,4):ADDR2$
340 DISPLAY AT(9,4):ADDR3$
350 DISPLAY AT(12,4):BIRTH$
360 RECNO=RECNO+1
370 REM GET UPDATE OPTION
380 ACCEPT AT(15,8)SIZE(1)VALIDATE("CDK")BEEP:R$
390 IF R$="" THEN 380
400 IF R$="D" THEN 240
410 IF R$="C" THEN GOSUB 780 :: GOTO 370
420 GOSUB 690
430 GOTO 240
440 REM END OF INPUT - ASK FOR ADDS
450 DISPLAY AT(4,4):" " :: DISPLAY AT(7,4):" " ::
    DISPLAY AT(8,4):" "
460 DISPLAY AT(9,4):" " :: DISPLAY AT(12,4):" "
470 FOR I=15 TO 24 :: DISPLAY AT(I,1):" " :: NEXT
    I
480 DISPLAY AT(15,1):"ADD RECORDS (Y OR N)?"
490 ACCEPT AT(15,23)SIZE(1)VALIDATE("YN")BEEP:R$ :
    : IF R$="" THEN 490
500 IF R$="N" THEN 580
510 REM ADD RECORDS
520 DISPLAY AT(22,1):"O.K. (Y OR N)?"
530 GOSUB 780 :: IF NAME$="END" OR NAME$="end" THE
    N 580
540 ACCEPT AT(22,16)SIZE(1)VALIDATE("YN")BEEP:R$ :
    : IF R$="" THEN 540
550 IF R$="N" THEN 530
560 GOSUB 690 !STORE ENTRIES
570 GOTO 510
580 REM WRITE TO TAPE
590 CALL CLEAR :: PRINT "** REMOVE INPUT TAPE **"
    :: CLOSE #1
600 CALL CLEAR :: PRINT "** INSERT OUTPUT TAPE **"
     :: OPEN #1:"CS1",OUTPUT,INTERNAL,FIXED 128
610 DISPLAY AT(12,1)ERASE ALL:"WRITING RECORD "
620 FOR I=0 TO P-1
630 DISPLAY AT(12,16):I
640 PRINT #1:I;NAMES$(I);ADDRS$(I,0);ADDRS$(I,1);A
    DDRS$(I,2);BDAYS$(I)
```

```
650 NEXT I
660 PRINT #1:99999;"ZZZ";"ZZZ";"ZZZ";"ZZZ";"ZZZ"
670 CLOSE #1 :: CALL CLEAR
680 END
690 REM STORE ENTRIES
700 IF P>MAXN THEN DISPLAY AT(24,1):"TABLE FULL" :
    : GOTO 770
710 NAMES$(P)=NAME$
720 ADDRS$(P,0)=ADDR1$
730 ADDRS$(P,1)=ADDR2$
740 ADDRS$(P,2)=ADDR3$
750 BDAYS$(P)=BIRTH$
760 P=P+1
770 RETURN
780 REM READ SCREEN
790 ACCEPT AT(4,4)SIZE(-25)BEEP:NAME$ :: IF NAME$=
    "" THEN 790
800 IF NAME$="END" OR NAME$="end" THEN 850
810 ACCEPT AT(7,4)SIZE(-25)BEEP:ADDR1$
820 ACCEPT AT(8,4)SIZE(-25)BEEP:ADDR2$
830 ACCEPT AT(9,4)SIZE(-25)BEEP:ADDR3$
840 ACCEPT AT(12,4)SIZE(-8)BEEP:BIRTH$
850 RETURN
```

### Relative File Examples

The RELATIVE type of file organization will only work with a disk system. Each record in a RELATIVE file must be of FIXED length. Records in the file are accessed by record number. The record number may range from 0 to 32767. Thus, there can be a maximum of 32768 records in a RELATIVE file.

The OPEN statement for a relative file will specify RELATIVE as the file organization. The REC clause of the INPUT # and PRINT # statements specifies the number of the record to be accessed. The REC clause may contain an arithmetic expression. Therefore, it is possible to compute a record number according to some formula. The desired record will then be read or written. Sometimes the phrase *random access* is used to describe this method of input and output. This is because the records can be accessed in any, hence random, order.

Even though the OPEN statement says RELATIVE, records may also be accessed sequentially. When the file is opened, the record number is set to zero. An INPUT # or PRINT # without a REC clause will access the record and

then increment the record number. Thus, you can treat a RELATIVE file as though it were a sequential file.

You can modify the name and address list programs so that they will work with RELATIVE files. Program 3-3 uses the name and address cassette file as input. The program writes the records to a RELATIVE file called NAMEADDR.

The PRINT # statement uses the REC clause. The data records are written beginning with record number one. The program adds one to the record number after each PRINT #. Notice that the record number is also included in the PRINT # statement as part of the data. This is just good practice. It gives us a way of double-checking INPUT # statements. To make sure the proper record was read, compare the record's record number to the record number in the REC clause.

Program 3-3 treats record zero differently. The program puts the actual number of records in the file in record zero. This is a good technique. You can test a computed record number against the number of records that you know are in the file. If the computer record number is larger, that may indicate a problem with the calculation.

Program 3-4 shows how to update the information in the file NAMEADDR. The program begins by reading record zero and obtaining the number of records in the file. Next, the program begins the update cycle. The program asks for a record number. The record number is tested to see if it is within the proper limits. If it is, the record is retrieved and displayed. You can either change it or leave it the way it is. The process continues until you type Q, indicating that you are ready to quit.

Program 3-4 does not have a delete function. If you want to remove a name from the list, just blank it out with the ERASE key and use the change function. The particular record will still be on the file. It can, however, be used for another name.

Look at the way records are added to the file. First, one must be added to the number of records on the file. This updated count will be used in the REC clause of a PRINT # statement. Thus, the new record is placed at the end of the file.

When the update process is finished, record zero must be rewritten to the file because records may have been added. The record counter must agree with the number of records

on the file. Otherwise, the check on the record number may prevent us from retrieving the new records.

RELATIVE files have several advantages. As Program 3-4 shows, you need to retrieve only those records that must be updated. You do not have to look at each record on the file, as you did in Program 3-2. If you have a disk system and have tried this program, you probably noticed how quickly records can be retrieved and updated. This is another characteristic of RELATIVE files.

There is a disadvantage, too. Suppose you had 100 entries on file. How could you ever remember which record number was associated with what person? Periodically, you would need to print a list of names and record numbers.

### Program 3-3. Load Relative File

```
100 REM LOAD RELATIVE FILE
110 RECNO=1
120 CALL CLEAR
130 OPEN #1:"CS1",INPUT ,INTERNAL,FIXED 128
140 OPEN #2:"DSK1.NAMEADDR",RELATIVE,OUTPUT,INTERN
    AL,FIXED 128
150 DISPLAY AT(12,1)ERASE ALL:"RECORD NUMBER "
160 INPUT #1:R,NAME$,ADDR1$,ADDR2$,ADDR3$,BIRTH$
170 IF R=99999 THEN 220
180 DISPLAY AT(12,15):RECNO
190 PRINT #2,REC RECNO:RECNO;NAME$;ADDR1$;ADDR2$;A
    DDR3$;BIRTH$
200 RECNO=RECNO+1
210 GOTO 160
220 PRINT #2,REC 0:0;STR$(RECNO-1);" ";" ";" ";" "
230 CALL CLEAR
240 CLOSE #2
250 CLOSE #1
260 CALL CLEAR
270 END
```

### Program 3-4. Update Relative File

```
100 REM UPDATE RELATIVE FILE
110 CALL CLEAR :: ON WARNING NEXT
120 OPEN #1:"DSK1.NAMEADDR",RELATIVE,INTERNAL,UPDA
    TE,FIXED 128
130 INPUT #1,REC 0:R,N$,T1$,T2$,T3$,T4$
140 MAXREC=VAL(N$)
150 DISPLAY AT(1,7)ERASE ALL:"UPDATE NAME LIST"
160 DISPLAY AT(3,1):"NAME":">"
```

```
170 DISPLAY AT(6,1):"ADDRESS":">":">":">"
180 DISPLAY AT(11,1):"BIRTHDAY":">"
190 DISPLAY AT(15,1):"PRESS:"
200 DISPLAY AT(17,2):"A - ADD ENTRY"
210 DISPLAY AT(18,2):"C - CHANGE ENTRY"
220 DISPLAY AT(19,2):"K - KEEP ENTRY"
230 DISPLAY AT(20,2):"Q - QUIT"
240 DISPLAY AT(22,1):"RECORD NUMBER"
250 REM INPUT A RECORD
260 ACCEPT AT(22,15)VALIDATE(DIGIT)SIZE(5)BEEP:REC
    NO
270 IF RECNO<1 OR RECNO>MAXREC THEN 260
280 INPUT #1,REC RECNO:R,NAME$,ADDR1$,ADDR2$,ADDR3
    $,BIRTH$
290 REM DISPLAY RECORD
300 DISPLAY AT(4,4):NAME$
310 DISPLAY AT(7,4):ADDR1$
320 DISPLAY AT(8,4):ADDR2$
330 DISPLAY AT(9,4):ADDR3$
340 DISPLAY AT(12,4):BIRTH$
350 REM GET UPDATE OPTION
360 ACCEPT AT(15,8)SIZE(1)VALIDATE("ACKQ")BEEP:R$
370 IF R$="" THEN 360
380 IF R$="A" THEN GOSUB 470 :: GOTO 350
390 IF R$="C" THEN GOSUB 570 :: GOSUB 540 :: GOTO
    250
400 IF R$="K" THEN 250
410 IF R$="Q" THEN 420
420 REM CLOSE FILES
430 PRINT #1,REC 0:0;STR$(MAXREC);" ";" ";" ";" "
440 CLOSE #1
450 CALL CLEAR
460 END
470 REM ADD A RECORD
480 GOSUB 570 !READ SCREEN
490 MAXREC=MAXREC+1
500 RECNO=MAXREC
510 DISPLAY AT(22,15):RECNO
520 GOSUB 540 !PRINT RECORD
530 RETURN
540 REM PRINT A RECORD
550 PRINT #1,REC RECNO:RECNO;NAME$;ADDR1$;ADDR2$;A
    DDR3$;BIRTH$
560 RETURN
570 REM READ SCREEN
580 ACCEPT AT(4,4)SIZE(-25)BEEP:NAME$ :: IF NAME$=
    "" THEN 580
590 IF NAME$="END" OR NAME$="end" THEN 640
600 ACCEPT AT(7,4)SIZE(-25)BEEP:ADDR1$
```

```
610 ACCEPT AT(8,4)SIZE(-25)BEEP:ADDR2$
620 ACCEPT AT(9,4)SIZE(-25)BEEP:ADDR3$
630 ACCEPT AT(12,4)SIZE(-8)BEEP:BIRTH$
640 RETURN
```

### Indexed File Examples

In the last section, it was pointed out that RELATIVE files could be a little inconvenient to use. Before a particular name and address can be accessed, it is necessary to know the proper record number. Needless to say, nobody will even attempt to memorize record numbers. It is best to keep a printout of the file handy.

With a little additional effort, this problem can be overcome. The computer will keep track of the record numbers for all the individuals on file. Essentially, the computer will be maintaining an index. So, if you supply the name of a person, the computer will be able to look up the appropriate record number. Then it can retrieve the record you want.

This type of file is called an *indexed file.* The 99/4A does not have an INDEXED option for the file organization. Instead, a SEQUENTIAL file for the index and a RELATIVE file for the data will be used.

Program 3-5 shows the index file load program for the name and address information. Two arrays are defined in the program. NAMEINDEX$ contains a list of all the names on file. RECNOINDEX contains the record number associated with each name. As each record is written to the RELATIVE file, an entry is recorded in NAMEINDEX$ and RECNOINDEX.

After the RELATIVE file NAMEADDR has been built, the index file XNAMEADDR is written. The first record contains a count of the number of records in the index file. The succeeding records contain a name and a record number. For every record on the RELATIVE file, there will be one record in the index file. This is critical. If you cannot find a name in the index, you will not be able to find a record number. Thus, you have to assume that the record is simply not present on the RELATIVE file.

Program 3-6 shows the indexed file update program. This program works almost identically to the RELATIVE file update program. However, Program 3-6 asks for a name instead of a record number. You can still change information and add new records.

One of the first things that Program 3-6 does is to read the index file XNAMEADDR. The names and record numbers are stored in the arrays NAMEINDEX$ and RECNOINDEX. These arrays are referenced in several different situations.

When you're retrieving a record, NAMEINDEX$ is searched. If the name is found, then its corresponding record number is used. If the name is not in NAMEINDEX$, the program will ask you to retype the name for which you are looking. The name you type in must match character for character one of the entries in NAMEINDEX$. So be careful of punctuation and spaces—they count, too.

Before a record is added, NAMEINDEX$ is searched again. This time, if the name is found, there is an error. An attempt has been made to add a duplicate record. Program 3-6 won't let you put the same person in the file more than once. The program will ask you to retype the name.

NAMEINDEX$ and RECNOINDEX$ are updated whenever a record is added or changed. When records are added, the new names and record numbers are stored in the arrays. The same thing is done when a record is changed. You're right; a change will not affect the record number. The name may change, so NAMEINDEX$ is updated. Just to be sure, RECNOINDEX is also updated.

When the update cycle is complete, the arrays NAMEINDEX$ and RECNOINDEX are written back to disk. The file XNAMEADDR is written again. It reflects new records and name changes. It remains synchronized with the RELATIVE file NAMEADDR.

A combination of SEQUENTIAL and RELATIVE files gives us an INDEXED type of file organization. Indexed files are easier to work with from the point of view of the person using the program. The requirement for learning or looking up record numbers is gone. The computer does this by means of an index file.

## Program 3-5. Load Indexed File

```
100 REM LOAD INDEXED FILE
110 DIM NAMEINDEX$(100),RECNOINDEX(100)
120 RECNO=1
130 CALL CLEAR
140 OPEN #1:"CS1",INPUT ,INTERNAL,FIXED 128
```

```
150 OPEN #2:"DSK1.NAMEADDR",RELATIVE,OUTPUT,INTERN
    AL,FIXED 128
160 DISPLAY AT(12,1)ERASE ALL:"RECORD NUMBER "
170 INPUT #1:R,NAME$,ADDR1$,ADDR2$,ADDR3$,BIRTH$
180 IF R=99999 THEN 250
190 DISPLAY AT(12,15):RECNO
200 PRINT #2,REC RECNO:RECNO;NAME$;ADDR1$;ADDR2$;A
    DDR3$;BIRTH$
210 NAMEINDEX$(RECNO)=NAME$
220 RECNOINDEX(RECNO)=RECNO
230 RECNO=RECNO+1
240 GOTO 170
250 PRINT #2,REC 0:0;STR$(RECNO-1);" ";" ";" ";" "
260 CALL CLEAR
270 CLOSE #2
280 CLOSE #1
290 CALL CLEAR
300 OPEN #3:"DSK1.XNAMEADDR",SEQUENTIAL,OUTPUT,INT
    ERNAL,VARIABLE
310 PRINT #3:RECNO-1
320 FOR I=1 TO RECNO-1
330 PRINT #3:NAMEINDEX$(I);RECNOINDEX(I)
340 NEXT I
350 CLOSE #3
360 END
```

## Program 3-6. Update Indexed File

```
100 REM UPDATE INDEXED FILE
110 CALL CLEAR :: ON WARNING NEXT
120 DIM NAMEINDEX$(100),RECNOINDEX(100)
130 OPEN #1:"DSK1.XNAMEADDR",SEQUENTIAL,INPUT ,INT
    ERNAL,VARIABLE
140 INPUT #1:MAXREC
150 FOR I=1 TO MAXREC
160 INPUT #1:NAMEINDEX$(I),RECNOINDEX(I)
170 NEXT I
180 CLOSE #1
190 OPEN #1:"DSK1.NAMEADDR",RELATIVE,INTERNAL,UPDA
    TE,FIXED 128
200 INPUT #1,REC 0:R,N$,T1$,T2$,T3$,T4$
210 MAXREC=VAL(N$)
220 DISPLAY AT(1,7)ERASE ALL:"UPDATE NAME LIST"
230 DISPLAY AT(3,1):"NAME":">"
240 DISPLAY AT(6,1):"ADDRESS":">":">":">"
250 DISPLAY AT(11,1):"BIRTHDAY":">"
260 DISPLAY AT(15,1):"PRESS:"
270 DISPLAY AT(17,2):"A - ADD ENTRY"
280 DISPLAY AT(18,2):"C - CHANGE ENTRY"
```

```
290 DISPLAY AT(19,2):"K - KEEP ENTRY"
300 DISPLAY AT(20,2):"Q - QUIT"
310 DISPLAY AT(22,1):"WHAT NAME ?":">"
320 REM INPUT A RECORD
330 ACCEPT AT(23,4)SIZE(-25)BEEP:NAME$ :: IF NAME$
    ="" THEN 330
340 GOSUB 810 !GET THE RECORD NUMBER
350 IF RECNO<1 OR RECNO>MAXREC THEN 330
360 INPUT #1,REC RECNO:R,NAME$,ADDR1$,ADDR2$,ADDR3
    $,BIRTH$
370 REM DISPLAY RECORD
380 DISPLAY AT(4,4):NAME$
390 DISPLAY AT(7,4):ADDR1$
400 DISPLAY AT(8,4):ADDR2$
410 DISPLAY AT(9,4):ADDR3$
420 DISPLAY AT(12,4):BIRTH$
430 REM GET UPDATE OPTION
440 ACCEPT AT(15,8)SIZE(1)VALIDATE("ACKQ")BEEP:R$
450 IF R$="" THEN 440
460 IF R$="A" THEN GOSUB 610 :: GOTO 430
470 IF R$="C" THEN GOSUB 720 :: GOSUB 670 :: GOTO
    320
480 IF R$="K" THEN 320
490 IF R$="Q" THEN 500
500 REM CLOSE FILES
510 PRINT #1,REC 0:0;STR$(MAXREC);" ";" ";" ";" "
520 CLOSE #1
530 OPEN #1:"DSK1.XNAMEADDR",SEQUENTIAL,OUTPUT,INT
    ERNAL,VARIABLE
540 PRINT #1:MAXREC
550 FOR I=1 TO MAXREC
560 PRINT #1:NAMEINDEX$(I);RECNOINDEX(I)
570 NEXT I
580 CLOSE #1
590 CALL CLEAR
600 END
610 REM ADD A RECORD
620 GOSUB 720 !READ SCREEN
630 MAXREC=MAXREC+1
640 RECNO=MAXREC
650 GOSUB 670 !PRINT RECORD
660 RETURN
670 REM PRINT A RECORD
680 NAMEINDEX$(RECNO)=NAME$
690 RECNOINDEX(RECNO)=RECNO
700 PRINT #1,REC RECNO:RECNO;NAME$;ADDR1$;ADDR2$;A
    DDR3$;BIRTH$
710 RETURN
720 REM READ SCREEN
```

```
730 ACCEPT AT(4,4)SIZE(-25)BEEP:NAME$ :: IF NAME$=
    "" THEN 730
740 IF NAME$="END" OR NAME$="end" THEN 800
750 IF R$="A" THEN GOSUB 810 :: IF RECNO<>-1 THEN
    730
760 ACCEPT AT(7,4)SIZE(-25)BEEP:ADDR1$
770 ACCEPT AT(8,4)SIZE(-25)BEEP:ADDR2$
780 ACCEPT AT(9,4)SIZE(-25)BEEP:ADDR3$
790 ACCEPT AT(12,4)SIZE(-8)BEEP:BIRTH$
800 RETURN
810 REM SEARCH NAMEINDEX$ FOR NAME$
820 RECNO=-1
830 FOR I=1 TO MAXREC
840 IF NAMEINDEX$(I)=NAME$ THEN RECNO=RECNOINDEX(I
    ):: GOTO 860
850 NEXT I
860 RETURN
```

## Backup Considerations

Whenever important information is stored on the computer, you need to know what would happen if the information were lost. Is there a way of recovering it? Imagine that you lost the computer file of birthdays. Probably this information is written down somewhere. Maybe you'll just need to type in the dates again. What if you threw away our birthday list since you knew it was automated? You may be in for a bit of embarrassment as birthdays go by that you should have remembered. Can you get away with blaming it on the computer?

So be prudent. Have some backup procedures ready. The TI hardware is very reliable. But things can happen. Your tape recorder may decide to dine on your most important tape, rendering it a twisted useless mess. Maybe someone decided to reformat the diskette that contained your record of tax deductions.

Backup procedures do not have to be complicated. Perhaps a periodic printout of important data will do. If so, plan on doing this every now and then.

If you have a tape system, you can write a simple backup program. Read the data into an array in memory. Then write the array to tape. If you have two tape recorders, you can read a record from CS1 and then write it to CS2. Thus, you can dispense with memory arrays.

There are several backup procedures that will work with a disk system. The File Manager command module can make copies of individual files or the entire diskette. You can write your own copy programs also.

Program 3-7 shows how to back up the RELATIVE name and address file to tape. Only the data records are written—record zero is not. The format of the tape is such that it will work with either of the load programs, Program 3-3 or 3-5. Note that the indexed file load program automatically builds the index file XNAMEADDR. Therefore, you do not need a separate backup procedure for it.

Regardless of what approach you take, be sure that the backup copy of your data is stored in a safe place. Usually it is a good idea to store the backup copy away from your computer room.

### Program 3-7. Backup Relative File

```
100 REM BACKUP RELATIVE FILE
110 CALL CLEAR
120 OPEN #1:"DSK1.NAMEADDR",RELATIVE,INPUT ,INTERN
    AL,FIXED 128
130 INPUT #1,REC 0:R,N$,T1$,T2$,T3$,T4$
140 MAXREC=VAL(N$)
150 OPEN #2:"CS1",OUTPUT,INTERNAL,FIXED 128
160 DISPLAY AT(12,1)ERASE ALL:"RECORD NUMBER"
170 FOR I=1 TO MAXREC
180 DISPLAY AT(12,15):I
190 INPUT #1,REC I:RECNO,NAME$,ADDR1$,ADDR2$,ADDR3
    $,BIRTH$
200 PRINT #2:RECNO;NAME$;ADDR1$;ADDR2$;ADDR3$;BIRT
    H$
210 NEXT I
220 PRINT #2:99999;"ZZZ";"ZZZ";"ZZZ";"ZZZ";"ZZZ"
230 CLOSE #1
240 CLOSE #2
250 CALL CLEAR
260 END
```

### Hex Dump Utility

Occasionally, when you are working with data files, you would like to know how the 99/4A is storing your data. The hex dump, Program 3-8, lets you look inside your file and see the data as the TI sees it. This can be very useful for debugging programs.

Program 3-8 can print the contents of most data files regardless of their formats (it will not print the contents of a program file). The data is displayed in both ASCII and hexadecimal form. Here is how COMPUTE! would be displayed.

```
C O M P U T E !
4 4 4 5 5 5 4 2
3 F D 0 5 4 5 1
```

Three lines are printed. The top line shows the ASCII representation of the data. The second and third lines show the hexadecimal representations of the ASCII characters. Interpret the three lines as columns. For example, C is hexadecimal 43, O is hexadecimal 4F, and so on.

The display lines can contain up to 25 characters. If a record is longer than 25 characters, as many display lines as necessary will be printed. Similarly, if a record is shorter than 25 characters, only a partial line will be printed.

When you start up the hex dump program, it will ask you for a filename. Type in the complete name of the file such as DSK1.NAMEADDR or CS1. Then, supply the proper file characteristics. There are four possible combinations of DISPLAY, INTERNAL, FIXED, and VARIABLE. Then type in the record length. All of these characteristics will be compared with the characteristics of the tape or disk file. If there is a match, the program will continue. Otherwise, you will be asked to type in the file characteristics again.

Beginning with the first record on the file, record zero, the records will be read, converted to ASCII and hexadecimal, and printed. After each 25-character segment has been printed, a short beep will sound. You have a second or so to press any key. If you do, the program will pause. The printing will not resume until you press another key. While the program is in the pause state, the screen color will be switched to white.

There are some variations in the display, depending on the file characteristics. The record number is shown for FIXED length records. VARIABLE length records, on the other hand, are printed continuously. There is no distinguishing one record from another. Records in INTERNAL format have a one-byte length field preceding each data field. The byte length indicates the number of characters in the data field. Numeric fields will always have

a length of eight. DISPLAY format records do not have this length byte.

Program 3-8 uses the EOF function for detecting end of file. Unfortunately, EOF does not work with cassette files. Therefore, there is no way to tell when a cassette file has reached the end. An I/O error usually results as the program tries to read past the end of the file. The ON ERROR statement in the program will detect this condition. The first screen will be displayed again. You can obtain a hex dump of another file at this point.

Try the hex dump program on a few small files. This will help you become familiar with the program. You will also learn a lot about how the TI stores information.

### Program 3-8. Hex Dump Program

```
100 REM HEX DUMP PROGRAM
110 ON WARNING NEXT
120 MAXCHARS=25
130 RECLEN=128
140 BYTECOUNT=256
150 RECNO=0
160 CALL CLEAR
170 FILE$="" :: FTYPE=4 :: RECLEN=80
180 DISPLAY AT(1,11):"HEX DUMP"
190 DISPLAY AT(4,1):"DEVICE.FILENAME"
200 DISPLAY AT(9,1):"FILE TYPE"
210 DISPLAY AT(11,2):"1  DISPLAY, FIXED"
220 DISPLAY AT(12,2):"2  DISPLAY, VARIABLE"
230 DISPLAY AT(13,2):"3  INTERNAL, FIXED"
240 DISPLAY AT(14,2):"4  INTERNAL, VARIABLE"
250 DISPLAY AT(17,1):"RECORD LENGTH"
260 DISPLAY AT(6,1):FILE$
270 DISPLAY AT(9,12)SIZE(1):STR$(FTYPE)
280 DISPLAY AT(17,15)SIZE(3):STR$(RECLEN)
290 ACCEPT AT(6,1)SIZE(-28)BEEP:FILE$ :: IF FILE$=
    "" THEN 290
300 ACCEPT AT(9,12)SIZE(1)VALIDATE("1234")BEEP:FTY
    PE
310 ACCEPT AT(17,15)SIZE(-3)VALIDATE(DIGIT):RECLEN
320 REM OPEN FILE TO BE DUMPED
330 ON ERROR 430
340 ON FTYPE GOTO 350,370,390,410
350 OPEN #2:FILE$,INPUT ,DISPLAY ,FIXED RECLEN
360 GOTO 470
370 OPEN #2:FILE$,INPUT ,DISPLAY ,VARIABLE RECLEN
```

```
380 GOTO 470
390 OPEN #2:FILE$,INPUT ,INTERNAL,FIXED RECLEN
400 GOTO 470
410 OPEN #2:FILE$,INPUT ,INTERNAL,VARIABLE RECLEN
420 GOTO 470
430 REM ERROR ROUTINE
440 CALL CLEAR
450 DISPLAY AT(24,1)BEEP:"*** I/O ERROR - RETRY **
    *"
460 RETURN 180
470 REM OPEN O.K.
480 CALL CLEAR
490 IF EOF(2)THEN 850
500 IF FTYPE=3 OR FTYPE=4 THEN INPUT #2:RCD$,
510 IF FTYPE=1 OR FTYPE=2 THEN LINPUT #2:RCD$
520 STRLEN=LEN(RCD$)
530 IF FTYPE=2 OR FTYPE=4 THEN 600
540 BYTECOUNT=BYTECOUNT+STRLEN
550 IF BYTECOUNT<RECLEN THEN 600
560 IF CHARCOUNT>0 THEN GOSUB 880 !PRINT REMAINDER
570 BYTECOUNT=STRLEN
580 PRINT :"RECORD ";RECNO :: PRINT
590 RECNO=RECNO+1
600 IF STRLEN<>0 THEN 670
610 LINE1$=LINE1$&"."
620 LINE2$=LINE2$&"0"
630 LINE3$=LINE3$&"0"
640 CHARCOUNT=CHARCOUNT+1 :: BYTECOUNT=BYTECOUNT+1
650 IF CHARCOUNT>=MAXCHARS THEN GOSUB 880 !PRINT D
    UMP LINE
660 GOTO 490
670 IF FTYPE=1 OR FTYPE=2 THEN 750
680 BYTE$=CHR$(STRLEN)
690 CALL TOHEX(BYTE$,CHAR$())
700 LINE1$=LINE1$&CHAR$(1)
710 LINE2$=LINE2$&CHAR$(2)
720 LINE3$=LINE3$&CHAR$(3)
730 CHARCOUNT=CHARCOUNT+1 :: BYTECOUNT=BYTECOUNT+1
740 IF CHARCOUNT>=MAXCHARS THEN GOSUB 880 !PRINT D
    UMP LINE
750 FOR P=1 TO STRLEN
760 BYTE$=SEG$(RCD$,P,1)
770 CALL TOHEX(BYTE$,CHAR$())
780 LINE1$=LINE1$&CHAR$(1)
790 LINE2$=LINE2$&CHAR$(2)
800 LINE3$=LINE3$&CHAR$(3)
810 CHARCOUNT=CHARCOUNT+1
820 IF CHARCOUNT>=MAXCHARS THEN GOSUB 880 !PRINT D
    UMP LINE
```

```
830 NEXT P
840 GOTO 490
850 IF CHARCOUNT>0 THEN GOSUB 880 ! PRINT REMAINDE
    R
860 CLOSE #2
870 END
880 REM PRINT DUMP LINE
890 PRINT LINE1$:LINE2$:LINE3$
900 PRINT
910 LINE1$,LINE2$,LINE3$="" :: CHARCOUNT=0
920 CALL PAUSE(25)
930 RETURN
940 SUB TOHEX(BYTE$,CHAR$())
950 DIM HEXCHAR$(15)
960 IF HEXCHAR$(1)<>"" THEN 990
970 FOR I=0 TO 9 :: HEXCHAR$(I)=CHR$(48+I):: NEXT
    I
980 FOR I=10 TO 15 :: HEXCHAR$(I)=CHR$(55+I):: NEX
    T I
990 CHAR$(1)=BYTE$
1000 IF BYTE$<CHR$(21)OR BYTE$>CHR$(126)THEN CHAR$
     (1)="."
1010 IF BYTE$<>"" THEN T=ASC(BYTE$)ELSE T=0
1020 CHAR$(2)=HEXCHAR$((T AND 240)/16)
1030 CHAR$(3)=HEXCHAR$(T AND 15)
1040 SUBEND
1050 SUB PAUSE(DELAY)
1060 CALL SOUND(50,440,0)
1070 FOR I=1 TO DELAY
1080 CALL KEY(0,R,S):: IF S=1 THEN 1110
1090 NEXT I
1100 GOTO 1140
1110 CALL SCREEN(16)
1120 CALL KEY(0,R,S):: IF S<>1 THEN 1120
1130 CALL SCREEN(8)
1140 SUBEND
```

# Chapter 4

# Electronic Spreadsheets

# Electronic Spreadsheets

How does your household manage its tax planning? Is April 14 always a day of surprises when you find out what you owe Uncle Sam? If taxes are not a particular problem, what about day-to-day financial planning? Do you have specific savings or investment goals? Do you plan and budget for major purchases?

No matter how cleverly programmed it is, your TI cannot do your financial planning for you by itself. However, there are programs which can make the job of analysis much easier. Such programs are called electronic spreadsheets, or just spreadsheets for short.

If mathematics or arithmetic is a stumbling block, a spreadsheet program may be the answer for you. The spreadsheet will remove the tedious error-prone calculator work from your analysis. The spreadsheet program will do calculations rapidly and accurately. It will give you more time to plan and to evaluate different contingencies.

The spreadsheet program is a calculation tool. Like any other tool, it must work in conjunction with other tools. For example, if you want to fasten two boards together, you subconsciously realize that a hammer alone just won't do. You fetch the nails as well. So it is with a spreadsheet program. The spreadsheet program is a tool, but it requires something else that is very important—your good judgment.

This point cannot be stressed too much. Given the proper mix of data and human judgment, the spreadsheet program can do wonders. But without enough of either, the results may be disappointing at best.

## What Is a Spreadsheet?

Visualize a large piece of paper containing rows and columns of numbers. Suppose also that there is a set of written rules or formulas. The formulas tell in precise detail how the numbers are to be calculated. Imagine that there is an accountant with

an eyeshade who is sitting at a desk calculating away.

After some time, the accountant brings you the results. You look at them, think about them, and maybe confer with others. You hate to ask, but you would really like to see what the results would be under different conditions. So you call the accountant over and point out which numbers should be changed. The accountant goes back to work. The calculator grinds away some more. Eventually, he delivers another set of answers.

This scenario describes spreadsheet analysis before the age of microcomputers. The spreadsheet is the paper with the rows and columns of numbers. The formulas tell what is to be done with the numbers. To emphasize the point again, notice that human judgment is an important ingredient as well.

Technology has advanced rapidly. Spreadsheet analyses can now be done on a home computer. The computer can replace the pencil, paper, and calculator. Spreadsheet programs will store rows and columns of numbers as well as formulas in memory. When told to do so, the computer will do the calculations quickly and accurately. And it will do them over and over again.

What about the accountant? The accountant is still hard at work. The accountant is busy with more complex tasks and analyses. There is no doubt that our accountant is using a spreadsheet program on the job also.

The term *model* will often be used here to refer to a spreadsheet. This is because the spreadsheet can be thought of as a model of some situation—financial or otherwise. Spreadsheets are abstract models. They use the language of mathematics to describe reality.

Let's take a look at an example of a spreadsheet analysis. Imagine you are putting aside a little bit of money each payday toward a college fund. The amount set aside is a percentage of your take-home pay minus expenses. The boss has suggested that you can expect a raise very shortly. You are also considering taking a new position. The new job will pay more, but will mean more expense since it requires more entertaining and the like. What effect will there be on the college fund?

Figure 4-1 shows how this information would be arrayed on a spreadsheet. If you keep good records, the figures for take-home pay and expenses should be accurate. The figures

for expenses associated with the new job, though, are estimates.

The result of the calculations shows that the college fund does not see any benefit from the new job. What if your new job expense figures are incorrect? With an electronic spreadsheet, you can easily try different numbers and recalculate.

This is a very simple example. It does point out, though, how the spreadsheet can be a helpful tool around the home.

## Spreadsheets and the TI

There are several ways for you to use spreadsheet programs on your TI-99/4A. You might wish to purchase a commercially available product. Texas Instruments offers Microsoft Corporation's *MultiPlan* product. This is the same *MultiPlan* that is available on many business microcomputers such as the IBM PC. *MultiPlan* is an advanced spreadsheet product that offers many features. However, *MultiPlan* requires a fully expanded TI.

You will need:

- a 32K memory expansion,
- a disk controller unit, and
- a disk drive (preferably two).

An RS232 interface and printer are optional. If you have a full TI system and you need a spreadsheet program, you should seriously consider *MultiPlan.*

What do you do if you don't have all of the required hardware? This chapter describes two spreadsheet programs that have been designed with your needs in mind. The programs are called "Tiny Plan" and "Tiny Plan 2." The names are similar because the programs themselves are similar.

Figure 4-2 outlines the hardware requirements for Tiny Plan and Tiny Plan 2; both will run on a TI with the Extended BASIC cartridge.

## The Tiny Plan Family Tree

What are the differences between Tiny Plan and Tiny Plan 2? Figure 4-3 summarizes the important features of these two spreadsheet programs.

Tiny Plan is just about the simplest spreadsheet program available for the 99/4A. Tiny Plan will accommodate a larger model than will Tiny Plan 2. Tiny Plan is well-suited for most household spreadsheet analyses.

Tiny Plan 2 incorporates several convenience features. It will print selected rows and columns of the spreadsheet. It will also pass results to other programs which will, in turn, plot the results. Of course, these extra features extract a price. Tiny Plan 2 cannot handle spreadsheets as large as Tiny Plan. And Tiny Plan 2 requires additional hardware for some of the extra features.

An important design feature of these two programs is that they are compatible. Tiny Plan and Tiny Plan 2 define spreadsheets in exactly the same way. In addition, the calculation features of the two are identical. So, you don't have to make an either-or choice. It is possible to run the same spreadsheet with either program (keeping in mind the size restrictions, of course). Whenever you need to, you can capitalize on the special features of each. But since the data files created and saved are not compatible, it is not possible to load data created by Tiny Plan and use it with Tiny Plan 2 and vice versa. If you wish to use both programs with the same information, you will have to enter the data twice.

Notice that both programs will make use of the 32K memory expansion board. This allows you to run much larger spreadsheets.

**Figure 4-1. Sample Spreadsheet Analysis**

| | Present Job | Raise | New Job |
|---|---|---|---|
| Take-home | 2000 | 2200 | 2400 |
| Expenses | 1500 | 1500 | 1700 |
| Left | 500 | 700 | 700 |
| College % | 10 | 10 | 10 |
| College $ | 50 | 70 | 70 |

**Figure 4-2. Tiny Plan and Tiny Plan 2 System Requirements**

**Tiny Plan**

Required:

- TI-99/4A console
- Extended BASIC
- Cassette tape recorder

Optional:

- 32K memory expansion

**Tiny Plan 2**
Required:
- TI-99/4A console
- Extended BASIC
- Cassette tape recorder

Optional:
- 32K memory expansion
- Disk drive
- Disk controller
- RS232 interface
- Printer

**Figure 4-3. Tiny Plan—Tiny Plan 2 Comparison**

| | Tiny Plan | Tiny Plan 2 |
|---|---|---|
| Number of rows | 26 | 11 |
| Number of columns | 16 | 11 |
| Row labels | Yes | Yes |
| Column labels | Yes | Yes |
| Row totals | Automatic | Automatic |
| Column totals | Automatic | Automatic |
| Calculations | +,−,*,/ %,%+, %−,%D | +,−,*,/ %,%+, %−,%D |
| Save/load data | Yes | Yes |
| Save/load models | Yes | Yes |
| Print results | No | Yes |
| Plot results | No | Yes |

## Getting Started

The sections that follow will describe how to use the spreadsheet programs. Unless stated otherwise, the instructions refer to both Tiny Plan and Tiny Plan 2.

You are eager to begin a spreadsheet analysis. Do you turn on the TI, load the program tape, and have a go at it? Nope. The first step is an old-fashioned pencil and paper step. This first step involves creating a written sketch of the model or spreadsheet. Look back at Figure 4-1. It represents a model of a hypothetical college fund.

What were the steps? First, three situations were defined:
—the present job,
—the anticipated raise, and
—the new job.
The three situations became column headings of a chart. Next, the common aspects of these three situations were listed:
—take-home pay,
—expenses,
—amount left,
—percent allocated to the college fund, and
—dollars allocated to the college fund.
These five items were listed as row headings of a chart.

Thus, the first step involves sketching a chart. You must decide what goes in the columns and what goes in the rows.

The next step is to fill in any numbers that do not need calculations. In this case the take-home pay in each of the three cases may be estimated. Similarly, an educated guess of expenses can be entered. The only other numbers known are the college fund allocation percentage.

For the numbers that must be calculated, the money left after expenses and the amount for the college fund, the computer must be told exactly how to perform the calculations. The calculation rules would say:

—subtract the expenses from the take-home pay, giving the amount left

—multiply the amount left by the college fund percentage, giving the amount of the college fund.

That completes the model for the college fund. Let's review what we did:

—sketched a chart and assigned column and row names,

—filled in numbers that were known or could be estimated, and

—determined the calculation rules for the numbers that the computer would compute.

Follow these three steps when you develop your own models. It doesn't take very long, and it will become second nature to you after a while.

### Tiny Plan Basics

Once the paper and pencil work is done, it is time to start thinking about using the spreadsheet programs. Below is a discussion describing the transformation of the paper and pencil

model into something that Tiny Plan understands.

Refer to Figure 4-4 which shows an empty Tiny Plan spreadsheet. This represents everything described about Tiny Plan up to this point—nothing. Tiny Plan starts out with rows and columns of zeros.

Notice, too, that column and row names are present. The columns are called C0, C1, C2, and so on. Similarly, R0, R1, and R2 are the names of the rows. If you like, use these names.

Tiny Plan certainly understands them. However, things would be a bit clearer if the names used were more descriptive. For example, the columns might be called

—Present job,
—Raise, and
—New job.

This way, the column names would match the pencil and paper model exactly. The same could be done with the row names.

The rows and columns are numbered beginning with zero. Do not be concerned with this too much. Just about all of the time, you will be using the row and column names and not the numbers. The print and plot options of Tiny Plan 2 represent the only instances where the numbers are necessary. Keep Figure 4-4 handy for future reference.

A spreadsheet full of zeros does not interest us a great deal (unless it is our tax bill). Once the desired row and column names are established, the proper places for the numbers can easily be located. Using the example again, the number 2000 goes in the Present Job column in the Take-Home Pay row.

The last step in the model definition process is describing the calculation rules to Tiny Plan. At this point, you will be programming the spreadsheet. Tiny Plan can remember a whole series of calculations. It can do the entire series at the push of a button. Each calculation in the series is called a *step*.

There are two calculation steps in the college fund model. The first is the subtraction of expenses from take-home pay. The second is the percentage calculation.

Tiny Plan's calculation methods have a few important features. The calculations are done on entire columns of numbers or on entire rows of numbers. For example, two columns of numbers could be added together and the sum placed in a

third column. Every number in the column participates in the calculation. Row and column calculation steps can be mixed in a calculation series. However, a row and column cannot be mixed in the same step.

Look at the example again. In this case, the calculations involve rows of numbers. When the expenses are subtracted from the take-home pay, three subtractions actually take place:

2000 − 1500,
2200 − 1500, and
2400 − 1700.

Can you begin to see the advantages of Tiny Plan? Visualize a model with 10 rows and columns and about 15 or so calculation steps. Would you like to do that by hand? Of course not. But your TI can handle such a spreadsheet at the push of a button.

There are many times when you'll want to add up all the numbers in a column or all the numbers in a row. This calculation turns out to be very common. As a timesaver, Tiny Plan has built these calculations in. Whenever a series of calculations is completed, Tiny Plan automatically computes the row and column sums. You don't even have to push a button.

The row sums are stored in the rightmost column. The column sums are stored in the bottommost row. In other words, Tiny Plan has reserved one row and one column for itself.

So the process of putting the model in Tiny Plan follows the pencil and paper approach very closely. You do not have to do anything on paper that is wasted. By the same token, when you get to the computer stage, there will be no extra steps that get in your way.

**Figure 4-4. An Empty Tiny Plan Spreadsheet**

| | | 0 | 1 | 2 | |
|---|---|---|---|---|---|
| | | C0 | C1 | C2 | ... |
| 0 | R0 | .00 | .00 | .00 | |
| 1 | R1 | .00 | .00 | .00 | |
| 2 | R2 | .00 | .00 | .00 | |
| | . | | | | |
| | . | | | | |
| | . | | | | |

## Operating Tiny Plan

Tiny Plan and Tiny Plan 2 are menu-driven programs. When you are operating Tiny Plan, it will display a list of possible selections. These selections represent the functions that can be performed from a particular point in the spreadsheet analysis. You may choose any one of the selections with a single keystroke: Press the number of the selection that you want. Tiny Plan will take over from there.

Many of the functions have, in turn, subfunctions. The subfunctions will be displayed on another menu screen. So you have another choice to make. Thus, Tiny Plan uses a multilevel menu system.

Suppose you do not choose any of the selections. If you just press ENTER, the prior menu screen will be displayed. This is a very convenient feature. Suppose a particular function was invoked by mistake. There is an easy way to escape.

When Tiny Plan is first started up, there will be a slight pause. The following message will be displayed:

"Initializing Tiny Plan"
"Just a moment..."

Tiny Plan is getting things ready for you. It is building an empty spreadsheet. The row and column names are being set to the R and C names. All of the numbers are being set to zero. This takes a little time.

After a few seconds, the main menu screen will be displayed. This is the focal point of Tiny Plan. All of the major functions are invoked from the main menu screen:

1 Define the model,
2 Run the model,
3 Load model and data,
4 Save model and data, and
5 Leave Tiny Plan.

What happens if ENTER is pressed from the main menu screen? There is no prior menu screen that can be displayed. Instead, Tiny Plan will reinitialize the spreadsheet.

This feature can be both convenient and troublesome. When you are finished with one spreadsheet, you can proceed to the next very easily. Just press ENTER from the main menu. After a few seconds, you will get an empty spreadsheet again. On the other hand, if ENTER is accidentally pressed, any work in progress will be lost. Needless to say, you need to be extra careful.

**Define the Model.** Tiny Plan model definition follows the pencil and paper methods. The model definition screen offers three selections:

1 Give row names,
2 Give column names, and
3 State calculation rules.

Whichever selection you choose, another screen will be presented. This screen will ask for additional details.

The assignment of row and column names is very similar. Tiny Plan will show you the current name of each row and column. Assuming the spreadsheet is empty, you will see names such as R0 and C0. If you want to use the name shown, hit ENTER. If another name is more appropriate, type in the new name and then hit ENTER. When you are finished assigning names, type "END" or "end".

Row and column names may be up to ten characters long. *Do not use the same name twice.* This is as confusing to Tiny Plan as it is to you. If there were two rows called Take-Home, which one would you use? Tiny Plan will let you use two Take-Home's. However, only the first occurrence of Take-Home will be used in any calculations.

Do not assign every row and column a name unless it is necessary. Tiny Plan already uses most of the TI's available 16K memory. If every possible row and column is assigned a ten-character name, there is a risk of not having enough memory left for Tiny Plan.

The calculation rules are entered one step at a time. The first step is step zero, followed by step one and so on.

Each calculation step has five parts:

1 the type of calculation (row or column),
2 the first row/column name,
3 the calculation symbol or operator,
4 the second row/column name, and
5 the answer row/column name.

As you type this information, Tiny Plan will do certain checks. Tiny Plan will make sure that it can find each of the row and column names. For example, suppose we specify XYZ as a row name that will be used in a multiplication step. XYZ should have been previously defined as a row name. If it was not, Tiny Plan will not let it be used in a calculation step.

Be careful when selecting row and column names. Lowercase letters can be used, but the row and column names must be typed in exactly the same way that they were originally defined. "Take-Home" is not the same as "TAKE HOME," "take home," "Take-home," or "Take home."

Tiny Plan also checks the operator. The operator must represent a mathematical operation that Tiny Plan knows how to do. Figure 4-5 shows the calculations that Tiny Plan knows. There are several percentage operators in addition to the standard arithmetic operators.

When you enter percentages, enter them as whole numbers, not as fractions. For example, ten percent would be typed as 10, not .10. Similarly, twelve and one half percent would be 12.5, not .125.

Tiny Plan will also calculate results this way. The %D operator yields a percentage as a result. Tiny Plan will automatically convert the result. Five percent will be displayed as 5 instead of .05.

Notice that Figure 4-5 also shows the order of the operands. This is important for all of the calculations except addition and multiplication. Suppose you were entering a division step for a column. The first column name would be the dividend and the second the divisor. This follows from Figure 4-5 which says the first column is divided by the second.

Tiny Plan remembers each calculation step as you enter it. However, the calculations are not done right away. They are just entered, checked, and stored. This makes sense when you consider that so far no data has been entered.

The three steps of assigning row names, assigning column names, and stating the calculation rules complete the model definition phase. The spreadsheet is beginning to take shape. At this point, you can go back and double-check your work. It's a good idea to make sure the row and column names are correct and that the calculations will do what you want.

**Run the Model.** This is the spreadsheet function that you will probably use most often. Tiny Plan 2 offers four selections:

1 Review or change data,
2 Do the calculations,
3 Print the results, and
4 Save data for plotting.

Selections 3 and 4 are not available in Tiny Plan. However, selections 1 and 2 are identical in the two programs.

Notice that the change data and calculate operations both come under the Run the Model function. This particular grouping is designed to make Tiny Plan easy to use. You may decide to change some numbers, do the calculations again, and then examine the new results. Spreadsheets are ideal for these kinds of "what if" applications. And Tiny Plan enables you to do this with a minimum of keystrokes.

There are too many numbers for Tiny Plan to show on the screen at one time. When you choose selection 1, you will see only a portion of the numbers. The portion that you can see is what will fit in Tiny Plan's display window. If that was all you could ever see, there would be some problems.

Fortunately, you can move the display window around. By doing this, you can eventually examine every number in the spreadsheet. When you are in the mode to Review or change data, selection 1, certain keys are active. They control the movement of the display window.

E move the display window up N lines
X move the display window down N lines
S move the display window to the left one column
D move the display window to the right one column

**Figure 4-5. Tiny Plan Operators**

| | |
|---|---|
| + | Add the 1st row/column to the 2nd row/column |
| − | Subtract the 2nd row/column from the 1st row/column |
| * | Multiply the 1st row/column by the 2nd row/column |
| / | Divide the 1st row/column by the 2nd row/column |
| % | Compute a percentage of the 1st row/column. The percentage is in the 2nd row/column. |
| %+ | Increase the 1st row/column by a percentage. The percentage is in the 2nd row/column. |
| %− | Decrease the 1st row/column by a percentage. The percentage is in the 2nd row/column. |
| %D | Compute the percent difference between the 2nd row/column and the 1st row/column. Use the 1st row/column as the base. |

C open the left column for change
Q exit the display/change mode

Moving the display window around is a slow operation on the TI. The keyboard will not be active until the display operation is finished. Listen for a beep. When it sounds, the keyboard is ready for your next command.

Normally, two columns of numbers are displayed. Eight digits of the number are shown—six digits before the decimal place and two after it. If a number is greater than 999,999.99, it cannot fit in the display format. In such a case, asterisks will be shown. Even so, the full accuracy of the TI will be retained.

Keys E and X move the display window up and down N lines at a time. Right now, N is set for 11 lines. You can alter this by changing the variable NL at the very beginning of the program.

The C key opens up the left column of numbers so that you can change them. Before you do this, position the display window so that the desired numbers show up in the left column. You might find the FCTN ERASE key handy here. You can completely erase a number from the screen. This gives you a clear space to type in a new number.

Once you are in the mode to change data and have positioned the numbers to be changed in the left column, move the cursor over the number to be changed using the FCTN and the arrow keys. Make the necessary change and press ENTER. Continue until you have made all the necessary changes in the data. Once all the changes have been made to the rows and columns displayed, move the cursor down until it disappears. You have now left the change data mode.

Press the Q key when you are finished examining and changing data. The Run menu will show up again.

Selection 2 does all of the calculations that you previously entered. They are done step by step. A message on line 24 tells you what step Tiny Plan is currently working on. You will find that the calculations are done pretty quickly.

When all the calculations are done, line 24 will show,

"Row and column totals".

Tiny Plan is computing the sums. There is a healthy pause at this point. Notice that selection 2 did not require any additional information. All the calculations were literally done at the push of a button.

Don't forget that the calculation rules are still in Tiny Plan's memory. You can cycle back and change some numbers and recalculate again. You have to supply the calculation rules only once.

You can use selection 3 if you are using Tiny Plan 2 and have a printer. As with the display mode, only a portion of the spreadsheet can be printed at a time. Tiny Plan asks you to define the print window by supplying the starting and ending column and row numbers. Tiny Plan also asks you how you want the numbers printed and how many spaces should be printed between columns. The print mode is fairly flexible. *Make sure the printer OPEN statement is appropriate for your printer.*

Tiny Plan 2 will also save data for subsequent plotting. It will save the numbers from two rows or from two columns. The numbers are written to an output file as X,Y pairs. (X represents the horizontal axis and Y the vertical.) Tiny Plan will ask you which row/column number should be used for X and which for Y. The file that is created this way can be used by the bar graph program in the next chapter.

This completes the Run mode. There are many functions available to you. The best way to learn them is to try them out.

**Load and Save.** After you've developed a model, you will probably find a need for it again. Selections 3 and 4 of the main menu invoke the load and save facilities of Tiny Plan. You may use either a cassette recorder or a disk drive as the storage device.

The save function stores all of the information about your model definition. In addition, all of the numbers in the spreadsheet are stored. Tiny Plan saves:

—the row and column names,
—the calculation rules, and
—the contents of the spreadsheet.

The load function does just the reverse. It reads all the above information from tape or disk. It then places the information in the spreadsheet. Thus, the spreadsheet looks just like it did when you saved it.

Here are a few suggestions for using the load and save functions. Suppose you want to save only the model definition and not the data. Well, after you finish defining the row and

column names and the calculation rules, go back to the main menu and use the save function. Yes, the spreadsheet will be saved too. But the numbers are all zero. In effect, you've managed to save just the model definition. So you can use it with a fresh set of data whenever you want.

What if, on the other hand, you are doing a monthly analysis of some sort. You would like to save the data instead of keying it in again every time. The save and load functions handle this nicely. Return to the main menu after you've entered the new month's data, but before doing any calculations. Use the save function. Tiny Plan will store the model definition and data. Since you have not done any calculations yet, none of the computed numbers will be stored. So you have the model definition and original data safely stored.

There is one little restriction, however. The data files created by Tiny Plan and Tiny Plan 2 will not work with each other. This is because the model sizes are quite different. You cannot save a spreadsheet with Tiny Plan and load it with Tiny Plan 2.

**Leave Tiny Plan.** When you are all finished with your spreadsheet analysis, you can use selection 5 on the main menu. Tiny Plan will clear the screen and put you back in BASIC mode.

There is no way that Tiny Plan can be resumed except by typing RUN. When this is done, you will start over again with an empty spreadsheet.

### Using Memory Expansion

Additional memory will allow you to use larger spreadsheets. However, this is not an automatic process. Some program changes will be required.

At the beginning of the program, there is a DIM statement for:

—MODEL,
—ROW$, and
—COL$.

These are arrays which contain the spreadsheet and row and column names. Adjust the dimensions of these arrays according to the number of rows and columns that you want. Note that MODEL is a two-dimensional array. The first dimension holds the rows and the second the columns.

The variables MAXCOL and MAXROW should be set so that they match the dimensions of MODEL. MAXCOL and MAXROW are very important to the proper operation of Tiny Plan.

You may also increase the number of calculation steps that Tiny Plan can handle. Find the DIM statement for the array CALC. Its dimensions will probably be 15 by 4. Increase the first dimension only. Increasing the second dimension will have no effect other than wasting memory. The variable MAXCALC should also be set to match the first dimension of MODEL.

These changes are not difficult. You can make them and try out Tiny Plan. Use the SIZE command to get an idea of how much memory is being used.

There are, however, additional changes that will be required. These changes are more difficult. Locate the save and load subroutines. They are marked with REM statements. These subroutines should be modified so they PRINT and INPUT the entire contents of ROW$, COL$, CALC, and MODEL.

These subroutines are written so that they pack as much data in a single record as possible. This improves saving and loading times especially with tape storage. The precise modifications will vary, depending on how you redimension the arrays.

Compare these same subroutines in Tiny Plan and Tiny Plan 2. Since Tiny Plan's arrays are larger, its save and load subroutines are a little different. This should give you some clues on how you might proceed.

### Sample Models

How will you use Tiny Plan around the home? Here are three samples which you can use as is or which you can modify for your own needs. The samples are:

—Investment analysis (Figure 4-6),
—Electricity usage (Figure 4-7), and
—Budget projections (Figure 4-8).

The referenced figures show the row and column names and the calculation rules.

Asterisks indicate the location of numbers that you should fill in. All the other numbers are computed.

The investment analysis model is a simple means of tracking the various savings instruments. Each instrument may have a different yield and a different initial investment.

There are two assumptions in this model. First, that the yield stays the same over the course of a few years. Second, that there are no withdrawals or additional deposits. Of course, if these assumptions do not hold, the actual results will differ from the predicted results.

The electricity usage model will help you analyze your monthly electric bill. It stores the monthly kilowatt hours used over the course of several years. It computes the month-to-month percent difference. You should be better able to watch your consumption. You will also be able to see if there is any effect due to home improvements such as caulking or adding insulation.

The budget model is the most complicated. It employs both row and column calculations. The model shows salary and major expense categories. A different inflation rate may be assigned to each entry. The total expenses are computed. Finally, the expenses are subtracted from the income to get the amount left, or *disposable income.*

You may want to tailor this model to your own circumstances a bit. Perhaps there is more than one source of income. Perhaps you would prefer a different set of expense categories. You may enter monthly or annual figures as you please. Try to make your expense figures as accurate as possible. You may want to keep detailed records of expenses for several months.

These are just some ideas. They are typical of the things that Tiny Plan can do. Try them out.

**Figure 4-6. Sample—Investment Analysis**

| | Principal | Yield | 1983 | 1984 | 1985 |
|---|---|---|---|---|---|
| CD | *** | *** | | | |
| All Savers | *** | *** | | | |
| Passbook | *** | *** | | | |

**Calculation Rules**

| Step | Type | Calculation | Answer |
|---|---|---|---|
| 0 | Column | Principal %+ Yield | 1983 |
| 1 | Column | 1983 %+ Yield | 1984 |
| 2 | Column | 1984 %+ Yield | 1985 |

Figure 4-7. Sample—Electricity Usage

| | Jan | Feb | Mar | Apr | ... |
|---|---|---|---|---|---|
| 1980 kwh | *** | *** | *** | *** | ... |
| 1981 kwh | *** | *** | *** | *** | ... |
| 1981 % | | | | | |
| 1982 kwh | *** | *** | *** | *** | ... |
| 1982 % | | | | | |
| 1983 kwh | *** | *** | *** | *** | ... |
| 1983 % | | | | | |
| 1984 kwh | *** | *** | *** | *** | ... |
| 1984 % | | | | | |

Calculation Rules

| Step | Type | Calculation | Answer |
|---|---|---|---|
| 0 | Row 1980 | kwh %D 1981 kwh | 1981 % |
| 1 | Row 1981 | kwh $D 1982 kwh | 1982 % |
| 2 | Row 1982 | kwh %D 1983 kwh | 1983 % |
| 3 | Row 1983 | kwh %D 1984 kwh | 1984 % |

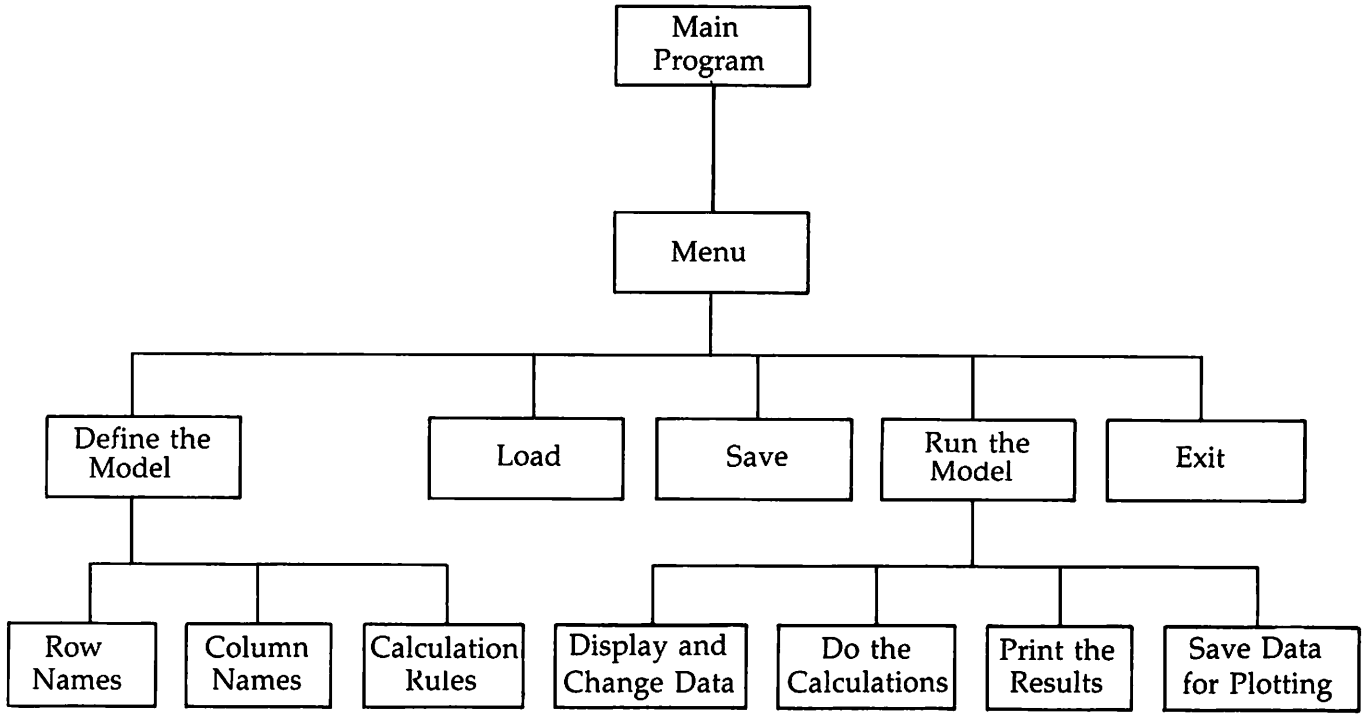
## How It Works

Tiny Plan and Tiny Plan 2 are very similar programs. In fact, Tiny Plan is just a smaller version of Tiny Plan 2.

The structure chart for Tiny Plan 2 is shown in Figure 4-9. The labels in the structure chart boxes generally correspond to REM statements that mark the beginning of a subroutine. The main program dimensions the arrays and initializes a few variables. The main menu displays the available selections. The main menu also invokes the proper subroutine to process the function selected. Notice that the Define and Run model functions themselves invoke other functions. Tiny Plan is structured in a similar manner. The Print Results and Save Data for Plotting subroutines are omitted. This frees up some memory for additional data storage.

Figure 4-10 lists the major variables used by Tiny Plan and Tiny Plan 2. An explanation of each of the subprograms is shown in Figure 4-11.

Figure 4-9. Tiny Plan 2 Program Structure



### Figure 4-8. Sample—Budget Projections

| | 1983 | 1983 Infl | 1984 | 1984 Infl | 1985 |
|---|---|---|---|---|---|
| Salary | *** | | *** | | |
| Food | *** | | *** | | |
| Mortgage | *** | | *** | | |
| Utilities | *** | | *** | | |
| Auto | *** | | *** | | |
| Clothes | *** | | *** | | |
| Other | *** | | *** | | |
| Total expense | | | | | |
| Amount Left | | | | | |

Note: "Infl" stands for inflation rate.

### Calculation Rules

| Step | Type | Calculation | Answer |
|---|---|---|---|
| 0 | Column | 1983 %+ 1983 Infl | 1984 |
| 1 | Column | 1984 %+ 1984 Infl | 1985 |
| 2 | Row | Food + Total expense | Total expense |
| 3 | Row | Mortgage + Total expense | Total expense |
| 4 | Row | Utilities + Total expense | Total expense |
| 5 | Row | Auto + Total expense | Total expense |
| 6 | Row | Clothes + Total expense | Total expense |
| 7 | Row | Other + Total expense | Total expense |
| 8 | Row | Salary − Total expense | Amount left |

### Figure 4-10. Tiny Plan and Tiny Plan 2 Variables

| | |
|---|---|
| MODEL(*n*,*n*) | This two-dimensional array contains the data for the model. You may enlarge the dimensions of the array if you are using the 32K memory expansion. |
| MAXROW | The maximum number of rows in the model. MAXROW must be set so that it matches the first dimension of the array MODEL. |
| MAXCOL | The maximum number of columns in the model. MAXCOL must be set so that it matches the second dimension of the array MODEL. |

| | |
|---|---|
| ROW$(*n*) | This string array contains the row names. The dimension of ROW$ must match the first dimension of the array MODEL. |
| COL$(*n*) | This string array contains the column names. The dimension of COL$ must match the second dimension of the array MODEL. |
| CAL(*n,n*) | This array contains the calculation rules. Set the first dimension of CALC to the number of calculation rules that can be stored. The second dimension of CALC contains, in subscripts zero through four:<br>—the type of calculation (row or column),<br>—a pointer to the first operand,<br>—a pointer to the operator,<br>—a pointer to the second operand, and<br>—a pointer to the result. |
| MAXCALC | The maximum number of calculation rules. MAXCALC must match the first dimension of the array CALC. |
| NL | NL governs the display of the model. Set NL to the number of rows that you want displayed at a time. A larger number lets you see more of the model at once. However, scrolling the display takes longer. |

**Figure 4-11. Tiny Plan and Tiny Plan 2 Subprograms**

| | |
|---|---|
| OPENI | Selects and opens an input device. |
| OPENO | Selects and opens an output device. |
| MENU | Displays the main menu and obtains the desired selection. |

**Program 4-1. Tiny Plan**

```
100 REM TINY PLAN
110 ON BREAK NEXT :: ON WARNING NEXT
120 NL=11 :: MAXROW=20 :: MAXCOL=15 :: MAXCALC=15
130 DIM MODEL(20,15),ROW$(20),COL$(15)
140 DIM OPLIST$(8),CHOICE$(10),CALC(15,4)
150 GOSUB 280 !INITIALIZE
160 REM  MAIN MENU
170 DATA 5," 1  Define the model"
```

```
180 DATA " 2  Run the model"
190 DATA " 3  Load model&data"
200 DATA " 4  Save model&data"
210 DATA " 5  Leave Tiny Plan"
220 RESTORE 170 :: READ N :: FOR I=1 TO N :: READ
    CHOICE$(I):: NEXT I
230 CALL MENU("Tiny Plan 2",CHOICE$(),N,FCODE)
240 IF FCODE=0 THEN 150
250 IF FCODE=5 THEN 270
260 ON FCODE GOSUB 380,1400,2140,2240 :: GOTO 220
270 CALL CLEAR :: END
280 REM INITIALIZE
290 DISPLAY AT(12,1)ERASE ALL:"Initializing Tiny P
    lan": :"Just a moment ..."
300 DATA 8,+,-,*,/,%,%+,%-,%D
310 RESTORE 300 :: READ NOPR :: FOR I=1 TO NOPR ::
     READ OPLIST$(I):: NEXT I
320 FOR I=0 TO MAXROW-1 :: ROW$(I)="R"&STR$(I):: N
    EXT I
330 FOR I=0 TO MAXCOL-1 :: COL$(I)="C"&STR$(I):: N
    EXT I
340 ROW$(MAXROW)="COL TOTAL" :: COL$(MAXCOL)="ROW
    TOTAL"
350 FOR I=0 TO MAXROW :: FOR J=0 TO MAXCOL :: MODE
    L(I,J)=0 :: NEXT J :: NEXT I
360 FOR I=0 TO MAXCALC :: FOR J=0 TO 4 :: CALC(I,J
    )=0 :: NEXT J :: NEXT I
370 RETURN
380 REM DEFINE MODEL
390 DATA 3," 1  Give row names"
400 DATA " 2  Give column names"
410 DATA " 3  State calculation rules"
420 RESTORE 390 :: READ N :: FOR I=1 TO N :: READ
    CHOICE$(I):: NEXT I
430 CALL MENU("Define the model",CHOICE$(),N,FCODE
    )
440 IF FCODE=0 THEN 460
450 ON FCODE GOSUB 470,580,690 :: GOTO 420
460 RETURN
470 REM ROW NAME
480 DISPLAY AT(1,8)ERASE ALL:"Give Row Names"
490 GOSUB 1160
500 FOR I=0 TO MAXROW-1
510 DISPLAY AT(3,1):"Row ";I;" of ";MAXROW
520 DISPLAY AT(8,3)SIZE(10):ROW$(I)
530 GOSUB 1210 !READ SCREEN
540 IF NM$="END" OR NM$="end" THEN 570
550 ROW$(I)=NM$
560 NEXT I
```

```
570 RETURN
580 REM COL NAME
590 DISPLAY AT(1,6)ERASE ALL:"Give Column Names"
600 GOSUB 1160
610 FOR I=0 TO MAXCOL-1
620 DISPLAY AT(3,1):"Column ";I;" of ";MAXCOL
630 DISPLAY AT(8,3)SIZE(10):COL$(I)
640 GOSUB 1210 !READ SCREEN
650 IF NM$="END" OR NM$="end" THEN 680
660 COL$(I)=NM$
670 NEXT I
680 RETURN
690 REM CALCULATION RULES
700 DISPLAY AT(1,2)ERASE ALL:"State Calculation Ru
    les"
710 DISPLAY AT(7,1):"Row(R) or Col(C) or Quit(Q)?"
720 DISPLAY AT(8,3):"->"
730 DISPLAY AT(10,1):"1st row/column name"
740 DISPLAY AT(11,3):"->"
750 DISPLAY AT(13,1):"Calculation:"
760 DISPLAY AT(14,3):"->"
770 DISPLAY AT(16,1):"2nd row/column name"
780 DISPLAY AT(17,3):"->"
790 DISPLAY AT(19,1):"Answer row/column name"
800 DISPLAY AT(20,3):"->"
810 DISPLAY AT(23,1):"Calculations can be:"
820 DISPLAY AT(24,1):"+ - * / % %+ %- %D"
830 FOR K=0 TO MAXCALC
840 DISPLAY AT(4,1):"Step ";K;" of ";MAXCALC
850 IF CALC(K,0)=2 THEN 900
860 T$="R" :: T1$=ROW$(CALC(K,1)):: T2$=OPLIST$(CA
    LC(K,2))
870 T3$=ROW$(CALC(K,3)):: T4$=ROW$(CALC(K,4))
880 GOTO 910
890 T$="C" :: T1$=COL$(CALC(K,1)):: T2$=OPLIST$(CA
    LC(K,2))
900 T3$=COL$(CALC(K,3)):: T4$=COL$(CALC(K,4))
910 DISPLAY AT(8,5)SIZE(1):T$
920 DISPLAY AT(11,5)SIZE(10):T1$
930 DISPLAY AT(14,5)SIZE(2):T2$
940 DISPLAY AT(17,5)SIZE(10):T3$
950 DISPLAY AT(20,5)SIZE(10):T4$
960 ACCEPT AT(8,5)SIZE(-1)VALIDATE("RCQ")BEEP:T$ :
    : IF T$="" THEN 960 ELSE IF T$="Q" THEN 1140
970 IF T$="R" THEN CALC(K,0)=1 ELSE CALC(K,0)=2
980 ACCEPT AT(11,5)SIZE(-10)BEEP:NM$ :: IF NM$=""
    THEN 980
990 IF T$="R" THEN GOSUB 1250 ELSE GOSUB 1300
1000 IF N=-1 THEN 980
```

```
1010 CALC(K,1)=N
1020 ACCEPT AT(14,5)SIZE(-2)VALIDATE("+-*/%D")REEP
     :NM$ :: IF NM$="" THEN 1020
1030 GOSUB 1350 :: IF N=-1 THEN 1020
1040 CALC(K,2)=N
1050 ACCEPT AT(17,5)SIZE(-10)BEEP:NM$ :: IF NM$=""
      THEN 1050
1060 IF T$="R" THEN GOSUB 1250 ELSE GOSUB 1300
1070 IF N=-1 THEN 1050
1080 CALC(K,3)=N
1090 ACCEPT AT(20,5)SIZE(-10)BEEP:NM$ :: IF NM$=""
      THEN 1090
1100 IF T$="R" THEN GOSUB 1250 ELSE GOSUB 1300
1110 IF N=-1 THEN 1090
1120 CALC(K,4)=N
1130 NEXT K :: GOTO 1150
1140 FOR J=0 TO 4 :: CALC(K,J)=0 :: NEXT J
1150 RETURN
1160 REM SHOW NAME SCREEN
1170 DISPLAY AT(5,1):"What name do you want?":"Typ
     e 'END' when done."
1180 DISPLAY AT(8,1):"->{10 SPACES}<-"
1190 DISPLAY AT(23,1):"Hit ENTER to use":"the info
     rmation shown."
1200 RETURN
1210 REM READ NAME SCREEN
1220 ACCEPT AT(8,3)SIZE(-10)BEEP:NM$ :: IF NM$=""
     THEN 1220
1230 IF NM$="END" OR NM$="end" THEN 1240
1240 RETURN
1250 REM ROW #
1260 FOR I=0 TO MAXROW-1
1270 IF NM$=ROW$(I)THEN N=I :: GOTO 1290
1280 NEXT I :: N=-1
1290 RETURN
1300 REM COL #
1310 FOR I=0 TO MAXCOL-1
1320 IF NM$=COL$(I)THEN N=I :: GOTO 1340
1330 NEXT I :: N=-1
1340 RETURN
1350 REM OPERATOR #
1360 FOR I=1 TO NOPR
1370 IF NM$=OPLIST$(I)THEN N=I :: GOTO 1390
1380 NEXT I :: N=-1
1390 RETURN
1400 REM RUN MODEL
1410 DATA 2," 1  Review or change data"
1420 DATA " 2  Do the calculations"
1430 RESTORE 1410 :: READ N :: FOR I=1 TO N :: REA
     D CHOICE$(I):: NEXT I
```

```
1440 CALL MENU("Run the Model",CHOICE$(),N,FCODE)
1450 IF FCODE=0 THEN 1470
1460 ON FCODE GOSUB 1480,1820 :: GOTO 1430
1470 RETURN
1480 REM DISPLAY & CHANGE
1490 DISPLAY AT(1,4)ERASE ALL:"Review or Change Da
     ta"
1500 IMAGE "######.##  ######.##"
1510 DISPLAY AT(22,1):"S,D  Scroll columns"
1520 DISPLAY AT(23,1):"E,X  Scroll rows"
1530 DISPLAY AT(24,1):"C{4 SPACES}Change data
     {5 SPACES}Q  Quit"
1540 RWB,CW=0
1550 REM WINDOW
1560 RW=RWB :: CT=CW+1 :: IF CT>MAXCOL THEN CT=1
1570 DISPLAY AT(3,9)SIZE(9):COL$(CW)
1580 DISPLAY AT(3,20)SIZE(9):COL$(CT)
1590 FOR I=5 TO 5+NL-1
1600 DISPLAY AT(I,1)SIZE(7):ROW$(RW)
1610 DISPLAY AT(I,9):USING 1500:MODEL(RW,CW),MODEL
     (RW,CT)
1620 RW=RW+1 :: IF RW>MAXROW THEN 1640
1630 NEXT I
1640 FOR K=I+1 TO 19 :: DISPLAY AT(K,1):" " :: NEX
     T K
1650 REM POLL KBD
1660 CALL SOUND(100,440,4)
1670 CALL KEY(0,R,S):: IF S<>1 THEN 1670 ELSE R$=C
     HR$(R)
1680 IF R$="S" THEN CW=CW-1 :: IF CW<0 THEN CW=MAX
     COL
1690 IF R$="D" THEN CW=CW+1 :: IF CW>MAXCOL THEN C
     W=0
1700 IF R$="E" THEN RWB=RWB-NL :: IF RWB<0 THEN RW
     B=0
1710 IF R$="X" THEN RWB=RWB+NL :: IF RWB>MAXROW TH
     EN RWB=0
1720 IF R$="Q" THEN 1810
1730 IF R$<>"C" THEN 1550
1740 REM READ SCREEN
1750 RW=RWB
1760 FOR I=5 TO 5+NL-1
1770 ACCEPT AT(I,9)SIZE(-9)VALIDATE(NUMERIC)BEEP:N
1780 MODEL(RW,CW)=N
1790 RW=RW+1 :: IF RW>MAXROW THEN 1650
1800 NEXT I :: GOTO 1550
1810 RETURN
1820 REM DO CALC
1830 FOR I=0 TO MAXCALC
```

```
1840 IF CALC(I,0)=0 THEN 1930
1850 DISPLAY AT(24,1):"Working. Step ";I
1860 IF CALC(I,0)=2 THEN 1900
1870 R1=CALC(I,1):: N=CALC(I,2):: R2=CALC(I,3):: R
     3=CALC(I,4)
1880 FOR K=0 TO MAXCOL-1 :: C1,C2,C3=K :: GOSUB 20
     20 :: NEXT K
1890 GOTO 1920
1900 C1=CALC(I,1):: N=CALC(I,2):: C2=CALC(I,3):: C
     3=CALC(I,4)
1910 FOR K=0 TO MAXROW-1 :: R1,R2,R3=K :: GOSUB 20
     20 :: NEXT K
1920 NEXT I
1930 DISPLAY AT(24,1):"Row and column totals."
1940 FOR I=0 TO MAXROW :: T=0 :: FOR J=0 TO MAXCOL
     -1
1950 T=T+MODEL(I,J):: NEXT J
1960 MODEL(I,MAXCOL)=T :: NEXT I
1970 FOR I=0 TO MAXCOL :: T=0 :: FOR J=0 TO MAXROW
     -1
1980 T=T+MODEL(J,I):: NEXT J
1990 MODEL(MAXROW,I)=T :: NEXT I
2000 DISPLAY AT(24,1):" "
2010 RETURN
2020 REM CALC
2030 ON N GOTO 2040,2050,2060,2070,2090,2100,2110,
     2120
2040 MODEL(R3,C3)=MODEL(R1,C1)+MODEL(R2,C2):: RETU
     RN
2050 MODEL(R3,C3)=MODEL(R1,C1)-MODEL(R2,C2):: RETU
     RN
2060 MODEL(R3,C3)=MODEL(R1,C1)*MODEL(R2,C2):: RETU
     RN
2070 IF MODEL(R2,C2)<>0 THEN MODEL(R3,C3)=MODEL(R1
     ,C1)/MODEL(R2,C2)
2080 RETURN
2090 MODEL(R3,C3)=MODEL(R1,C1)*MODEL(R2,C2)/100 ::
      RETURN
2100 MODEL(R3,C3)=MODEL(R1,C1)+(MODEL(R1,C1)*MODEL
     (R2,C2)/100):: RETURN
2110 MODEL(R3,C3)=MODEL(R1,C1)-(MODEL(R1,C1)*MODEL
     (R2,C2)/100):: RETURN
2120 IF MODEL(R1,C1)<>0 THEN MODEL(R3,C3)=((MODEL(
     R2,C2)-MODEL(R1,C1))/MODEL(R1,C1))*100
2130 RETURN
2140 REM LOAD
2150 CALL OPENI(192,FCODE):: IF FCODE=0 THEN 2230
2160 INPUT #2:MAXROW,MAXCOL,MAXCALC
2170 FOR I=0 TO 10 :: INPUT #2:ROW$(I),:: NEXT I
```

```
2180 FOR I=11 TO 20 :: INPUT #2:ROW$(I),:: NEXT I
2190 FOR I=0 TO 15 :: INPUT #2:COL$(I),:: NEXT I
2200 FOR I=0 TO MAXCALC :: FOR J=0 TO 4 :: INPUT #
     2:CALC(I,J),:: NEXT J :: NEXT I
2210 FOR I=0 TO MAXROW :: FOR J=0 TO MAXCOL :: INP
     UT #2:MODEL(I,J),:: NEXT J :: NEXT I
2220 CLOSE #2
2230 RETURN
2240 REM SAVE
2250 CALL OPENO(192,FCODE):: IF FCODE=0 THEN 2330
2260 PRINT #2:MAXROW;MAXCOL;MAXCALC
2270 FOR I=0 TO 10 :: PRINT #2:ROW$(I);:: NEXT I :
     : PRINT #2
2280 FOR I=11 TO 20 :: PRINT #2:ROW$(I);:: NEXT I
     :: PRINT #2
2290 FOR I=0 TO 15 :: PRINT #2:COL$(I);:: NEXT I :
     : PRINT #2
2300 FOR I=0 TO MAXCALC :: FOR J=0 TO 4 :: PRINT #
     2:CALC(I,J);:: NEXT J :: PRINT #2 :: NEXT I
2310 FOR I=0 TO MAXROW :: FOR J=0 TO MAXCOL :: PRI
     NT #2:MODEL(I,J);:: NEXT J :: PRINT #2 :: NEX
     T I
2320 CLOSE #2
2330 RETURN
2340 SUB OPENI(RL,FCODE)
2350 DATA 3
2360 DATA " 1  Cassette 1  CS1"
2370 DATA " 2  Disk"
2380 DATA " 3  Other"
2390 RESTORE 2350 :: READ N :: FOR I=1 TO N :: REA
     D CHOICE$(I):: NEXT I
2400 CALL MENU("Open Input File",CHOICE$(),N,FCODE
     )
2410 IF FCODE=0 THEN 2480
2420 IF FCODE=1 THEN 2470
2430 DISPLAY AT(15,1):"Type device.filename"
2440 ACCEPT AT(17,1)SIZE(-28)VALIDATE(UALPHA,DIGIT
     ,".")BEEP:NM$ :: IF NM$="" THEN 2440
2450 OPEN #2:NM$,INTERNAL,INPUT ,VARIABLE RL
2460 GOTO 2480
2470 OPEN "CS1",INTERNAL,INPUT ,FIXED RL
2480 SUBEND
2490 SUB OPENO(RL,FCODE)
2500 DATA 4," 1  Cassette 1  CS1"
2510 DATA " 2  Cassette 2  CS2"
2520 DATA " 3  Disk"
2530 DATA " 4  Other"
2540 RESTORE 2500 :: READ N :: FOR I=1 TO N :: REA
     D CHOICE$(I):: NEXT I
```

```
2550 CALL MENU("Open Output File",CHOICE$(),N,FCOD
     E)
2560 IF FCODE=0 THEN 2640
2570 IF FCODE=1 OR FCODE=2 THEN 2620
2580 DISPLAY AT(15,1):"Type device.filename"
2590 ACCEPT AT(17,1)SIZE(-28)VALIDATE(UALPHA,DIGIT
     ,".")BEEP:NM$ :: IF NM$="" THEN 2590
2600 OPEN #2:NM$,INTERNAL,OUTPUT,VARIABLE RL
2610 GOTO 2640
2620 IF FCODE=1 THEN NM$="CS1" ELSE NM$="CS2"
2630 OPEN OUTPUT #2:NM$,INTERNAL,OUTPUT,FIXED RL
2640 SUBEND
2650 SUB MENU(TITLE$,CHOICE$(),N,FUNC)
2660 C=(28-LEN(TITLE$))/2
2670 DISPLAY AT(1,C)ERASE ALL:TITLE$
2680 DISPLAY AT(4,1):"Do you want to:"
2690 R=6 :: FOR I=1 TO N
2700 DISPLAY AT(R,1):CHOICE$(I):: R=R+2
2710 NEXT I
2720 R=R+1
2730 DISPLAY AT(R,1)BEEP:"Type your selection -> "
2740 CALL KEY(0,R2,S):: IF S<>1 THEN 2740
2750 IF R2=13 THEN FUNC=0 :: GOTO 2780
2760 IF R2<49 OR R2>48+N THEN 2730
2770 FUNC=R2-48 :: DISPLAY AT(R,24)SIZE(1):CHR$(R2
     )
2780 SUBEND
```

## Program 4-2. Tiny Plan 2

```
100 REM TINY PLAN 2
110 ON BREAK NEXT :: ON WARNING NEXT
120 NL=11 :: MAXROW,MAXCOL=10 :: MAXCALC=15
130 DIM MODEL(10,10),ROW$(10),COL$(10)
140 DIM OPLIST$(8),CHOICE$(10),CALC(15,4)
150 GOSUB 280 !INITIALIZE
160 REM  MAIN MENU
170 DATA 5," 1  Define the model"
180 DATA " 2  Run the model"
190 DATA " 3  Load model&data"
200 DATA " 4  Save model&data"
210 DATA " 5  Leave Tiny Plan"
220 RESTORE 170 :: READ N :: FOR I=1 TO N :: READ
    CHOICE$(I):: NEXT I
230 CALL MENU("Tiny Plan 2",CHOICE$(),N,FCODE)
240 IF FCODE=0 THEN 150
250 IF FCODE=5 THEN 270
260 ON FCODE GOSUB 380,1400,2860,2950 :: GOTO 220
270 CALL CLEAR :: END
```

```
280 REM INITIALIZE
290 DISPLAY AT(12,1)ERASE ALL:"Initializing Tiny P
    lan": :"Just a moment ..."
300 DATA 8,+,-,*,/,%,%+,%-,%D
310 RESTORE 300 :: READ NOPR :: FOR I=1 TO NOPR ::
     READ OPLIST$(I):: NEXT I
320 FOR I=0 TO MAXROW-1 :: ROW$(I)="R"&STR$(I):: N
    EXT I
330 FOR I=0 TO MAXCOL-1 :: COL$(I)="C"&STR$(I):: N
    EXT I
340 ROW$(MAXROW)="COL TOTAL" :: COL$(MAXCOL)="ROW
    TOTAL"
350 FOR I=0 TO MAXROW :: FOR J=0 TO MAXCOL :: MODE
    L(I,J)=0 :: NEXT J :: NEXT I
360 FOR I=0 TO MAXCALC :: FOR J=0 TO 4 :: CALC(I,J
    )=0 :: NEXT J :: NEXT I
370 RETURN
380 REM DEFINE MODEL
390 DATA 3," 1  Give row names"
400 DATA " 2  Give column names"
410 DATA " 3  State calculation rules"
420 RESTORE 390 :: READ N :: FOR I=1 TO N :: READ
    CHOICE$(I):: NEXT I
430 CALL MENU("Define the model",CHOICE$(),N,FCODE
    )
440 IF FCODE=0 THEN 460
450 ON FCODE GOSUB 470,580,690 :: GOTO 420
460 RETURN
470 REM ROW NAME
480 DISPLAY AT(1,8)ERASE ALL:"Give Row Names"
490 GOSUB 1160
500 FOR I=0 TO MAXROW-1
510 DISPLAY AT(3,1):"Row ";I;" of ";MAXROW
520 DISPLAY AT(8,3)SIZE(10):ROW$(I)
530 GOSUB 1210 !READ SCREEN
540 IF NM$="END" OR NM$="end" THEN 570
550 ROW$(I)=NM$
560 NEXT I
570 RETURN
580 REM COL NAME
590 DISPLAY AT(1,6)ERASE ALL:"Give Column Names"
600 GOSUB 1160
610 FOR I=0 TO MAXCOL-1
620 DISPLAY AT(3,1):"Column ";I;" of ";MAXCOL
630 DISPLAY AT(8,3)SIZE(10):COL$(I)
640 GOSUB 1210 !READ SCREEN
650 IF NM$="END" OR NM$="end" THEN 680
660 COL$(I)=NM$
670 NEXT I
```

```
680 RETURN
690 REM CALCULATION RULES
700 DISPLAY AT(1,2)ERASE ALL:"State Calculation Ru
    les"
710 DISPLAY AT(7,1):"Row(R) or Col(C) or Quit(Q)?"
720 DISPLAY AT(8,3):"->"
730 DISPLAY AT(10,1):"1st row/column name"
740 DISPLAY AT(11,3):"->"
750 DISPLAY AT(13,1):"Calculation:"
760 DISPLAY AT(14,3):"->"
770 DISPLAY AT(16,1):"2nd row/column name"
780 DISPLAY AT(17,3):"->"
790 DISPLAY AT(19,1):"Answer row/column name"
800 DISPLAY AT(20,3):"->"
810 DISPLAY AT(23,1):"Calculations can be:"
820 DISPLAY AT(24,1):"+ - * / % %+ %- %D"
830 FOR K=0 TO MAXCALC
840 DISPLAY AT(4,1):"Step ";K;" of ";MAXCALC
850 IF CALC(K,0)=2 THEN 900
860 T$="R" :: T1$=ROW$(CALC(K,1)):: T2$=OPLIST$(CA
    LC(K,2))
870 T3$=ROW$(CALC(K,3)):: T4$=ROW$(CALC(K,4))
880 GOTO 910
890 T$="C" :: T1$=COL$(CALC(K,1)):: T2$=OPLIST$(CA
    LC(K,2))
900 T3$=COL$(CALC(K,3)):: T4$=COL$(CALC(K,4))
910 DISPLAY AT(8,5)SIZE(1):T$
920 DISPLAY AT(11,5)SIZE(10):T1$
930 DISPLAY AT(14,5)SIZE(2):T2$
940 DISPLAY AT(17,5)SIZE(10):T3$
950 DISPLAY AT(20,5)SIZE(10):T4$
960 ACCEPT AT(8,5)SIZE(-1)VALIDATE("RCQ")BEEP:T$ :
    : IF T$="" THEN 960 ELSE IF T$="Q" THEN 1140
970 IF T$="R" THEN CALC(K,0)=1 ELSE CALC(K,0)=2
980 ACCEPT AT(11,5)SIZE(-10)BEEP:NM$ :: IF NM$=""
    THEN 980
990 IF T$="R" THEN GOSUB 1250 ELSE GOSUB 1300
1000 IF N=-1 THEN 980
1010 CALC(K,1)=N
1020 ACCEPT AT(14,5)SIZE(-2)VALIDATE("+-*/%D")BEEP
     :NM$ :: IF NM$="" THEN 1020
1030 GOSUB 1350 :: IF N=-1 THEN 1020
1040 CALC(K,2)=N
1050 ACCEPT AT(17,5)SIZE(-10)BEEP:NM$ :: IF NM$=""
      THEN 1050
1060 IF T$="R" THEN GOSUB 1250 ELSE GOSUB 1300
1070 IF N=-1 THEN 1050
1080 CALC(K,3)=N
1090 ACCEPT AT(20,5)SIZE(-10)BEEP:NM$ :: IF NM$=""
      THEN 1090
```

```
1100 IF T$="R" THEN GOSUB 1250 ELSE GOSUB 1300
1110 IF N=-1 THEN 1090
1120 CALC(K,4)=N
1130 NEXT K :: GOTO 1150
1140 FOR J=0 TO 4 :: CALC(K,J)=0 :: NEXT J
1150 RETURN
1160 REM SHOW NAME SCREEN
1170 DISPLAY AT(5,1):"What name do you want?":"Typ
     e 'END' when done."
1180 DISPLAY AT(8,1):"->{10 SPACES}<-"
1190 DISPLAY AT(23,1):"Hit ENTER to use":"the info
     rmation shown."
1200 RETURN
1210 REM READ NAME SCREEN
1220 ACCEPT AT(8,3)SIZE(-10)BEEP:NM$ :: IF NM$=""
     THEN 1220
1230 IF NM$="END" OR NM$="end" THEN 1240
1240 RETURN
1250 REM ROW #
1260 FOR I=0 TO MAXROW-1
1270 IF NM$=ROW$(I)THEN N=I :: GOTO 1290
1280 NEXT I :: N=-1
1290 RETURN
1300 REM COL #
1310 FOR I=0 TO MAXCOL-1
1320 IF NM$=COL$(I)THEN N=I :: GOTO 1340
1330 NEXT I :: N=-1
1340 RETURN
1350 REM OPERATOR #
1360 FOR I=1 TO NOPR
1370 IF NM$=OPLIST$(I)THEN N=I :: GOTO 1390
1380 NEXT I :: N=-1
1390 RETURN
1400 REM RUN MODEL
1410 DATA 4," 1  Review or change data"
1420 DATA " 2  Do the calculations"
1430 DATA " 3  Print the results"
1440 DATA " 4  Save data for plotting"
1450 RESTORE 1410 :: READ N :: FOR I=1 TO N :: REA
     D CHOICE$(I):: NEXT I
1460 CALL MENU("Run the Model",CHOICE$(),N,FCODE)
1470 IF FCODE=0 THEN 1490
1480 ON FCODE GOSUB 1500,1840,2160,2570 :: GOTO 14
     50
1490 RETURN
1500 REM DISPLAY & CHANGE
1510 DISPLAY AT(1,4)ERASE ALL:"Review or Change Da
     ta"
1520 IMAGE "######.##  ######.##"
```

```
1530 DISPLAY AT(22,1):"S,D  Scroll columns"
1540 DISPLAY AT(23,1):"E,X  Scroll rows"
1550 DISPLAY AT(24,1):"C{4 SPACES}Change data
     {5 SPACES}Q  Quit"
1560 RWB,CW=0
1570 REM WINDOW
1580 RW=RWB :: CT=CW+1 :: IF CT>MAXCOL THEN CT=1
1590 DISPLAY AT(3,9)SIZE(9):COL$(CW)
1600 DISPLAY AT(3,20)SIZE(9):COL$(CT)
1610 FOR I=5 TO 5+NL-1
1620 DISPLAY AT(I,1)SIZE(7):ROW$(RW)
1630 DISPLAY AT(I,9):USING 1520:MODEL(RW,CW),MODEL
     (RW,CT)
1640 RW=RW+1 :: IF RW>MAXROW THEN 1660
1650 NEXT I
1660 FOR K=I+1 TO 19 :: DISPLAY AT(K,1):" " :: NEX
     T K
1670 REM POLL KBD
1680 CALL SOUND(100,440,4)
1690 CALL KEY(0,R,S):: IF S<>1 THEN 1690 ELSE R$=C
     HR$(R)
1700 IF R$="S" THEN CW=CW-1 :: IF CW<0 THEN CW=MAX
     COL
1710 IF R$="D" THEN CW=CW+1 :: IF CW>MAXCOL THEN C
     W=0
1720 IF R$="E" THEN RWB=RWB-NL :: IF RWB<0 THEN RW
     B=0
1730 IF R$="X" THEN RWB=RWB+NL :: IF RWB>MAXROW TH
     EN RWB=0
1740 IF R$="Q" THEN 1830
1750 IF R$<>"C" THEN 1570
1760 REM READ SCREEN
1770 RW=RWB
1780 FOR I=5 TO 5+NL-1
1790 ACCEPT AT(I,9)SIZE(-9)VALIDATE(NUMERIC)BEEP:N
1800 MODEL(RW,CW)=N
1810 RW=RW+1 :: IF RW>MAXROW THEN 1670
1820 NEXT I :: GOTO 1570
1830 RETURN
1840 REM DO CALC
1850 FOR I=0 TO MAXCALC
1860 IF CALC(I,0)=0 THEN 1950
1870 DISPLAY AT(24,1):"Working. Step ";I
1880 IF CALC(I,0)=2 THEN 1920
1890 R1=CALC(I,1):: N=CALC(I,2):: R2=CALC(I,3):: R
     3=CALC(I,4)
1900 FOR K=0 TO MAXCOL-1 :: C1,C2,C3=K :: GOSUB 20
     40 :: NEXT K
1910 GOTO 1940
```

```
1920 C1=CALC(I,1):: N=CALC(I,2):: C2=CALC(I,3):: C
     3=CALC(I,4)
1930 FOR K=Ø TO MAXROW-1 :: R1,R2,R3=K :: GOSUB 2Ø
     4Ø :: NEXT K
1940 NEXT I
1950 DISPLAY AT(24,1):"Row and column totals."
1960 FOR I=Ø TO MAXROW :: T=Ø :: FOR J=Ø TO MAXCOL
     -1
1970 T=T+MODEL(I,J):: NEXT J
1980 MODEL(I,MAXCOL)=T :: NEXT I
1990 FOR I=Ø TO MAXCOL :: T=Ø :: FOR J=Ø TO MAXROW
     -1
2000 T=T+MODEL(J,I):: NEXT J
2010 MODEL(MAXROW,I)=T :: NEXT I
2020 DISPLAY AT(24,1):" "
2030 RETURN
2040 REM CALC
2050 ON N GOTO 2060,2070,2080,2090,2110,2120,2130,
     2140
2060 MODEL(R3,C3)=MODEL(R1,C1)+MODEL(R2,C2):: RETU
     RN
2070 MODEL(R3,C3)=MODEL(R1,C1)-MODEL(R2,C2):: RETU
     RN
2080 MODEL(R3,C3)=MODEL(R1,C1)*MODEL(R2,C2):: RETU
     RN
2090 IF MODEL(R2,C2)<>Ø THEN MODEL(R3,C3)=MODEL(R1
     ,C1)/MODEL(R2,C2)
2100 RETURN
2110 MODEL(R3,C3)=MODEL(R1,C1)*MODEL(R2,C2)/100 ::
      RETURN
2120 MODEL(R3,C3)=MODEL(R1,C1)+(MODEL(R1,C1)*MODEL
     (R2,C2)/100):: RETURN
2130 MODEL(R3,C3)=MODEL(R1,C1)-(MODEL(R1,C1)*MODEL
     (R2,C2)/100):: RETURN
2140 IF MODEL(R1,C1)<>Ø THEN MODEL(R3,C3)=((MODEL(
     R2,C2)-MODEL(R1,C1))/MODEL(R1,C1))*100
2150 RETURN
2160 REM PRINT
2170 DISPLAY AT(1,6)ERASE ALL:"Print the Results"
2180 DISPLAY AT(4,1):"Starting column Ø-";MAXCOL
2190 DISPLAY AT(5,1):"Ending column{3 SPACES}Ø-";M
     AXCOL
2200 DISPLAY AT(6,1):"Starting row{4 SPACES}Ø-";MA
     XROW
2210 DISPLAY AT(7,1):"Ending row{6 SPACES}Ø-";MAXR
     OW
2220 DISPLAY AT(10,1):"Digits before decimal"
2230 DISPLAY AT(11,1):"Digits after decimal"
2240 DISPLAY AT(12,1):"(14 digits max.)"
```

```
2250 DISPLAY AT(15,1):"Spaces between columns"
2260 DISPLAY AT(21,1):"O.K. to print (Y or N)?"
2270 ACCEPT AT(4,26)SIZE(-3)VALIDATE(DIGIT)BEEP:C1
2280 ACCEPT AT(5,26)SIZE(-3)VALIDATE(DIGIT)BEEP:C2
2290 IF C1<0 OR C1>MAXCOL OR C2<0 OR C2>MAXCOL OR
     C2<C1 THEN 2270
2300 ACCEPT AT(6,26)SIZE(-3)VALIDATE(DIGIT)BEEP:R1
2310 ACCEPT AT(7,26)SIZE(-3)VALIDATE(DIGIT)BEEP:R2
2320 IF R1<0 OR R1>MAXROW OR R2<0 OR R2>MAXROW OR
     R2<R1 THEN 2300
2330 ACCEPT AT(10,26)SIZE(-2)VALIDATE(DIGIT)BEEP:D1
2340 ACCEPT AT(11,26)SIZE(-2)VALIDATE(DIGIT)BEEP:D2
2350 IF D1+D2>14 THEN 2330
2360 F$=RPT$("#",D1):: IF D2>0 THEN F$=F$&"."&RPT$
     ("#",D2)
2370 ACCEPT AT(15,26)SIZE(-3)VALIDATE(DIGIT)BEEP:SP
2380 IF SP>30 THEN 2370
2390 L=LEN(F$)
2400 PW=10+SP+(C2-C1+1)*(L+SP):: FW=L+SP
2410 DISPLAY AT(18,1):"Print width is ";PW
2420 ACCEPT AT(21,28)SIZE(-1)VALIDATE("YN")BEEP:R$
      :: IF R$="" THEN 2420 ELSE IF R$="N" THEN 22
     70
2430 REM * OPEN PRINTER *
2440 OPEN #1:"RS232.BA=1200",DISPLAY ,OUTPUT,VARIA
     BLE PW
2450 K=0 :: FOR C=C1 TO C2
2460 PRINT #1:TAB(10+SP+K*FW);COL$(C);:: K=K+1
2470 NEXT C
2480 PRINT #1 :: PRINT #1
2490 FOR R=R1 TO R2 :: PRINT #1:ROW$(R);:: K=0
2500 FOR C=C1 TO C2
2510 PRINT #1:TAB(10+SP+K*FW);
2520 PRINT #1,USING F$:MODEL(R,C);:: K=K+1
2530 NEXT C :: PRINT #1 :: NEXT R
2540 PRINT #1 :: PRINT #1
2550 CLOSE #1
2560 RETURN
2570 REM SAVE FOR PLOT
2580 DISPLAY AT(1,4)ERASE ALL:"Save Data for Plott
     ing"
2590 DISPLAY AT(4,1):"Plot rows(R) or columns(C)"
2600 DISPLAY AT(7,1):"X-axis (horizontal)"
2610 DISPLAY AT(9,7):"number"
2620 DISPLAY AT(12,1):"Y-axis (vertical)"
2630 DISPLAY AT(14,7):"number"
2640 DISPLAY AT(22,1):"Row number range 0-";MAXROW
2650 DISPLAY AT(23,1):"Col number range )-";MAXCOL
2660 ACCEPT AT(4,28)SIZE(1)VALIDATE("RC")BEEP:T$ :
     : IF T$="" THEN 2660
```

```
2670 IF T$="R" THEN T1$="Row" ELSE T1$="Col"
2680 DISPLAY AT(9,3)SIZE(3):T1$ :: DISPLAY AT(14,3
     )SIZE(3):T1$
2690 ACCEPT AT(9,27)SIZE(-2)VALIDATE(NUMERIC)BEEP:
     D1 :: IF D1<0 THEN 2690
2700 IF T$="R" AND D1>MAXROW THEN 2690
2710 IF T$="C" AND D1>MAXCOL THEN 2690
2720 ACCEPT AT(14,27)SIZE(-2)VALIDATE(NUMERIC)BEEP
     :D2 :: IF D2<0 THEN 2720
2730 IF T$="R" AND D2>MAXROW THEN 2720
2740 IF T$="C" AND D2>MAXCOL THEN 2720
2750 CALL OPENO(64,FCODE):: IF FCODE=0 THEN 2850
2760 IF FCODE<0 THEN 2850
2770 IF T$="C" THEN 2810
2780 FOR I=1 TO MAXCOL
2790 PRINT #2:MODEL(D1,I);MODEL(D2,I)
2800 NEXT I :: GOTO 2840
2810 FOR I=0 TO MAXROW
2820 PRINT #2:MODEL(I,D1);MODEL(I,D2)
2830 NEXT I
2840 PRINT #2:-1;-1 :: CLOSE #2
2850 RETURN
2860 REM LOAD
2870 CALL OPENI(192,FCODE):: IF FCODE=0 THEN 2940
2880 INPUT #2:MAXROW,MAXCOL,MAXCALC
2890 FOR I=0 TO 10 :: INPUT #2:ROW$(I),:: NEXT I
2900 FOR I=0 TO 10 :: INPUT #2:COL$(I),:: NEXT I
2910 FOR I=0 TO MAXCALC :: FOR J=0 TO 4 :: INPUT #
     2:CALC(I,J),:: NEXT J :: NEXT I
2920 FOR I=0 TO MAXROW :: FOR J=0 TO MAXCOL :: INP
     UT #2:MODEL(I,J),:: NEXT J :: NEXT I
2930 CLOSE #2
2940 RETURN
2950 REM SAVE
2960 CALL OPENO(192,FCODE):: IF FCODE=0 THEN 3030
2970 PRINT #2:MAXROW;MAXCOL;MAXCALC
2980 FOR I=0 TO 10 :: PRINT #2:ROW$(I);:: NEXT I :
     : PRINT #2
2990 FOR I=0 TO 10 :: PRINT #2:COL$(I);:: NEXT I :
     : PRINT #2
3000 FOR I=0 TO MAXCALC :: FOR J=0 TO 4 :: PRINT #
     2:CALC(I,J);:: NEXT J :: PRINT #2 :: NEXT I
3010 FOR I=0 TO MAXROW :: FOR J=0 TO MAXCOL :: PRI
     NT #2:MODEL(I,J);:: NEXT J :: PRINT #2 :: NEX
     T I
3020 CLOSE #2
3030 RETURN
3040 SUB OPENI(RL,FCODE)
3050 DATA 3
3060 DATA " 1  Cassette 1  CS1"
```

```
3070 DATA " 2  Disk"
3080 DATA " 3  Other"
3090 RESTORE 3050 :: READ N :: FOR I=1 TO N :: REA
     D CHOICE$(I):: NEXT I
3100 CALL MENU("Open Input File",CHOICE$(),N,FCODE
     )
3110 IF FCODE=0 THEN 3180
3120 IF FCODE=1 THEN 3170
3130 DISPLAY AT(15,1):"Type device.filename"
3140 ACCEPT AT(17,1)SIZE(-28)VALIDATE(UALPHA,DIGIT
     ,".")BEEP:NM$ :: IF NM$="" THEN 3140
3150 OPEN #2:NM$,INTERNAL,INPUT ,VARIABLE RL
3160 GOTO 3180
3170 OPEN "CS1",INTERNAL,INPUT ,FIXED RL
3180 SUBEND
3190 SUB OPENO(RL,FCODE)
3200 DATA 4," 1  Cassette 1  CS1"
3210 DATA " 2  Cassette 2  CS2"
3220 DATA " 3  Disk"
3230 DATA " 4  Other"
3240 RESTORE 3200 :: READ N :: FOR I=1 TO N :: REA
     D CHOICE$(I):: NEXT I
3250 CALL MENU("Open Output File",CHOICE$(),N,FCOD
     E)
3260 IF FCODE=0 THEN 3340
3270 IF FCODE=1 OR FCODE=2 THEN 3320
3280 DISPLAY AT(15,1):"Type device.filename"
3290 ACCEPT AT(17,1)SIZE(-28)VALIDATE(UALPHA,DIGIT
     ,".")BEEP:NM$ :: IF NM$="" THEN 3290
3300 OPEN #2:NM$,INTERNAL,OUTPUT,VARIABLE RL
3310 GOTO 3340
3320 IF FCODE=1 THEN NM$="CS1" ELSE NM$="CS2"
3330 OPEN #2:NM$,INTERNAL,OUTPUT,FIXED RL
3340 SUBEND
3350 SUB MENU(TITLE$,CHOICE$(),N,FUNC)
3360 C=(28-LEN(TITLE$))/2
3370 DISPLAY AT(1,C)ERASE ALL:TITLE$
3380 DISPLAY AT(4,1):"Do you want to:"
3390 R=6 :: FOR I=1 TO N
3400 DISPLAY AT(R,1):CHOICE$(I):: R=R+2
3410 NEXT I
3420 R=R+1
3430 DISPLAY AT(R,1)BEEP:"Type your selection -> "
3440 CALL KEY(0,R2,S):: IF S<>1 THEN 3440
3450 IF R2=13 THEN FUNC=0 :: GOTO 3480
3460 IF R2<49 OR R2>48+N THEN 3430
3470 FUNC=R2-48 :: DISPLAY AT(R,24)SIZE(1):CHR$(R2
     )
3480 SUBEND
```

# Chapter 5

# Computer Graphics

# Computer Graphics

The last chapter discussed electronic spreadsheets. Perhaps you are collecting household information for which a spreadsheet approach is not really required. For example, you may simply be collecting household expenses by category. Or you might be recording gas mileage at every fill up.

Whether it was computed by a spreadsheet program or just recorded and saved, you probably can think of household information that you would like to review and study. Usually the purpose of such study is to look for trends. Are the grocery expenses going up, down, or staying about the same? Has the gas mileage on the family car worsened? Is it time for a new set of plugs and points? The telecommunications industry is changing rapidly. Our telephone bills will require closer scrutiny. Can any trends be isolated?

How do we go about drawing conclusions from all the data—from just columns of numbers? It is easy to see the smaller month-to-month variations, but ideally it's best to review an entire set of data.

In order to make the proper changes in a family's household management, the overall trends and tendencies must be studied and evaluated. This is really the only way to make changes that will be beneficial. Short-range fluctuations are to be expected, but long-range trends are needed to react appropriately. If, for example, a particular expense category has been rising steadily, you might want to take some action.

There are mathematical tools that can be used to perform this type of data analysis. Examples of such tools are curve fitting programs, descriptive statistics, and multiple linear regression analysis. These tools do a good job, but they are specialized subjects in their own right.

Computer graphics is another tool in the analyst's tool kit. With computer graphics, you can *see* your data. This visual

representation has much more impact than rows and columns of numbers. By viewing the data, you can begin making conclusions about trends and patterns very easily. In fact, professional analysts often prepare a graph of the data before they use the more sophisticated techniques. They want some idea of how the data is behaving before studying it in more detail.

### Bar Charts

There are several different types of graphs or charts:

—line graphs,
—pie charts, and
—bar charts.

There are other types, but these are the most common. Line graphs show the relationship of one data item or *variable* as a function of another. A good example of this is the familiar graph of a sine wave. Pie charts help the analyst visualize the contribution of different parts to the whole. A pie chart would be good for showing different expense categories and how they compare. Bar charts are most useful for examining and comparing data over discrete intervals such as time. For example, bar charts are ideal for displaying income on a month-to-month basis. Figure 5-1 illustrates these three graphic techniques.

This chapter will discuss bar chart programs. Almost all of the household graphs can be done quite nicely with bar charts. Your TI can produce very effective bar chart displays on your television or monitor.

The TI-99/4A hardware does have a high-resolution graphics mode. This mode enables the TI to produce line graphs and pie charts as well. However, specialized machine language software is required. Unfortunately, such software is beyond the scope of this book.

### System Requirements

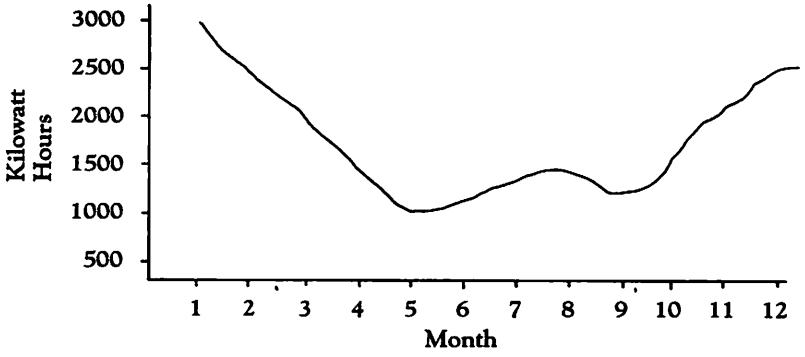
There are two programs in this chapter:

—Bar Charts, and
—Bar Charts 2.

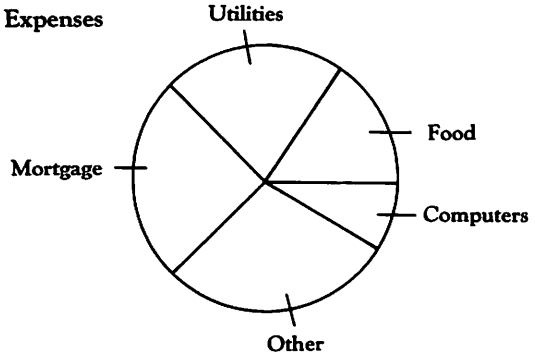
All of these programs have been designed for a 16K TI-99/4A and Extended BASIC. Figure 5-2 shows the equipment that you'll need for the two bar chart programs.

## Figure 5-1. Types of Graphs

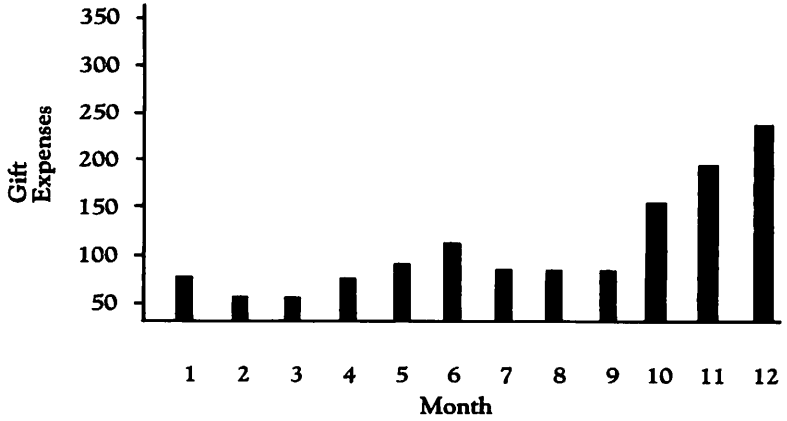
### Line Graph




### Pie Chart




### Bar Chart



You will probably find yourself using Bar Charts most often. It produces the charts on your TV or monitor. Charts may contain up to 15 vertical and 12 horizontal intervals. If you have a printer, Bar Charts can print a copy of the chart. However, no extra equipment is really needed.

Bar Charts 2 can produce charts with a great deal more detail than Bar Charts. The resolution of Bar Charts 2 is 25 vertical and 35 horizontal intervals. These charts are too large for a video display. Bar Charts 2 requires a printer for proper operation.

In all other respects, Bar Charts and Bar Charts 2 are identical. The operating instructions apply to both of these programs unless stated otherwise.

### Figure 5-2. Bar Charts and Bar Charts 2 System Requirements

#### Bar Charts

Required:
- —TI-99/4A console
- —Extended BASIC
- —Cassette tape recorder

Optional:
- —32K memory expansion
- —RS232 interface
- —Printer
- —Disk drive
- —Disk controller

#### Bar Charts 2

Required:
- —TI-99/4A console
- —Extended BASIC
- —Cassette tape recorder
- —RS232 interface
- —Printer

Optional:
- —32K memory expansion
- —Disk drive
- —Disk controller

## Bar Charts General Operation

Bar Charts is a menu-driven program. The particular techniques employed require you to type in your selection and then hit ENTER. In other words, Bar Charts does not use a single-keystroke approach.

The main menu screen lists the major functions that are available:

1 Enter data,
2 Change data,
3 Save data,
4 Display data, and
5 End bar charts.

Notice that the first three selections deal with data manipulation. In fact, Bar Charts contains its own built-in data manager. This means that you do not need a separate program to record data and to fix data when mistakes have been made. You can do all this with Bar Charts. Data management is such an important topic that it will be treated separately.

Selection 4 is used to create a bar graph from data. Once selection 4 is chosen, another display screen will appear and ask for chart titles and various scaling options.

The final option, selection 5, puts you back in the BASIC environment. As with the other programs in this book, you must be careful. Make sure that all data is properly saved on tape or disk before ending the program.

## Data Management

Before a bar chart can be produced, the data for the chart must be stored in the computer's memory. Bar Chart expects the data in a certain format.

The data is always stored as X-Y pairs. The first number, or X, represents a position along the X or horizontal axis. Likewise, Y, the second number, represents a position on the Y or vertical axis. Each such pair of numbers is called an *observation.*

Let's look at an example. The expenses for the first three months of the year were:

January 642.10
February 531.29
March 589.67

A pair of numbers are needed for Bar Charts. In this

example the month number will be used for X and the expense amount for Y:

1 642.10
2 531.29
3 589.67

Thus, there are three observations or three pairs of numbers. The month's number will often be used as the X variable. There are quite a few household transactions that will be tracked monthly. If daily information is desired, the day of the month will be X. Occasionally, circumstances may arise in which the quarter number of the year could be used.

Not every bar chart will use time as the X axis. Suppose you are doing a bit of home energy analysis and want a chart that will show electricity consumption for the lowest temperature in a 24-hour period. The temperature reading could go on the X axis and the kilowatt hours on the Y axis.

Regardless of the actual situation, Bar Charts will be happy as long as the data is arranged as X-Y pairs.

### Data Management: Enter Data

How is data entered into the computer? Selection 1 on the main menu invokes the Enter Data function. Use this function when you want to put data in the computer.

The data entry screen will present you with another choice. The data may come from:

K Keyboard,
T Tape, or
D Disk.

Choose the appropriate option. If you are entering observations for the first time, the keyboard will be the source of input. Select tape or disk if you've previously saved the observations and want them read in again.

Assume that the keyboard is the input device. Bar Charts will

—display the current observation number,
—ask for the X variable,
—ask for the Y variable, and
—ask if there is another observation.

This process repeats until you tell bar charts that you've typed in all the observations.

Observations entered by the keyboard are always stored after any observations that may already be in memory. This means that keyboard data entry acts as an append function. We may return to the Enter Data function and type in additional observations whenever necessary.

Tape and disk input follow the standard TI procedures. When you are using disk input, you will be asked for a filename such as "DSK1.EXPENSES".

Tape and disk data entry functions do not work in the append mode. When tape or disk is specified as the input device, all of the observations in memory are cleared. Observations from tape or disk start with observation number one.

An observation of $-1,-1$ for tape and disk input signals end of file. This tells Bar Charts to turn off the tape recorder or stop reading from the disk. Both Bar Charts and Tiny Plan automatically write a $-1,-1$ as the last record on a file. Any of your own programs that will transfer data to Bar Charts should do the same thing.

The main menu screen will be displayed when data entry is completed.

### Data Management: Change Data

When you type in a set of observations, the data is not locked away forever in some memory chip. Perhaps you've spotted something unusual on a bar chart. You would like to go back and review the observations. Perhaps you'll discover that grocery expenses were very high one month because someone typed in the mortgage payment by accident. Perhaps nothing is wrong with the data at all. There must be another explanation.

Bar Charts provides the facilities for examining and changing data. These functions are invoked by means of selection 2 on the main menu.

Bar Charts will display each observation one at a time. The cursor will be placed at the X variable and then the Y variable. You may type in a new value for either X or Y. If you want the current values, just hit ENTER.

Bar Charts will ask you what you want to do next. There are several choices:

D Delete this observation,
S See this observation,

N See next observation,
P See previous observation, and
Q Quit.

D will delete an observation. The observation is removed from memory. Then, of course, the observation will not appear on any bar charts. The delete function is a handy way of taking care of duplicate entries that sometimes occur.

The S function takes you to a specific observation. Bar Charts always starts out by showing the first observation. What if the 50th observation requires correction? With this function you can avoid the time-consuming process of going observation by observation until we arrive at the 50th one. Bar Charts will ask for an observation number. If you type 50, the 50th observation will be displayed. Then you can make the needed corrections.

N and P are often used together. N moves the display forward one observation. P moves the display backward one observation. By using these two functions together, you do not need to know exactly which observation had an error, but rather its approximate location. You could use S to position the display at observation 45, then use N and P to go forward and backward until you home in on the problem observation.

The Q option will return the program to the main menu screen.

Notice that the data change functions are designed only for changing and deleting observations. You cannot add observations at this point in Bar Charts. To add observations, go back to the Enter Data option of the main menu.

All of these functions presume that there are some observations on which to work. If, by chance, you try to change data before typing any in, Bar Charts will say,

* NO OBSERVATIONS ENTERED *

You will be quickly escorted back to the main menu.

### Data Management: Save Data

In most instances, a permanent record of the data is important. This is true if the data is being collected periodically rather than all at once.

Selection 2 from the main menu invokes the save data function. The observations can be saved on one of several devices:

N Null device,
1 Cassette unit #1,
2 Cassette unit #2, and
D Disk

The null device is not a new TI peripheral. It is just an escape mechanism. Maybe, on reflection, you really aren't quite ready for saving the data yet. If you select the Null option, the data is *not* saved. It is simply a way to get back to the main menu.

The other device options are standard in their approach. Just follow the instructions that appear on the screen. If you are using the disk option, you will be asked for a filename.

As part of the data save operation, Bar Charts will automatically write a $-1,-1$ as the last observation. This marks the end of the file. When the save operation is completed, Bar Charts will display the main menu screen again.

### Display Data

The next major function on the main menu screen is Selection 4—display data. This is where you will see results of the data displayed graphically.

Selection 4 displays a full screen on which you can specify many options. These options determine how the bar charts will look. This is the information that you may wish to add to the display:

—chart titles,
—type of bar chart, and
—X and Y axis scales.

If you wish, you can ignore all of these options. Whenever Bar Charts asks for information, just hit ENTER. The bar charts will still be displayed on the screen and printed if you are using the print option.

In the following sections, we will focus on ways of customizing the bar charts by using the display options.

### Chart Titles

A bar chart without titles is fine for casual use. However, a title is also one of the ways that a chart conveys information. If you are showing the chart to others or if you must reference it yourself at a later time, you'll want to include a title.

With Bar Charts, you can specify:

— the title of the chart,
— the X axis label, and
— the Y axis label.

Each of these titles or labels may contain up to 28 characters of information. Upper- and lowercase letters may be freely mixed.

The chart title should give a good idea of the information that is being conveyed. Here are a few examples of chart titles:

Daily expenses for Aug. 1983,
1982 monthly kilowatt hours, and
1983 gas mileage—Escort.

Similarly, the X and Y axis should be labeled with explanatory information. Frequently, the X axis represents time. So an X axis label might identify the specific units of time involved, such as days or months. Generally, the Y axis label will state the units of the quantity that is being charted. For example, the Y axis label might say

Expense dollars,
Kilowatt hours, or
Miles per gallon.

### Chart Type

Bar Charts will do three kinds of computations, one at a time, on the data, and then display the results. The computations are:

—average the figures within each interval,
—sum the figures within each interval, and
—count the occurrences within each interval.

The particular selection that you choose governs the type of bar chart that you will see. Bar Charts will ask you what type of chart you want. It will ask you if you want:

—a sum,
—an average, or
—a frequency chart.

These three types of charts correspond to the three types of data computations.

An example will illustrate this. Expense monitoring is a common household application. Let's say that you at least keep a record of your expenses. You write down the data of

the expense, the amount, and the category. An excerpt from the log might contain:

| | | |
|---|---|---|
| 12/1 | 50.00 | groceries |
| 12/1 | 12.50 | gas |
| 12/1 | 5.00 | meals |
| 12/2 | 100.00 | utilities |
| 12/2 | 10.00 | contribution |

Let's take a look at the possible bar chart variations. First of all, the day number will always be placed along the X or horizontal axis. If a sum chart is desired, Bar Charts will add up the expenses for each day. The total expenses for December 1 are $67.50, and for December 2 they are $110.00. Bar Charts will graph 67.50 in the first interval and 110.00 in the second. The sum chart gives a day-by-day record of total expenses.

When an average chart is requested, Bar Charts will show the average expenses for each day. Using the above figures, the average for December 1 is $22.50, and the average for December 2 is $55.00. This type of chart does not seem too useful for budget applications. What can be seen from average expenses? There are many high amounts (mortgage payments) and low amounts (chewing gum). The average, therefore, tends to be just a hodgepodge.

On the other hand, the average chart can come in handy for other applications. A good example is an application where periodic meter readings are being taken. The readings are taken several times and then an average is computed. You might want to try this yourself. Weigh yourself at different times of the day, over a period of several days. Write down the scale readings. Each evening, type in the day's weights. Let Bar Charts compute your average daily weight. Look at the resulting bar chart. Can you draw any conclusions?

The frequency chart is the third type. Bar Charts counts how many numbers or readings fall into each X axis interval. This count is then graphed. In the expense example, there were three expense transactions on December 1 and two on December 2. This particular type of chart can give useful insight into buying patterns. Does the number of expense transactions diminish at the end of a pay period? What happens after a payday? Of course, there will be some seasonal influence in such records.

The chart options give a great deal of flexibility. You can type in a set of data and have it displayed three different ways. Best of all, no extra programming is required.

## X and Y Axis Scales

Bar Charts and Bar Charts 2 produce charts with different levels of detail.

| Intervals | Bar Charts | Bar Charts 2 |
|---|---|---|
| X axis | 12 | 35 |
| Y axis | 15 | 25 |

The number of X axis intervals governs the number of bars that will be on a bar chart. Thus, there can be up to 12 bars with Bar Charts and 35 with Bar Charts 2. The number of Y axis intervals determines a charting program's ability to show differences between numbers that are close together in magnitude. Generally speaking, there can be a finer degree of detail with a larger number of Y axis intervals.

Both of these programs have a certain specific number of X and Y intervals. Does this mean that they are limited in the range of numbers that they can display? Consider Bar Charts. The X axis is limited to 12 intervals. This is fine for charting monthly data. The month number will range from 1 to 12. But, what about daily information? Is Bar Charts lost when you try to display numbers from 1 to 31 along its X axis? Of course not.

Bar Charts and Bar Charts 2 use a process called *scaling*. Scaling is a mathematical technique. It makes sure that your data, no matter what size the numbers are, can be properly displayed.

Here is how the scaling process works. The data is examined. The largest X axis number and the largest Y axis number are located. The number of X axis intervals is divided by the largest X axis number. The same thing is done for the Y axis. The quotient, then, becomes the size of each interval. Consider the display of daily information with Bar Charts again. The largest day number is 31. There are 12 X axis intervals. Therefore, each bar, or interval, represents 31/12 or 2.58 days.

Scaling is completely automatic. The calculations are done before the bar charts are displayed. The programs scale the data so that the charts will fill the available X and Y intervals as much as possible. In other words, you will obtain the largest chart that the programs can produce.

The scaling operations can produce results that are awkward to use. How do you interpret a bar chart where each bar counts for 2.58 days? And you can get equally strange factors for the Y axis.

There is a way around this. You can turn off the automatic scaling calculations. You can tell the programs what factors you want used for scaling. How do you know which factors should be used? Well, let Bar Charts run in the automatic mode first. Look at the scaling factors that it has chosen. Then, adjust them appropriately by rounding them up to the nearest integer, for example. Using the above illustration, you might try using 3 days instead of 2.58 days for the X axis interval. It's much easier to analyze units of 3 whole days. You could also try units of 5 or 7 days, although this will reduce the number of bars on the chart.

### Scale Options

There are six options—three for the X axis and three for the Y axis—that you can use for controlling chart scaling. These options are:

X axis
- —number of intervals
- —computer or manual scaling
- —interval size

Y axis
- —number of intervals
- —computer or manual scaling
- —interval size

Notice that you can control the number of intervals as well as the interval size.

These six options have standard values, called *defaults,* which the computer will use unless you specify differently. The default values are shown on the screen. If the default value is acceptable to you, press ENTER when the cursor points to it. Otherwise, type in the value that you want.

When would you want to vary the number of chart intervals? Suppose you are collecting and graphing monthly data. At the end of March, three months of data are available. Tell Bar Charts that there are only three X axis intervals. The data for January, February, and March will be plotted properly in intervals 1, 2, and 3.

In addition to the number of intervals, you may specify

computer or manual scaling. Computer scaling is the default. In this case, the X and Y interval sizes are calculated as previously discussed. If you specify manual scaling, you must then give the interval size that should be used. You can also plot the three months' data another way, by asking for manual scaling and an interval size of one.

Manual scaling is also handy for adjusting Y axis intervals. If expenses are being plotted, you would like each Y axis interval to be a convenient amount such as $100 or $500 rather than a computed amount such as $237.82.

### Bar Chart Examples

Figures 5-3 and 5-4 show the types of charts that can be obtained. These samples also illustrate the effects of the scaling options.

Both examples were produced using the data shown below.

| X | Y | X | Y |
|---|---|---|---|
| 1 | 1 | 6 | 6 |
| 2 | 2 | 7 | 7 |
| 3 | 3 | 8 | 8 |
| 4 | 4 | 9 | 9 |
| 5 | 5 | 10 | 10 |

This is the familiar Y=X function. It produces a line at a 45 degree angle when plotted on graph paper.

Look at Figure 5-3. This bar chart was produced using the default scaling options—computer scaling. Notice that the data is stretched so that it fills all of Bar Chart's display area. Notice, too, that the scaling factors are not integers.

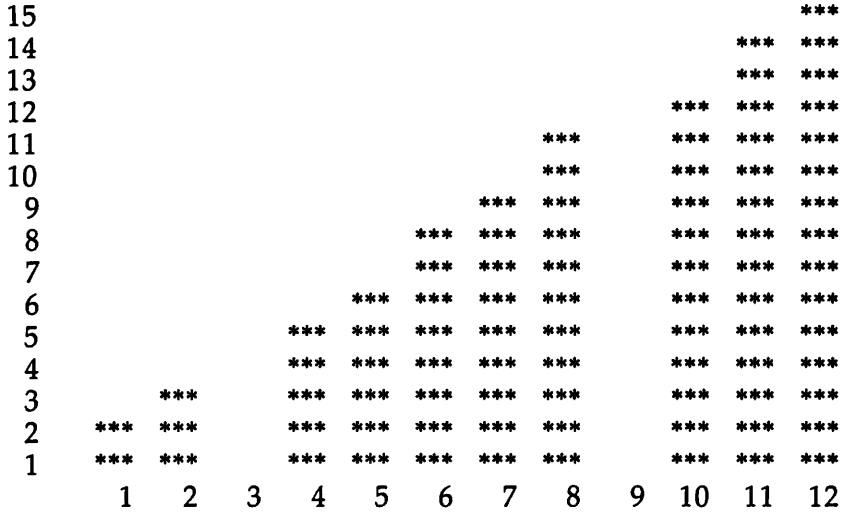
The same data is displayed in Figure 5-4 but uses our own scaling factors. The X and Y interval sizes were set at 1.0. Notice the differences in the charts. Does Figure 5-4 represent what you expected to see in the first place?

These examples serve another purpose as well. Try graphing this same set of data. If you've typed in Bar Charts properly, you should be able to duplicate the examples. Thus, this set of simple data acts as a calibration mechanism.

### Display Data Summary

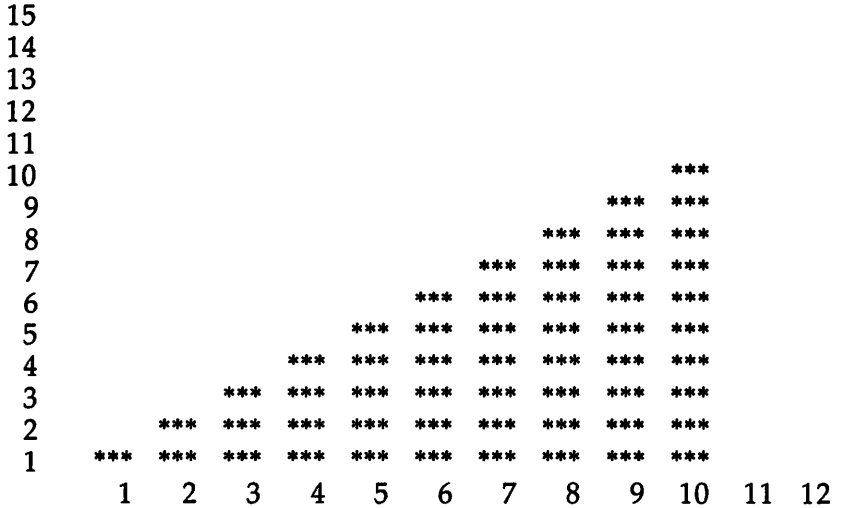
Selection 4 of the main menu screen invokes the bar chart display function. You have a great deal of control over the

**Figure 5-3. Sample Bar Charts Computer Scaling**




X-axis: INTEGERS 1–10 Scale: .83
Y-axis: INTEGERS 1–10 Scale: .67

**Figure 5-4. Sample Bar Charts Manual Scaling**




X-axis: INTEGERS 1–10 Scale: 1.00
Y-axis: INTEGERS 1–10 Scale: 1.00

ultimate format of the bar chart. You may specify:

—the chart titles,
—the type of bar chart, and
—the X and Y axis scales.

Although there are many options, Bar Charts has default values for all of them. You can obtain your initial charts by using them. Then you can dress up the bar chart by adding titles and adjusting the scaling factors.

### Print Bar Charts

Both Bar Charts and Bar Charts 2 have the capability of printing the charts. Bar Charts will display a chart on your TV or video monitor. After the chart is displayed, you may press the P key. Doing so will produce a printed copy of the chart.

Bar Charts 2 does not display its charts. After you have finished completing the chart options, Bar Charts 2 will begin printing a copy of the chart.

Both programs handle the printing the same way. They ask you for the name of an output device. You should respond with "RS232" plus the appropriate options if you are using a serial printer. If you are using a parallel printer, respond with PIO.

Why not put the device name in the program? These programs illustrate another programming technique. Do you have to respond with the printer's device name? No. Suppose you specified a disk file as the output device. Well, Bar Charts will send the chart to disk. What good will it do there? Other programs can then access the charts. A prime example of this is word processing. You can produce a chart with Bar Charts, direct it to disk, and later include it in a document. This is very nice, indeed.

Bar Charts and Bar Charts 2 assume you have a standard 80-column printer. No special graphics characters are used. If you have a printer with graphics capability, you may want to enhance the chart-printing routines.

### How It Works

The structure chart for Bar Charts is shown in Figure 5-5. The labels in the structure chart boxes generally correspond to REM statements that mark the beginning of a subroutine. The main program dimensions the arrays and initializes a few variables. The main menu displays the available selections and

invokes the proper subroutine to process the selected function. Both Display Data and Enter Data invoke other functions. Bar Charts 2 has a very similar structure. The Display Axis and Display Bars subroutines have been removed. The Print Chart subroutine has been revised to work exclusively with a printer.

Figure 5-6 lists the major variables used by Bar Charts and Bar Charts 2. The only subprogram used is a simple delay loop.

**Figure 5-5. Bar Charts Program Structure**

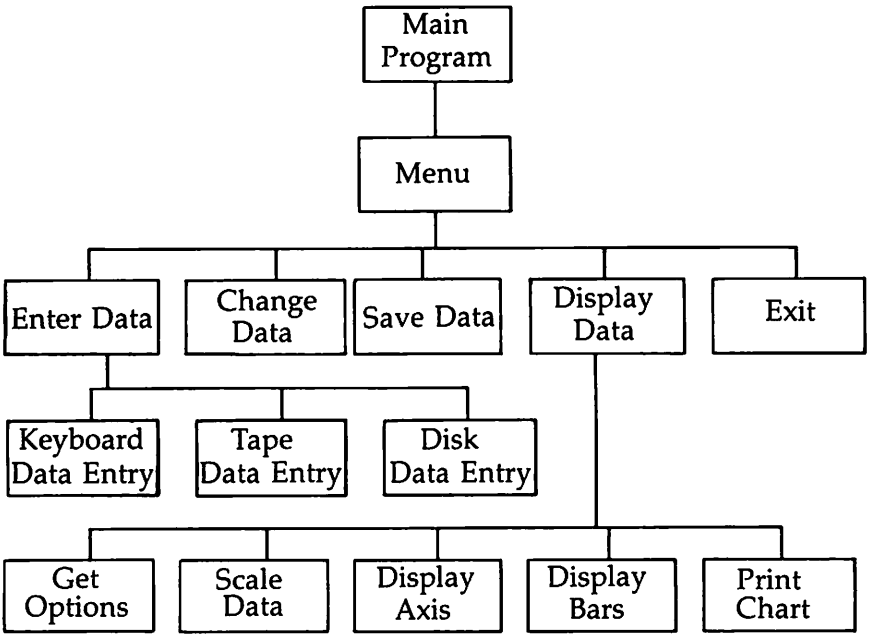



**Figure 5-6. Bar Charts and Bar Charts 2 Variables**

OBS(*n*,*n*)— This two-dimensional array contains the observations or data to be plotted. The data is stored as X,Y pairs. If you have the 32K memory expansion, you may increase the first dimension of OBS.

MAXN— The maximum number of observations. MAXN must match the first dimension of the array OBS.

XMAXINT— The maximum number of X axis intervals.
XINT— The actual number of X axis intervals.
YMAXINT— The maximum number of Y axis intervals.
YINT— The actual number of Y axis intervals.
NCOUNT(*n*)— The number of observations that falls in each X axis interval. The dimension of NCOUNT must match XMAXINT.
TOTAL(*n*)— The sum of the Y values of the observations that fall in each X axis interval. The dimension of TOTAL must match XMAXINT.
AVERAGE(*n*)— The average value of the Y values of the observations that fall in each X axis interval. The dimension of AVERAGE must match XMAXINT.

## Program 5-1. Bar Charts1

```
100 REM BAR CHARTS
110 ON WARNING NEXT :: ON BREAK NEXT
120 OPTION BASE 1
130 MAXN=100 :: XMAXINT,XINT=12 :: YMAXINT,YINT=15
140 DIM ORS(100,2),NCOUNT(12),TOTAL(12),AVERAGE(12
    )
150 GOSUB 2550 !INIT
160 GOSUB 2610 !MENU
170 IF FINI=0 THEN 160
180 GOSUB 2770 !END
190 END
200 REM ENTER DATA
210 GOSUB 320 !ENTER MENU
220 IF INPDEV$="" THEN 210
230 IF INPDEV$="K" THEN GOSUB 420
240 IF INPDEV$="T" THEN GOSUB 600
250 IF INPDEV$<>"D" THEN 310
260 DISPLAY AT(13,3):"FILE NAME "
270 DISPLAY AT(15,3):FILENAME$
280 ACCEPT AT(15,3)SIZE(-15)VALIDATE(UALPHA,".",DI
    GIT)BEEP:FILENAME$
290 IF FILENAME$="" THEN 280
300 GOSUB 720
310 RETURN
320 REM ENTER MENU
330 CALL CLEAR
340 DISPLAY AT(2,7):"** Enter Data **"
350 DISPLAY AT(5,1):"Read data from:"
360 DISPLAY AT(7,3):"K  Keyboard"
```

```
370 DISPLAY AT(9,3):"T  Tape"
380 DISPLAY AT(11,3):"D  Disk"
390 DISPLAY AT(22,1):"Type your selection"
400 ACCEPT AT(22,25)BEEP SIZE(1)VALIDATE("KTD"):IN
    PDEV$
410 RETURN
420 REM KEYBOARD ENTRY
430 CALL CLEAR
440 DISPLAY AT(2,1):"** Keyboard Data Entry **"
450 DISPLAY AT(5,1):"Observation number"
460 DISPLAY AT(7,3):"Value for X"
470 DISPLAY AT(9,3):"Value for Y"
480 DISPLAY AT(22,1):"Press 'Q' to quit;"
490 DISPLAY AT(23,7):"ENTER to continue."
500 FOR I=NOBS+1 TO MAXN
510 DISPLAY AT(5,21)SIZE(5):USING "#####":I
520 ACCEPT AT(22,28)SIZE(-1)BEEP:R$ :: IF R$="Q" T
    HEN 580
530 DISPLAY AT(7,16):I
540 ACCEPT AT(7,16)BEEP VALIDATE(NUMERIC)SIZE(-10)
    :X
550 ACCEPT AT(9,16)BEEP VALIDATE(NUMERIC)SIZE(-10)
    :Y
560 OBS(I,1)=X :: OBS(I,2)=Y
570 NEXT I
580 NOBS=I-1
590 RETURN
600 REM TAPE DATA ENTRY
610 CALL CLEAR
620 DISPLAY AT(2,2):"** Tape Data Entry"
630 OPEN #1:"CS1",SEQUENTIAL,INTERNAL,FIXED,INPUT
640 FOR I=1 TO MAXN
650 INPUT #1:X,Y
660 IF X=-1 AND Y=-1 THEN 690
670 OBS(I,1)=X :: OBS(I,2)=Y
680 NEXT I
690 NOBS=I-1
700 CLOSE #1
710 RETURN
720 REM DISK DATA ENTRY
730 DISPLAY AT(2,2)ERASE ALL:"** DISK DATA ENTRY *
    *"
740 OPEN #2:FILENAME$,INTERNAL,VARIABLE,INPUT
750 FOR I=1 TO MAXN
760 INPUT #2:X,Y
770 IF X=-1 AND Y=-1 THEN 800
780 OBS(I,1)=X :: OBS(I,2)=Y
790 NEXT I
800 NOBS=I-1
```

```
810 CLOSE #2
820 RETURN
830 REM CHANGE DATA
840 CALL CLEAR
850 DISPLAY AT(2,6):"** Change Data **"
860 DISPLAY AT(5,1):"Observation number"
870 DISPLAY AT(7,3):"Value for X"
880 DISPLAY AT(9,3):"Value for Y"
890 DISPLAY AT(17,1):"Type function code N"
900 DISPLAY AT(19,2):"D  Delete this observation"
910 DISPLAY AT(20,2):"S  See this observation"
920 DISPLAY AT(21,2):"N  See next observation"
930 DISPLAY AT(22,2):"P  See previous observation"
940 DISPLAY AT(23,2):"Q  Quit"
950 OBSNO=1
960 IF NOBS=0 THEN DISPLAY AT(13,1):MSG1$ :: CALL
    DELAY(1000):: RETURN
970 DISPLAY AT(5,20)SIZE(5):USING "#####":OBSNO
980 DISPLAY AT(7,16)SIZE(10):OBS(OBSNO,1)
990 DISPLAY AT(9,16)SIZE(10):OBS(OBSNO,2)
1000 ACCEPT AT(7,16)BEEP VALIDATE(NUMERIC)SIZE(-10
     ):OBS(OBSNO,1)
1010 ACCEPT AT(9,16)BEEP VALIDATE(NUMERIC)SIZE(-10
     ):OBS(OBSNO,2)
1020 ACCEPT AT(17,20)BEEP VALIDATE("DSNPQ")SIZE(-1
     ):R$
1030 IF R$="" THEN 1020
1040 IF R$="D" THEN GOSUB 1120
1050 IF R$="S" THEN ACCEPT AT(5,20)BEEP VALIDATE(N
     UMERIC)SIZE(5):OBSNO
1060 IF R$="N" THEN OBSNO=OBSNO+1
1070 IF R$="P" THEN OBSNO=OBSNO-1
1080 IF R$="Q" THEN RETURN
1090 IF OBSNO<1 THEN OBSNO=NOBS
1100 IF OBSNO>NOBS THEN OBSNO=1
1110 GOTO 960
1120 REM DELETE OBSERVATION
1130 FOR I=OBSNO TO NOBS
1140 OBS(I,1)=OBS(I+1,1)
1150 OBS(I,2)=OBS(I+1,2)
1160 NEXT I
1170 IF NOBS>=1 THEN OBS(NOBS,1),OBS(NOBS,2)=0 ::
     NOBS=NOBS-1
1180 RETURN
1190 REM SAVE DATA
1200 CALL CLEAR
1210 DISPLAY AT(2,6):"** Save Data **"
1220 DISPLAY AT(5,1):"Save data on"
1230 DISPLAY AT(7,3):"N  Null device"
```

```
1240 DISPLAY AT(9,3):"1  Cassette unit #1"
1250 DISPLAY AT(11,3):"2  Cassette unit #2"
1260 DISPLAY AT(13,3):"D  Disk"
1270 ACCEPT AT(5,24)BEEP SIZE(1)VALIDATE("N12D"):R
     $
1280 IF R$="" THEN 1270
1290 IF R$="N" THEN RETURN
1300 IF R$="1" OR R$="2" THEN 1380
1310 DISPLAY AT(15,3):"FILE NAME"
1320 IF FILENAME$="" THEN FILENAME$="DSK1."
1330 DISPLAY AT(17,3):FILENAME$
1340 ACCEPT AT(17,3)SIZE(-15)VALIDATE(UALPHA,".",D
     IGIT)BEEP:FILENAME$
1350 IF FILENAME$="" THEN 1340
1360 OPEN #1:FILENAME$,INTERNAL,VARIABLE,OUTPUT
1370 GOTO 1390
1380 OPEN #1:"CS"&R$,SEQUENTIAL,INTERNAL,OUTPUT,FI
     XED
1390 FOR I=1 TO NOBS
1400 PRINT #1:OBS(I,1),OBS(I,2)
1410 NEXT I
1420 PRINT #1:-1,-1
1430 CLOSE #1
1440 RETURN
1450 REM DISPLAY DATA
1460 IF NOBS=0 THEN DISPLAY AT(5,1)ERASE ALL:MSG1$
      :: CALL DELAY(500):: RETURN
1470 GOSUB 1540 !GET OPTIONS
1480 GOSUB 1900 !SCALE DATA
1490 GOSUB 2050 !DISPLAY AXIS
1500 GOSUB 2350 !DISPLAY BARS
1510 CALL KEY(0,R,S):: IF S<>1 THEN 1510
1520 R$=CHR$(R):: IF R$="P" THEN GOSUB 2800 !print
1530 RETURN
1540 REM GET DISPLAY OPTIONS
1550 CALL CLEAR
1560 DISPLAY AT(1,6):"** Display Data **"
1570 DISPLAY AT(3,1):"Enter titles for"
1580 DISPLAY AT(4,3):"Chart  ";CTITLE$
1590 DISPLAY AT(5,3):"X axis ";XTITLE$
1600 DISPLAY AT(6,3):"Y axis ";YTITLE$
1610 DISPLAY AT(8,1):"Enter type of chart ";CTYPE$
1620 DISPLAY AT(9,3):"A  average{4 SPACES}S  sum"
1630 DISPLAY AT(10,3):"F  frequency"
1640 DISPLAY AT(13,1):"X axis options"
1650 DISPLAY AT(14,3):"# of intervals (1-12)";XINT
1660 DISPLAY AT(15,3):"scaling:  C  computer  ";XS
     CLE$
1670 DISPLAY AT(16,13):"M  manual"
```

```
1680 DISPLAY AT(17,3):"interval size  "
1690 DISPLAY AT(17,18):USING "####.##":XINTSIZE
1700 DISPLAY AT(19,1):"Y axis options"
1710 DISPLAY AT(20,3):"# of intervals (1-15)";YINT
1720 DISPLAY AT(21,3):"scaling:  C  computer  ";YS
     CLE$
1730 DISPLAY AT(22,13):"M  manual"
1740 DISPLAY AT(23,3):"interval size  "
1750 DISPLAY AT(23,18):USING "####.##":YINTSIZE
1760 ACCEPT AT(4,10)BEEP SIZE(-15):CTITLE$
1770 ACCEPT AT(5,10)BEEP SIZE(-15):XTITLE$
1780 ACCEPT AT(6,10)BEEP SIZE(-15):YTITLE$
1790 ACCEPT AT(8,21)BEEP SIZE(-1)VALIDATE("AFS"):C
     TYPE$
1800 IF CTYPE$="" THEN 1790
1810 ACCEPT AT(14,25)BEEP SIZE(-2)VALIDATE(DIGIT):
     XINT :: IF XINT<1 OR XINT>XMAXINT THEN 1810
1820 ACCEPT AT(15,26)BEEP SIZE(-1)VALIDATE("CM"):X
     SCLE$
1830 IF XSCLE$="" THEN 1820
1840 IF XSCLE$="M" THEN ACCEPT AT(17,18)BEEP SIZE(
     8)VALIDATE(NUMERIC):XINTSIZE
1850 ACCEPT AT(20,25)BEEP SIZE(-2)VALIDATE(DIGIT):
     YINT :: IF YINT<1 OR YINT>YMAXINT THEN 1850
1860 ACCEPT AT(21,26)BEEP SIZE(-1)VALIDATE("CM"):Y
     SCLE$
1870 IF YSCLE$="" THEN 1860
1880 IF YSCLE$="M" THEN ACCEPT AT(23,18)BEEP SIZE(
     8)VALIDATE(NUMERIC):YINTSIZE
1890 RETURN
1900 REM SCALE DATA
1910 FOR I=1 TO XINT :: NCOUNT(I),TOTAL(I),AVERAGE
     (I)=0 :: NEXT I
1920 XMAX=-99999E-99
1930 FOR I=1 TO NOBS
1940 IF OBS(I,1)>XMAX THEN XMAX=OBS(I,1)
1950 NEXT I
1960 IF XSCLE$="C" THEN XINTSIZE=XMAX/XINT
1970 FOR I=1 TO NOBS
1980 CELL=INT(OBS(I,1)/XINTSIZE+.5)
1990 IF CELL<1 OR CELL>XINT THEN 2030
2000 NCOUNT(CELL)=NCOUNT(CELL)+1
2010 TOTAL(CELL)=TOTAL(CELL)+OBS(I,2)
2020 IF NCOUNT(CELL)>0 THEN AVERAGE(CELL)=TOTAL(CE
     LL)/NCOUNT(CELL)
2030 NEXT I
2040 RETURN
2050 REM DISPLAY AXIS
2060 CALL CHAR(128,"0404040404040407")!Y
```

```
2070 CALL CHAR(129,"0000000000001010FF")!X
2080 CALL CHAR(130,"4040404C5212120C")!10
2090 CALL CHAR(131,"4040404242020202")!11
2100 CALL CHAR(132,"40404C524204081E")!12
2110 CALL CHAR(136,"FFFFFFFFFFFFFFFF")!BAR
2120 CALL CLEAR
2130 CALL VCHAR(1,6,128,15)
2140 CALL HCHAR(15,7,129,24)
2150 J=1
2160 FOR I=YMAXINT TO 1 STEP -1
2170 DISPLAY AT(J,1)SIZE(2):USING "##":I
2180 J=J+1
2190 NEXT I
2200 J=5
2210 FOR I=1 TO 9
2220 DISPLAY AT(17,J)SIZE(2):USING "##":I
2230 J=J+2
2240 NEXT I
2250 CALL HCHAR(17,26,130,1)
2260 CALL HCHAR(17,28,131,1)
2270 CALL HCHAR(17,30,132,1)
2280 DISPLAY AT(19,(28-LEN(CTITLE$))/2):CTITLE$
2290 DISPLAY AT(20,1):"X: ";XTITLE$
2300 DISPLAY AT(21,1):"Y: ";YTITLE$;
2310 DISPLAY AT(23,1):"Press: P  to print"
2320 DISPLAY AT(24,8):"any key to continue"
2330 CALL COLOR(14,7,7)
2340 RETURN
2350 REM DISPLAY BARS
2360 YMAX=-99999E-99
2370 FOR I=1 TO XINT
2380 IF CTYPE$="F" AND NCOUNT(I)>YMAX THEN YMAX=NC
     OUNT(I)
2390 IF CTYPE$="S" AND TOTAL(I)>YMAX THEN YMAX=TOT
     AL(I)
2400 IF CTYPE$="A" AND AVERAGE(I)>YMAX THEN YMAX=A
     VERAGE(I)
2410 NEXT I
2420 IF YSCLE$="C" THEN YINTSIZE=YMAX/YINT
2430 COL=8
2440 FOR I=1 TO XINT
2450 IF CTYPE$="F" THEN NBAR=INT(NCOUNT(I)/YINTSIZ
     E+.5)
2460 IF CTYPE$="S" THEN NBAR=INT(TOTAL(I)/YINTSIZE
     +.5)
2470 IF CTYPE$="A" THEN NBAR=INT(AVERAGE(I)/YINTSI
     ZE+.5)
2480 IF NBAR>15 THEN NBAR=15
2490 IF NBAR>0 THEN CALL VCHAR(16-NBAR,COL,136,NBA
     R)
```

```
2500 COL=COL+2
2510 NEXT I
2520 DISPLAY AT(20,19):USING "* ####.##":XINTSIZE
2530 DISPLAY AT(21,19):USING "* ####.##":YINTSIZE
2540 RETURN
2550 REM INIT
2560 NOBS=0
2570 XINTSIZE,YINTSIZE=0 :: XSCLE$,YSCLE$="C" :: C
     TYPE$="S"
2580 FILENAME$="DSK1." :: PRINTFILE$="RS232.BA=120
     0"
2590 MSG1$="* NO OBSERVATIONS ENTERED *"
2600 RETURN
2610 REM MENU
2620 CALL CLEAR
2630 DISPLAY AT(2,7):"** Bar Charts **"
2640 DISPLAY AT(5,1):"Do you want to:"
2650 DISPLAY AT(7,3):"1  Enter data"
2660 DISPLAY AT(9,3):"2  Change data"
2670 DISPLAY AT(11,3):"3  Save data"
2680 DISPLAY AT(13,3):"4  Display data"
2690 DISPLAY AT(15,3):"5  End bar charts"
2700 DISPLAY AT(22,1):"Type your selection:"
2710 ACCEPT AT(22,23)BEEP SIZE(1)VALIDATE("12345")
     :R$
2720 IF R$="" THEN 2710
2730 FCODE=VAL(R$)
2740 IF FCODE=5 THEN FINI=1 :: GOTO 2760
2750 ON FCODE GOSUB 200,830,1190,1450
2760 RETURN
2770 REM END
2780 CALL CLEAR
2790 RETURN
2800 REM PRINT GRAPH
2810 DISPLAY AT(1,7)ERASE ALL:"Print Bar Charts"
2820 DISPLAY AT(6,1):"Type output device name:"
2830 DISPLAY AT(8,1):PRINTFILE$
2840 ACCEPT AT(8,1)SIZE(-28)VALIDATE(UALPHA,DIGIT,
     ".")BEEP:PRINTFILE$
2850 IF PRINTFILE$="" THEN 2840
2860 T$=SEG$(PRINTFILE$,1,2)
2870 IF T$="CS" THEN OPEN #1:PRINTFILE$,OUTPUT,DIS
     PLAY ,FIXED 128
2880 IF T$<>"CS" THEN OPEN #1:PRINTFILE$,OUTPUT,DI
     SPLAY ,VARIABLE
2890 PRINT #1: : :TAB((80-LEN(CTITLE$))/2);CTITLE$
2900 PRINT #1: : : :
2910 FOR I=YMAXINT TO 1 STEP -1
2920 PRINT #1:TAB(10);
```

```
2930 PRINT #1,USING "##{3 SPACES}":I;
2940 FOR J=1 TO XINT
2950 IF CTYPE$="F" THEN NBAR=INT(NCOUNT(J)/YINTSIZ
     E+.5)
2960 IF CTYPE$="S" THEN NBAR=INT(TOTAL(J)/YINTSIZE
     +.5)
2970 IF CTYPE$="A" THEN NBAR=INT(AVERAGE(J)/YINTSI
     ZE+.5)
2980 IF NBAR>=I THEN PRINT #1:"*** ";ELSE PRINT #1
     :"{4 SPACES}";
2990 NEXT J
3000 PRINT #1
3010 NEXT I
3020 PRINT #1:TAB(15);RPT$("-",47)
3030 PRINT #1:TAB(15);
3040 FOR I=1 TO 12 :: PRINT #1,USING "### ":I;:: N
     EXT I
3050 PRINT #1: : :TAB(11);"X-axis: ";XTITLE$;TAB(4
     9);"Scale:";
3060 PRINT #1,USING " #####.##":XINTSIZE
3070 PRINT #1: :TAB(11);"Y-axis: ";YTITLE$;TAB(49)
     ;"Scale:";
3080 PRINT #1,USING " #####.##":YINTSIZE
3090 PRINT #1:CHR$(12)!form feed
3100 CLOSE #1
3110 RETURN
3120 SUB DELAY(LENGTH)
3130 FOR I=1 TO LENGTH :: NEXT I
3140 SUBEND
```

## Program 5-2. Bar Charts 2

```
100 REM BAR CHARTS 2
110 ON WARNING NEXT :: ON BREAK NEXT
120 OPTION BASE 1
130 MAXN=100 :: XMAXINT,XINT=35 :: YMAXINT,YINT=25
140 DIM OBS(100,2),NCOUNT(35),TOTAL(35),AVERAGE(35
    )
150 IMAGE ####.##
160 GOSUB 2030 !INIT
170 GOSUB 2090 !MENU
180 IF FINI=0 THEN 170
190 GOSUB 2250 !END
200 END
210 REM ENTER DATA
220 GOSUB 330 !ENTER MENU
230 IF INPDEV$="" THEN 220
240 IF INPDEV$="K" THEN GOSUB 430
250 IF INPDEV$="T" THEN GOSUB 610
```

```
260 IF INPDEV$<>"D" THEN 320
270 DISPLAY AT(13,3):"FILE NAME "
280 DISPLAY AT(15,3):FILENAME$
290 ACCEPT AT(15,3)SIZE(-15)VALIDATE(UALPHA,".",DI
    GIT)BEEP:FILENAME$
300 IF FILENAME$="" THEN 290
310 GOSUB 730
320 RETURN
330 REM ENTER MENU
340 CALL CLEAR
350 DISPLAY AT(2,7):"** Enter Data **"
360 DISPLAY AT(5,1):"Read data from:"
370 DISPLAY AT(7,3):"K  Keyboard"
380 DISPLAY AT(9,3):"T  Tape"
390 DISPLAY AT(11,3):"D  Disk"
400 DISPLAY AT(22,1):"Type your selection"
410 ACCEPT AT(22,25)BEEP SIZE(1)VALIDATE("KTD"):IN
    PDEV$
420 RETURN
430 REM KEYBOARD ENTRY
440 CALL CLEAR
450 DISPLAY AT(2,1):"** Keyboard Data Entry **"
460 DISPLAY AT(5,1):"Observation number"
470 DISPLAY AT(7,3):"Value for X"
480 DISPLAY AT(9,3):"Value for Y"
490 DISPLAY AT(22,1):"Press 'Q' to quit;"
500 DISPLAY AT(23,7):"ENTER to continue."
510 FOR I=NOBS+1 TO MAXN
520 DISPLAY AT(5,21)SIZE(5):USING "#####":I
530 ACCEPT AT(22,28)SIZE(-1)BEEP:R$ :: IF R$="Q" T
    HEN 590
540 DISPLAY AT(7,16):I
550 ACCEPT AT(7,16)BEEP VALIDATE(NUMERIC)SIZE(-10)
    :X
560 ACCEPT AT(9,16)BEEP VALIDATE(NUMERIC)SIZE(-10)
    :Y
570 OBS(I,1)=X :: OBS(I,2)=Y
580 NEXT I
590 NOBS=I-1
600 RETURN
610 REM TAPE DATA ENTRY
620 CALL CLEAR
630 DISPLAY AT(2,2):"** Tape Data Entry"
640 OPEN #1:"CS1",SEQUENTIAL,INTERNAL,FIXED,INPUT
650 FOR I=1 TO MAXN
660 INPUT #1:X,Y
670 IF X=-1 AND Y=-1 THEN 700
680 OBS(I,1)=X :: OBS(I,2)=Y
```

```
690 NEXT I
700 NOBS=I-1
710 CLOSE #1
720 RETURN
730 REM DISK DATA ENTRY
740 DISPLAY AT(2,2)ERASE ALL:"** DISK DATA ENTRY *
    *"
750 OPEN #2:FILENAME$,INTERNAL,VARIABLE,INPUT
760 FOR I=1 TO MAXN
770 INPUT #2:X,Y
780 IF X=-1 AND Y=-1 THEN 810
790 OBS(I,1)=X :: OBS(I,2)=Y
800 NEXT I
810 NOBS=I-1
820 CLOSE #2
830 RETURN
840 REM CHANGE DATA
850 CALL CLEAR
860 DISPLAY AT(2,6):"** Change Data **"
870 DISPLAY AT(5,1):"Observation number"
880 DISPLAY AT(7,3):"Value for X"
890 DISPLAY AT(9,3):"Value for Y"
900 DISPLAY AT(17,1):"Type function code N"
910 DISPLAY AT(19,2):"D  Delete this observation"
920 DISPLAY AT(20,2):"S  See this observation"
930 DISPLAY AT(21,2):"N  See next observation"
940 DISPLAY AT(22,2):"P  See previous observation"
950 DISPLAY AT(23,2):"Q  Quit"
960 OBSNO=1
970 IF NOBS=0 THEN DISPLAY AT(13,1):MSG1$ :: CALL
    DELAY(1000):: RETURN
980 DISPLAY AT(5,20)SIZE(5):USING "#####":OBSNO
990 DISPLAY AT(7,16)SIZE(10):OBS(OBSNO,1)
1000 DISPLAY AT(9,16)SIZE(10):OBS(OBSNO,2)
1010 ACCEPT AT(7,16)BEEP VALIDATE(NUMERIC)SIZE(-10
     ):OBS(OBSNO,1)
1020 ACCEPT AT(9,16)BEEP VALIDATE(NUMERIC)SIZE(-10
     ):OBS(OBSNO,2)
1030 ACCEPT AT(17,20)BEEP VALIDATE("DSNPQ")SIZE(-1
     ):R$
1040 IF R$="" THEN 1030
1050 IF R$="D" THEN GOSUB 1130
1060 IF R$="S" THEN ACCEPT AT(5,20)BEEP VALIDATE(N
     UMERIC)SIZE(5):OBSNO
1070 IF R$="N" THEN OBSNO=OBSNO+1
1080 IF R$="P" THEN OBSNO=OBSNO-1
1090 IF R$="Q" THEN RETURN
1100 IF OBSNO<1 THEN OBSNO=NOBS
1110 IF OBSNO>NOBS THEN OBSNO=1
```

```
1120 GOTO 970
1130 REM DELETE OBSERVATION
1140 FOR I=OBSNO TO NOBS
1150 OBS(I,1)=OBS(I+1,1)
1160 OBS(I,2)=OBS(I+1,2)
1170 NEXT I
1180 IF NOBS>=1 THEN OBS(NOBS,1),OBS(NOBS,2)=0 ::
     NOBS=NOBS-1
1190 RETURN
1200 REM SAVE DATA
1210 CALL CLEAR
1220 DISPLAY AT(2,6):"** Save Data **"
1230 DISPLAY AT(5,1):"Save data on"
1240 DISPLAY AT(7,3):"N  Null device"
1250 DISPLAY AT(9,3):"1  Cassette unit #1"
1260 DISPLAY AT(11,3):"2  Cassette unit #2"
1270 DISPLAY AT(13,3):"D  Disk"
1280 ACCEPT AT(5,24)BEEP SIZE(1)VALIDATE("N12D"):R
     $
1290 IF R$="" THEN 1280
1300 IF R$="N" THEN RETURN
1310 IF R$="1" OR R$="2" THEN 1390
1320 DISPLAY AT(15,3):"FILE NAME"
1330 IF FILENAME$="" THEN FILENAME$="DSK1."
1340 DISPLAY AT(17,3):FILENAME$
1350 ACCEPT AT(17,3)SIZE(-15)VALIDATE(UALPHA,".",D
     IGIT)BEEP:FILENAME$
1360 IF FILENAME$="" THEN 1350
1370 OPEN #1:FILENAME$,INTERNAL,VARIABLE,OUTPUT
1380 GOTO 1400
1390 OPEN #1:"CS"&R$,SEQUENTIAL,INTERNAL,OUTPUT,FI
     XED
1400 FOR I=1 TO NOBS
1410 PRINT #1:OBS(I,1),OBS(I,2)
1420 NEXT I
1430 PRINT #1:-1,-1
1440 CLOSE #1
1450 RETURN
1460 REM DISPLAY DATA
1470 IF NOBS=0 THEN DISPLAY AT(5,1)ERASE ALL:MSG1$
      :: CALL DELAY(500):: RETURN
1480 GOSUB 1520 !GET OPTIONS
1490 GOSUB 1880 !SCALE DATA
1500 GOSUB 2280 !print
1510 RETURN
1520 REM GET DISPLAY OPTIONS
1530 CALL CLEAR
1540 DISPLAY AT(1,6):"** Display Data **"
1550 DISPLAY AT(3,1):"Enter titles for"
```

```
1560 DISPLAY AT(4,3):"Chart  ";CTITLE$
1570 DISPLAY AT(5,3):"X axis ";XTITLE$
1580 DISPLAY AT(6,3):"Y axis ";YTITLE$
1590 DISPLAY AT(8,1):"Enter type of chart ";CTYPE$
1600 DISPLAY AT(9,3):"A  average{4 SPACES}S  sum"
1610 DISPLAY AT(10,3):"F  frequency"
1620 DISPLAY AT(13,1):"X axis options"
1630 DISPLAY AT(14,3):"# of intervals (1-35)";XINT
1640 DISPLAY AT(15,3):"scaling:  C  computer  ";XS
     CLE$
1650 DISPLAY AT(16,13):"M  manual"
1660 DISPLAY AT(17,3):"interval size  "
1670 DISPLAY AT(17,18):USING 150:XINTSIZE
1680 DISPLAY AT(19,1):"Y axis options"
1690 DISPLAY AT(20,3):"# of intervals (1-25)";YINT
1700 DISPLAY AT(21,3):"scaling:  C  computer  ";YS
     CLE$
1710 DISPLAY AT(22,13):"M  manual"
1720 DISPLAY AT(23,3):"interval size  "
1730 DISPLAY AT(23,18):USING 150:YINTSIZE
1740 ACCEPT AT(4,10)BEEP SIZE(-15):CTITLE$
1750 ACCEPT AT(5,10)BEEP SIZE(-15):XTITLE$
1760 ACCEPT AT(6,10)BEEP SIZE(-15):YTITLE$
1770 ACCEPT AT(8,21)BEEP SIZE(-1)VALIDATE("AFS"):C
     TYPE$
1780 IF CTYPE$="" THEN 1770
1790 ACCEPT AT(14,25)BEEP SIZE(-2)VALIDATE(DIGIT):
     XINT :: IF XINT<1 OR XINT>XMAXINT THEN 1790
1800 ACCEPT AT(15,26)BEEP SIZE(-1)VALIDATE("CM"):X
     SCLE$
1810 IF XSCLE$="" THEN 1800
1820 IF XSCLE$="M" THEN ACCEPT AT(17,18)BEEP SIZE(
     8)VALIDATE(NUMERIC):XINTSIZE
1830 ACCEPT AT(20,25)BEEP SIZE(-2)VALIDATE(DIGIT):
     YINT :: IF YINT<1 OR YINT>YMAXINT THEN 1830
1840 ACCEPT AT(21,26)BEEP SIZE(-1)VALIDATE("CM"):Y
     SCLE$
1850 IF YSCLE$="" THEN 1840
1860 IF YSCLE$="M" THEN ACCEPT AT(23,18)BEEP SIZE(
     8)VALIDATE(NUMERIC):YINTSIZE
1870 RETURN
1880 REM SCALE DATA
1890 FOR I=1 TO XINT :: NCOUNT(I),TOTAL(I),AVERAGE
     (I)=0 :: NEXT I
1900 XMAX=-99999E-99
1910 FOR I=1 TO NOBS
1920 IF OBS(I,1)>XMAX THEN XMAX=OBS(I,1)
1930 NEXT I
1940 IF XSCLE$="C" THEN XINTSIZE=XMAX/XINT
```

```
1950 FOR I=1 TO NOBS
1960 CELL=INT(OBS(I,1)/XINTSIZE+.5)
1970 IF CELL<1 OR CELL>XINT THEN 2010
1980 NCOUNT(CELL)=NCOUNT(CELL)+1
1990 TOTAL(CELL)=TOTAL(CELL)+OBS(I,2)
2000 IF NCOUNT(CELL)>0 THEN AVERAGE(CELL)=TOTAL(CE
     LL)/NCOUNT(CELL)
2010 NEXT I
2020 RETURN
2030 REM INIT
2040 NOBS=0
2050 XINTSIZE,YINTSIZE=0 :: XSCLE$,YSCLE$="C" :: C
     TYPE$="S"
2060 FILENAME$="DSK1." :: PRINTFILE$="RS232.BA=120
     0"
2070 MSG1$="* NO OBSERVATIONS ENTERED *"
2080 RETURN
2090 REM MENU
2100 CALL CLEAR
2110 DISPLAY AT(2,7):"** Bar Charts **"
2120 DISPLAY AT(5,1):"Do you want to:"
2130 DISPLAY AT(7,3):"1  Enter data"
2140 DISPLAY AT(9,3):"2  Change data"
2150 DISPLAY AT(11,3):"3  Save data"
2160 DISPLAY AT(13,3):"4  Display data"
2170 DISPLAY AT(15,3):"5  End bar charts"
2180 DISPLAY AT(22,1):"Type your selection:"
2190 ACCEPT AT(22,23)BEEP SIZE(1)VALIDATE("12345")
     :R$
2200 IF R$="" THEN 2190
2210 FCODE=VAL(R$)
2220 IF FCODE=5 THEN FINI=1 :: GOTO 2240
2230 ON FCODE GOSUB 210,840,1200,1460
2240 RETURN
2250 REM END
2260 CALL CLEAR
2270 RETURN
2280 REM PRINT GRAPH
2290 DISPLAY AT(1,7)ERASE ALL:"Print Bar Charts"
2300 DISPLAY AT(6,1):"Type output device name:"
2310 DISPLAY AT(8,1):PRINTFILE$
2320 ACCEPT AT(8,1)SIZE(-28)VALIDATE(UALPHA,DIGIT,
     ".")BEEP:PRINTFILE$
2330 IF PRINTFILE$="" THEN 2320
2340 T$=SEG$(PRINTFILE$,1,2)
2350 IF T$="CS" THEN OPEN #1:PRINTFILE$,OUTPUT,DIS
     PLAY ,FIXED 128
2360 IF T$<>"CS" THEN OPEN #1:PRINTFILE$,OUTPUT,DI
     SPLAY ,VARIABLE
```

```
2370 DISPLAY AT(12,1)ERASE ALL:"Press any key to s
     top":"printing."
2380 REM PRINT GRAPH
2390 YMAX=-99999E-99 :: FOR I=1 TO XINT
2400 IF CTYPE$="F" AND NCOUNT(I)>YMAX THEN YMAX=NC
     OUNT(I)
2410 IF CTYPE$="S" AND TOTAL(I)>YMAX THEN YMAX=TOT
     AL(I)
2420 IF CTYPE$="A" AND AVERAGE(I)>YMAX THEN YMAX=A
     VERAGE(I)
2430 NEXT I
2440 IF YSCLE$="C" THEN YINTSIZE=YMAX/YINT
2450 PRINT #1: : :TAB((80-LEN(CTITLE$))/2);CTITLE$
2460 PRINT #1: : : :
2470 FOR I=YINT TO 1 STEP -1
2480 CALL KEY(0,R,S):: IF S<>0 THEN 2670
2490 PRINT #1:TAB(6);
2500 PRINT #1,USING "##  ":I;
2510 FOR J=1 TO XINT
2520 IF CTYPE$="F" THEN NBAR=INT(NCOUNT(J)/YINTSIZ
     E+.5)
2530 IF CTYPE$="S" THEN NBAR=INT(TOTAL(J)/YINTSIZE
     +.5)
2540 IF CTYPE$="A" THEN NBAR=INT(AVERAGE(J)/YINTSI
     ZE+.5)
2550 IF NBAR>=I THEN PRINT #1:"* ";ELSE PRINT #1:"
       ";
2560 NEXT J
2570 PRINT #1
2580 NEXT I
2590 PRINT #1:TAB(10);RPT$("-",70)
2600 PRINT #1:TAB(28);"1";TAB(48);"2";TAB(68);"3"
2610 PRINT #1:TAB(10);RPT$("1 2 3 4 5 6 7 8 9 0 ",
     3);
2620 PRINT #1:"1 2 3 4 5"
2630 PRINT #1: : :TAB(11);"X-axis: ";XTITLE$;TAB(4
     9);"Scale: ";
2640 PRINT #1,USING 150:XINTSIZE
2650 PRINT #1: :TAB(11);"Y-axis: ";YTITLE$;TAB(49)
     ;"Scale: ";
2660 PRINT #1,USING 150:YINTSIZE
2670 PRINT #1:CHR$(12)!form feed
2680 CLOSE #1
2690 RETURN
2700 SUB DELAY(LENGTH)
2710 FOR I=1 TO LENGTH :: NEXT I
2720 SUBEND
```

# Chapter 6

# Electronic Card File

# An Electronic Card File

How many times have you turned your house upside down looking for a piece of paper? Perhaps you needed an address for a friend you haven't seen in years. Maybe you were looking for the parts list for your five-year-old lawn mower that just made some strange noises and quit working.

Does this sound familiar? Sure, we all have a filing system. Nobody wants to lose important papers. But too often household filing systems work only one way. They are excellent for storing information. But how about retrieving things? Can you get what you need rapidly? Or, are you heard saying, "I put it in a safe place so I could find it again"?

Suppose on some rainy weekend you decided to reorganize your filing system. How might you proceed? You might start by arranging a central file. A shoe box in the bedroom closet and a shoe box in the broom closet and a shoe box everywhere just won't do. You'll need everything stored in one place so you will have a better chance of finding something again.

Next, you must decide what to keep in this new central file. Certainly health records, insurance policies, appliance warranties, receipts for computer purchases, and the like. Some items might best be kept in a safe-deposit box. Examples of these would be titles, deeds, certificates of deposit, and so forth. Even though these items may be stored in a safe-deposit box, it's a good idea to keep a record of them in a file. For example, you might record the document's number, description, and when it was put in the safe-deposit box.

Of course a household filing system will be designed for ease of retrieval. It would be embarrassing to go to all this trouble and still not be able to find anything. So the central file should be divided according to categories or *subjects*. All the warranties will be kept together, as will all the automobile

records. Thus, one starting point is setting up the list of subjects.

Let's carry this idea a bit further. Within each subject, there may be more than one document. How will a specific document be located? It's always possible to search through each document one by one. If you are unlucky enough not to know the subject, you might have to search the entire file document by document.

There is a technique that can be used to speed up document retrieval by creating a cross-reference or document index file. The index file will contain a list of all documents on file for a given subject. This is much like the author or subject index files at your public library. When searching for a document, you would look first in the index file to make sure the document is present.

For such a system to work, each document must be identified in some fashion. For many household documents, this is not a problem. Insurance policies usually have a policy number. Likewise, appliances have serial numbers. If nothing else, a sequence number will work. As a rule of thumb, each document should have its own number. This way the documents will not get confused.

Obviously, a great deal of effort is required to operate such a filing system. It takes a good bit of work to set it up to start with. As documents are added, they must be numbered and filed correctly. If a document index file is employed, it too must be kept up-to-date.

## Description

Not many households would bother with such a filing system. The cost in terms of time is just too great.

This is where your TI-99/4A comes in. For decades, computers have been advertised as labor-saving devices. In this chapter, you will see how the 99/4A will help with household filing needs.

Of course, papers are not going to be stored inside the computer. Rather, the computer will be used to record information about the documents. Things that you might keep track of are the location of the actual documents, summary information, expiration dates, and the like.

But let's think not only in terms of documents. Suppose you have an extensive music library. Organizing the selections

by composer would be handy. Or you may want to store your favorite recipes. You may even want to organize feeding and watering instructions for your household plants. The Electronic Card File satisfies these requirements beautifully.

Think of the Electronic Card File as simply an electronic version of a manual filing system. The Electronic Card File gives you a way to store and organize any type of information with the help of your TI.

### System Requirements

There are two versions of the Electronic Card File:

—Tape Card File and
—Disk Card File.

These programs differ in terms of the functions that they perform and in the hardware they require. Figure 6-1 summarizes the system requirements for these programs.

One of the strong points of the 99/4A is the way that it can gradually be expanded. The Tape Card File is designed to work with just the console. The tape version provides the simple functions of data entry and retrieval. If, on the other hand, you have both the disk system and the memory expansion, you will want to use the Disk Card File. The disk version has some additional cross-referencing features plus the capacity to accept more text for each file entry.

### Tape Card File

**Description.** The Tape Card File is an information storage and retrieval program. It works much like a 3-by-5 index card filing system. As Figure 6-2 shows, the file contains one or more index cards which are called *entries.* Each card has some identifying information written at the top. This information can be thought of as the title of the card; it will be called the card's *identifier.* The remainder of the index card contains notes.

The Tape Card File automates this concept on your TI. Figure 6-3 illustrates the Tape Card File screen format. You type the identifier name at the top of the screen. There is room for 10 lines of 25 characters each for whatever information you want. In other words, there can be up to 250 characters of text per entry. Several lines at the bottom of the screen are used by the program for operating messages and instructions.

# Electronic Card File

## Figure 6-1. Electronic Card File

System Requirements
Tape Card File
    Required:
    —TI-99/4A console
    —Extended BASIC
    —Cassette tape recorder
    Optional:
    —RS232 Interface
    —Printer
Disk Card File
    Required:
    —TI-99/4A console
    —Extended BASIC
    —32K Memory Expansion
    —Disk Drive
    —Disk Controller
    Optional:
    —RS232 Interface
    —Printer

The Tape Card File has two restrictions which should be kept in mind. First, each identifier must be given its own name. You cannot use the same identifier for more than one entry. After all, the only way that an entry can be retrieved is by its identifier. In practice, this restriction should not pose too much of a problem. More importantly, all of the entries must be in the computer's memory at the same time. This will limit you to a total of about 4000 characters of text on a 16K 99/4A. You may set up as many tape files of information as needed, however. For example, you might use one tape file to keep track of insurance documents and another to keep track of appliance warranties.

**Operation.** The functions available through the main menu screen are self-explanatory. You may:

1 load data
2 save data
3 list entries
4 retrieve an entry
5 add an entry
6 return to BASIC

**Figure 6-2. 3-by-5 Card Filing System**

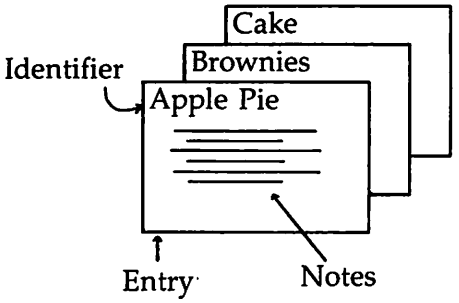
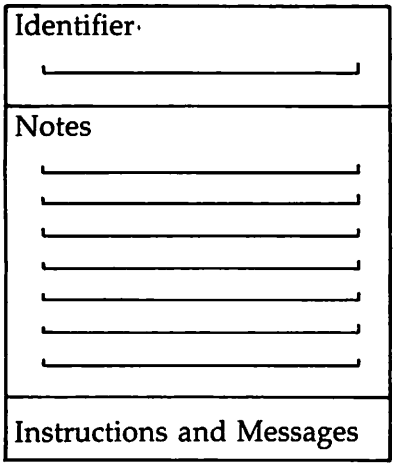



**Figure 6-3. Tape Card File Screen Format**




Selections 1 and 2 provide for the permanent storage of information. When you save data on tape, just make sure that you use a C60 cassette. TI-99/4A tape files seem to be great tape eaters. A C10 tape will handle only very small files.

Use selection 3 when you want to find out what is stored in the file. This option lists the identifier of each entry in the file.

To retrieve a specific entry, select option 4. The program will ask you to type in the identifier of the entry that you want. If the entry cannot be found, you will be asked to retype the identifier. When the entry is found, it will be displayed in the format shown in Figure 6-3.

At this point, you have three options available. First, you may simply wish to retrieve another entry. If so, just type ENTER. Perhaps you would like to revise the information shown on the screen. If you do, press R for revise. The cursor will be placed so that you can change any of the notes that you have previously typed. Notice that you cannot change the identifier. The ERASE key can be used to delete entire lines that are no longer needed. Finally, you may obtain a printout of the screen by pressing P for print. Only the text is printed—not the special graphics characters.

Selection 5 is used to add an entry to the file. You will be given a blank screen and asked to type in an identifier and any appropriate notes. When you have finished preparing an entry, you can type P, R, or ENTER. P and R work exactly as described above. ENTER, however, is used to place the information in the file. Keep in mind, that the identifier cannot be changed once the entry is in the file. Each time an entry is made, the screen will show how many characters of text have been typed. Stop when you get close to 4000 characters!

Selection 6 returns control to BASIC. Make sure that you have saved the file on tape before ending the program. To help you remember this, selection 6 will ask you if you are really sure that you want to end the program.

**Precautions.** There are several general rules to remember when using the Tape Card File. When you are writing BASIC programs, it is a good practice to save the program every now and then just in case. The same holds true for the Tape Card File. There is nothing more frustrating than losing an hour's worth of typing. Play it safe.

If you have a lot of information, watch the character count carefully. Do not go over 4000 characters on an unexpanded 99/4A. If you do, you are likely to get a memory full error. Needless to say, that is an unpleasantness of the highest degree.

As presently designed, the tape card will accept up to 30 entries. Once this limit is reached, additional entries will not be accepted. All other operations will continue to function as

described, though. On the average, you can type in about 133 characters per entry (4000 characters / 30 entries) in a 16K 99/4A.

If you do not have a printer, you may find it helpful to disable the print subroutine. This may be accomplished by placing a SUBEXIT statement immediately after the SUB PRINTCARD(ID$,BUFF$()) statement. This will prevent another unpleasantness—an I/O error if you accidentally press the P key.

If you do have a printer, make sure that the OPEN statement in the program will work with your printer. The program uses the first RS232 serial port with all the standard TI options. However, the baud rate is set to 1200. Locate the print subroutine and double-check the OPEN statement. Change it as required and make your own copy of the program.

If you follow these hints, you should find the Tape Card File easy to use and trouble-free.

## Disk Card File

**Description.** The Disk Card File program is a more advanced filing system. Figure 6-4 shows how the Disk Card File organizes your information. The file itself contains one or more subjects. Each subject, in turn, contains one or more entries. Each entry contains information relating to the particular subject. In addition, the Disk Card File program maintains a separate cross-reference or index file. The index file lets you take a quick look at all the entries belonging to a subject. As you can see, the Disk Card File closely follows the operation of the central filing system described earlier.

The Disk Card File screen format is shown in Figure 6-5. At the top of the screen there is room to type in the appropriate subject and identifier name. You may type two pages of notes per entry. Each page of notes will hold up to 10 lines of 24 characters each. Thus, there are 2 x 240 or 480 characters available per entry.

The Disk Card File program has three restrictions. First, there may be no more than 80 entries for one subject. Second, each entry must have its own identifier, and the information stored in a file must not exceed the available disk capacity. A single-sided diskette will hold about 150 entries.

As was shown in Figure 6-1, the Disk Card File program

does require the memory expansion option. The program is just too large to run in a 16K TI.

**Figure 6-4. Disk Card File Organization**

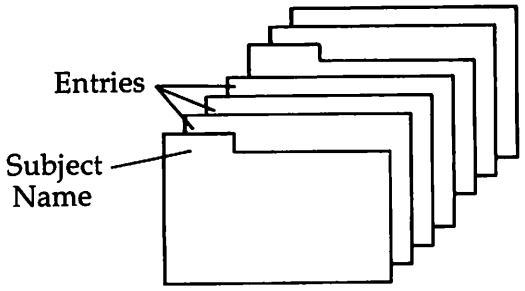
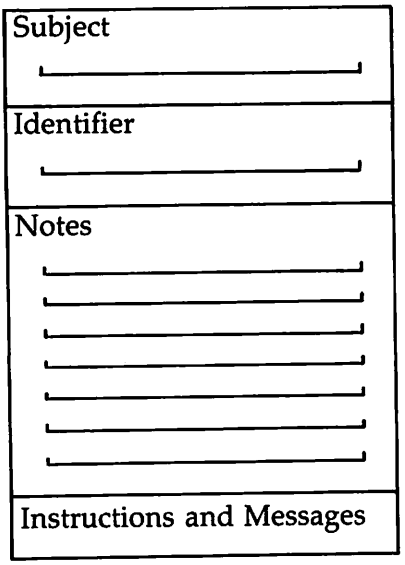



**Figure 6-5. Disk Card File Screen Format**



**Operation.** The main menu screen provides a fairly extensive set of functions:

1 list subjects
2 list entries by subject
3 browse a subject
4 browse entries
5 retrieve an entry
6 add an entry
7 return to BASIC

Notice that most of the functions have something to do with retrieving information. The Disk Card File is designed to help you locate and get what you need quickly. There are no selections for saving or loading data. This is not an omission. This Disk Card File automatically takes care of saving your information.

Let's discuss how the Disk Card File saves and loads information before discussing the other operations. When you first start the Disk Card File program, it will ask you for a filename. Type in a filename up to nine characters long. (Do not type DSK1 in front of the filename. The Disk Card File will do this for you.) Now, you will be asked whether or not this is a new file. A new file is one that you have not previously saved on the diskette. After you have entered this information, you will be given a chance to verify it. If the file is old, the subject cross-references or index file will be read from disk into the computer's memory.

At this point, the main menu screen will be displayed. If you add any new entries, they will be saved immediately on the diskette. Similarly, any changes to existing entries will also be saved immediately. While these changes and additions are being made, the cross-reference file, which is now in memory, is constantly being updated. The cross-reference or index file is written back to the diskette when you have indicated that you are ready to return to BASIC.

Selections 1 and 2 both produce lists on your display screen. Selection 1 shows you the names of all the subjects that have been typed in so far. The subjects are listed alphabetically. The second selection is similar. However, it lists each of the identifiers present within each subject. The subjects are still listed alphabetically. The identifiers, though, are listed in the order in which they were entered. You can use these two

selections to help you locate a specific entry.

Selections 3 and 4 are called *browses*. A browse displays another entry every time you type ENTER. The browse selections are used when you want to review information and possibly make a few corrections at the same time. When you browse through a subject, you will be able to review all of the details for each of the entries in that subject. Again, the entries are displayed in the order in which they were typed. Selection 4 will display the entries in alphabetical order regardless of the subject to which they belong.

Selections 3 and 4 have another very handy feature. Very seldom would you want to browse through a file from the beginning to the end. So when you use either of these browse selections, you may specify the starting point of the browse. In other words, you may specify which subject or which identifier should be the first to be displayed.

But suppose you can't remember the exact spelling of a subject or an identifier. For example, let's assume you have a file of composers. The composers' last names have been used as the subject. You want to browse through the subjects for Schumann. Alas, you can never quite remember if Schumann is spelled Schuman or Schumann. Well, pick selection 3 to browse through a subject. Then, specify that you want to begin the browse with "Schum". Notice that you must be very careful with upper- and lowercase letters. The program will look for any subject containing the letters "Schum". If Schumann is the only subject that meets this condition, you're in luck. Now, suppose that "Schu" was specified as the starting point. There is a good chance that the browse would have started with Schubert. You need to type in enough of the subject or identifier name so that you end up with the one desired subject.

Selection 5 is used to retrieve a specific entry. The entire identifier name must be typed in. The search features of the browse modes are not available in selection 5. Therefore, the program will search for an exact match with the identifier that you specify.

When an entry is displayed (through selections 3, 4, or 5), you have several options similar to those in the Tape Card File. You may press P to print the entry, R to revise information, or ENTER to continue. There are two additional options in the Disk Card File. Two pages of notes are available for

each entry. If you want to see what's on page two, press 2. Likewise, if you want to go back to page one, press 1.

Use selection 6 to add entries. Both a subject and an identifier are required. The same options—P, R, 1, 2, and ENTER—are available. Remember that once an entry is added to the file, its subject and identifier cannot be changed.

When you are finished, selection 7 will return control to BASIC. The updated cross-reference or index file will be written back to the diskette automatically. There it will be ready for the next time you want to use it. Remember, you must use selection 7, return control to BASIC, in order for the updated cross-reference or index file to be saved to disk.

**Precautions.** The same general set of precautions that were discussed for the Tape Card File also apply to the Disk Card File. The techniques may be a little different, though.

Use the Disk Manager cartridge regularly. Before you sit down to add a lot of new entries to a particular file, make sure there is plenty of room on the diskette. As a rule of thumb, each entry will gobble up two sectors. If you are planning to add 20 entries, there should be at least 40 available sectors on the diskette. Don't forget that the index files also require space. Their space requirements are very modest, but are difficult to calculate ahead of time. As an aid, the Disk Card File will display the number of entries in the file at the bottom of the main menu screen.

Be sure to make backup copies of your important files. It is inconvenient to do this with a single drive system. Weigh the inconvenience against the agony of a lost or destroyed diskette.

Suppose there is a problem of some sort while you are entering information. Let's say there was a momentary power loss. Is any of the work salvageable? Perhaps. Any new entries since the last time you exercised selection 7 will have to be reentered. Recall that the index files are kept in the computer's memory. The index files are written back to the diskette automatically when you choose selection 7. Adding entries is the only action that would affect the index files. Changing existing entries does not—they have already been given a subject and an identifier. Therefore, if a power failure did occur, the new entries would be on the diskette, but they would not be retrievable. There was no chance for the program to write the revised index files to the diskette. So, the new entries would

have to be typed in again in order to get things straightened out. This may sound a bit complicated, but it's handy to know if you ever run into this type of situation.

As was discussed earlier, you may want to disable the print subroutine if you do not have a printer. This is a very prudent step that could prevent an accidental program crash. Follow the instructions given for the Tape Card File. If you do have a printer, make sure that the OPEN statement is correct for your particular printer.

### Just for Practice

Figure 6-6 lists some sample information that you can use with the Tape Card File or Disk Card File programs. The application is illustrative of how you might make a household inventory. In the example there are four subjects or categories—appliances, computers, stereos, and televisions. Each subject has one or more entries.

Try typing this information. If you are using the Tape Card File, do not type the subjects; just type the identifier and accompanying notes. Notice that there are two entries for televisions. The entries are called TV 1 and TV 2. Remember that you cannot use the same identifier name more than once.

After you have entered the information, try the different retrieval options available for the program that you are using. The Tape Card File should be able to list the names of the entries as well as display any entry. There are several retrieval options in the Disk Card File. Try each of them.

Next, you might try changing an entry. Retrieve the specified entry in which you're interested. When the notes are displayed, press R to revise them. The cursor should be positioned to the beginning of the notes. Make any changes that you wish. After making the changes, retrieve this same entry again just to make sure that the changes were actually applied and that everything is working.

If you have a printer, try printing an entry or two. If you experience any difficulties here, check the OPEN statement in the program. The OPEN statement must match the communications specifications of your printer.

Finally, attempt to save the information and load it back again. The save and load process should not change the information in any way. The subject, if applicable, identifier, and notes should say exactly the same things after the

information has been reloaded from tape or diskette.

This example shows how you might use the Electronic Card File in your home. The example also provides you with a chance to enter and manipulate information. This is the best way to learn how to use the Tape Card File and Disk Card File.

**Figure 6-6. Sample Information—Electronic Card File**

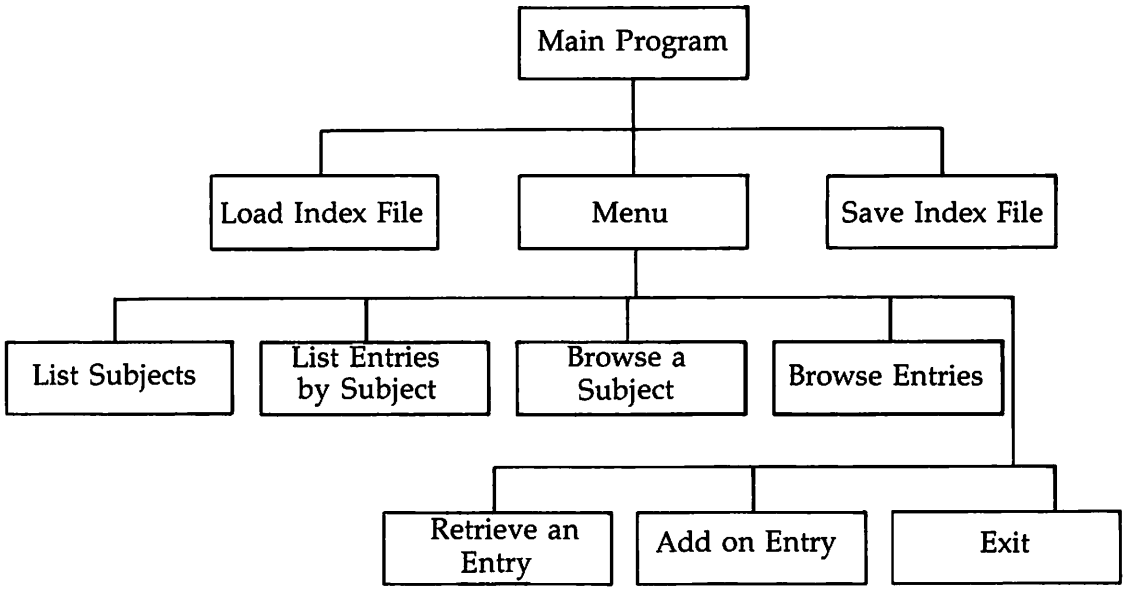
| | |
|---|---|
| SUBJECT: | Appliances |
| IDENTIFIER: | Toaster |
| NOTES: | Purchased 1/2/83; # 187428 |
| SUBJECT: | Appliances |
| IDENTIFIER: | Microwave |
| NOTES: | Purchased 5/18/83; # 98456 |
| SUBJECT: | Computers |
| IDENTIFIER: | TI-99/4A |
| NOTES: | Purchased 2/6/82; # 100899 |
| SUBJECT: | Stereo |
| IDENTIFIER: | Speakers |
| NOTES: | Purchased 3/1/75; # 00045 |
| SUBJECT: | Televisions |
| IDENTIFIER: | TV 1 |
| NOTES: | Purchased 9/4/80; # 998001 |
| SUBJECT: | Televisions |
| IDENTIFIER: | TV 2 |
| NOTES: | Purchased 6/9/83; # 472819 |

## How It Works

The Disk Card File program is perhaps the most complex program in this book. It uses a combination of RELATIVE and SEQUENTIAL files to form an indexed file structure. This is similar to the technique discussed in Chapter 3. However, the Disk Card File program carries the concept a step further. The Disk Card File allows multiple entries per subject. Thus, a subject by itself will not uniquely identify an entry.

How does the program keep track of the entries that belong to each subject? Disk Card File keeps a list of record numbers associated with each subject. When a record is added, the subject is located. Then the record number is added

**Figure 6-7. Disk Card File Program Structure**



to the list. The process works for retrieval as well. If you have a subject, you can search the subject list for a match. When a match is found, the proper record numbers will be made available. Then you can display each of the entries for the subject.

The Tape Card File program is a simplified version of Disk Card File. Tape Card File keeps all of the entries in memory rather than on disk. Furthermore, the subject cross-referencing capabilities are not present in Tape Card File.

Figure 6-7 shows the program structure of Disk Card File. The main program dimensions the arrays and initializes a few variables. If you are working with an existing data file, the index files are loaded. The main menu shows the different selections that are available. One of several subroutines will be invoked, depending on the function selected. When you are finished, control returns to the main program. The index files are automatically written back to disk.

Disk Card File's major variables are listed in Figure 6-8. Figure 6-9 describes the many subprograms used by Disk Card File.

The Tape Card File program is a bit different. However, the information presented in Figures 6-7 through 6-9 should give you enough information to follow the Tape Card File program.

### Figure 6-8. Disk Card File Variables

| | |
|---|---|
| SUBJECT$(*n*,*n*)— | This two-dimensional string array contains the list of subjects and associated record numbers. The subject names are stored in SUBJECT$(X,1). Record numbers are stored in SUBJECT$(X,2). Record numbers are stored as a list of three-digit numbers with leading zeros. For example, record numbers 2 and 8 would be stored as "002008". This is the way multiple entries are associated with a particular subject. |
| IDLIST$(*n*)— | This string array contains an identifier and its record number. The record number is in characters 1 through 3. The identifier occupies characters 4 and onward. Suppose that the recipe for apple |

| | |
|---|---|
| | pie is on record 26. The IDLIST$ entry would be "026APPLE PIE". |
| NMAX— | The maximum number of entries allowed. The dimensions of IDLIST$ and SUBJECT$ should match NMAX. Two sectors of disk space are consumed by each entry. If NMAX is 150, 300 sectors will be used. |
| BUFFER$(*n*)— | This string array is a hold area for two pages of notes. |

**Figure 6-9. Disk Card File Subprograms**

| | |
|---|---|
| DELAY— | A simple delay loop. |
| GETREC— | Reads an entry from the disk file. An entry consists of 20 lines of 24 characters. Two disk records are needed for one entry. |
| PUTREC— | Writes an entry to the disk file. Two disk records are written for each entry. |
| GETNAME— | Asks the user for a nine-character filename. Appends an " X" to the filename, thereby obtaining the index filename. |
| FINDSUBJ— | Finds the subject name associated with a given record number. Searches SUBJECT$(X,2) for the record number. |
| SEARCHID— | Finds the record number belonging to a given identifier. Searches IDLIST$(X) for an identifier match. |
| CHANGECARD— | Changes the information for an entry. Uses several other subprograms. |
| LISTENTRY— | Lists the identifier names of the entries belonging to a subject. |
| SEARCHSUBJ— | Searches the subject list SUBJECT$(X,1) for a match on a user-supplied subject name. |
| MATCHID— | Searches the identifier list IDLIST$(X) for a match with a given identifier name. |
| ADDID— | Adds a new identifier name to the identifier list. Identifier names are kept in alphabetical order. Makes sure that no duplicates are added. |

| | |
|---|---|
| ADDSUBJ— | Adds a new subject and record number to the subject list. If the subject is already in the subject list, the record number of the new record is added to the record number list. |
| LISTSUBJ— | Lists all the subjects in the subject list. |
| BLANKLINES— | Clears the screen from a beginning row to an ending row. |
| MAINMENU— | Displays the main menu and obtains the selection. |
| PRINTCARD— | Prints the information for an entry. Note: The OPEN statement must be set up to work with your printer. |
| READCARD— | Reads the screen. Obtains the subject, identifier, and notes for an entry. Handles paging of notes. |
| CLEANSTR— | Removes special graphics characters from the end of a string read from the screen. |
| WRITECARD— | Displays the information for an entry on the screen. Handles paging of notes. |
| GETRESPONSE— | Gets response key from the keyboard. Accepts specified keys. |
| GETKEY— | Displays a message line. Waits for a key to be pressed. |
| GETYN— | Accepts a Y or N response. |
| DRAWCARD— | Displays a blank entry ready for subsequent input. |
| DRAW BORDER— | Draws the border around the entry, using graphics characters. |
| TITLESCREEN— | DISPLAYS the title screen, using graphics characters. |

## Program 6-1. Tape Card File

```
100 REM TAPE CARD FILE
110 ON BREAK NEXT :: ON WARNING NEXT :: OPTION BAS
    E 1
120 NEXTREC=1 :: NMAX=30
130 DIM IDL$(30),NT$(30,10),BUFF$(10)
140 REM MAIN ROUTINE
150 CALL MAINMENU(FCODE):: IF FCODE=6 THEN 170
160 ON FCODE GOSUB 210,360,530,750,560 :: GOTO 150
170 DISPLAY AT(3,1)ERASE ALL:"Are you sure (Y or N
    )?"
180 CALL GETKEY(R,""):: IF R<>89 THEN 150
```

```
190 CALL CLEAR :: PRINT "Thank You": :"Program Com
    pleted"
200 END
210 REM LOAD DATA
220 CALL CLEAR :: FOR I=1 TO NMAX :: IDL$(I)="" ::
     FOR J=1 TO 10 :: NT$(I,J)="" :: NEXT J :: NEX
    T I
230 OPEN #2:"CS1",INTERNAL,INPUT ,FIXED 192
240 INPUT #2:LIM :: NEXTREC=LIM+1
250 FOR I=1 TO LIM STEP 5
260 INPUT #2:IDL$(I),IDL$(I+1),IDL$(I+2),IDL$(I+3)
    ,IDL$(I+4)
270 NEXT I
280 FOR I=1 TO LIM
290 INPUT #2:RTYPE$,:: IF RTYPE$="2" THEN 330
300 INPUT #2:NT$(I,1),NT$(I,2),NT$(I,3),NT$(I,4),N
    T$(I,5),
310 INPUT #2:NT$(I,6),NT$(I,7),NT$(I,8),NT$(I,9),N
    T$(I,10)
320 GOTO 340
330 FOR J=1 TO 10 STEP 5 :: INPUT #2:NT$(I,J),NT$(
    I,J+1),NT$(I,J+2),NT$(I,J+3),NT$(I,J+4):: NEXT
     J
340 NEXT I :: CLOSE #2
350 RETURN
360 REM SAVE DATA
370 CALL CLEAR :: LIM=NEXTREC-1
380 OPEN #2:"CS1",INTERNAL,OUTPUT,FIXED 192
390 PRINT #2:LIM
400 FOR I=1 TO LIM STEP 5
410 PRINT #2:IDL$(I);IDL$(I+1);IDL$(I+2);IDL$(I+3)
    ;IDL$(I+4)
420 NEXT I
430 FOR I=1 TO LIM :: T=0 :: FOR J=1 TO 10 :: T=T+
    LEN(NT$(I,J)):: NEXT J
440 IF T>180 THEN 480
450 PRINT #2:"1";NT$(I,1);NT$(I,2);NT$(I,3);NT$(I,
    4);NT$(I,5);
460 PRINT #2:NT$(I,6);NT$(I,7);NT$(I,8);NT$(I,9);N
    T$(I,10)
470 GOTO 510
480 PRINT #2:"2";:: FOR J=1 TO 10 STEP 5
490 PRINT #2:NT$(I,J);NT$(I,J+1);NT$(I,J+2);NT$(I,
    J+3);NT$(I,J+4)
500 NEXT J
510 NEXT I :: CLOSE #2
520 RETURN
530 REM LIST ENTRIES
540 CALL LISTENTRY(IDL$(),NMAX,RCODE)
```

```
550 RETURN
560 REM ADD AN ENTRY
570 CALL DRAWCARD :: ID$="" :: FOR I=1 TO 10 :: BU
    FF$(I)="" :: NEXT I
580 CALL WRITECARD(ID$,BUFF$())
590 CALL READCARD(ID$,BUFF$(),"A")
600 CALL GETRESPONSE(RESP)
610 IF RESP=1 THEN 590
620 IF RESP=5 THEN CALL PRINTCARD(ID$,BUFF$()):: G
    OTO 600
630 DISPLAY AT(20,1):"* Working.  Just a moment. *
    "
640 CALL ADDID(ID$,IDL$(),NMAX,NEXTREC,RCODE)
650 IF RCODE>0 THEN DISPLAY AT(20,1):"*Identifier
    already present*" :: PAGE=1 :: GOTO 590
660 IF RCODE<0 THEN DISPLAY AT(20,1):"*Sorry, memo
    ry is full. *" :: CALL DELAY(2000):: GOTO 740
670 FOR I=1 TO 10 :: NT$(NEXTREC,I)=BUFF$(I):: NEX
    T I
680 T=0 :: FOR I=1 TO NEXTREC :: T=T+LEN(IDL$(I)):
    : FOR J=1 TO 10 :: T=T+LEN(NT$(I,J)):: NEXT J
    :: NEXT I
690 DISPLAY AT(24,1)ERASE ALL:"Characters used ";T
700 NEXTREC=NEXTREC+1
710 DISPLAY AT(12,1):"Do you want to add another
    entry (Y or N)?"
720 CALL GETKEY(RESP,"")
730 IF RESP=89 THEN 560 ELSE IF RESP<>78 THEN 710
740 RETURN
750 REM RETRIEVE AN ENTRY
760 DISPLAY AT(1,6)ERASE ALL:"Retrieve an Entry"
770 DISPLAY AT(3,1):"Type the entry identifier." :
    : DISPLAY AT(20,1):"Hit ENTER when done."
780 ACCEPT AT(5,1)SIZE(-25)BEEP:ID$ :: IF ID$="" T
    HEN 860
790 CALL MATCHID(ID$,IDL$(),NMAX,RCODE)
800 IF RCODE=0 THEN DISPLAY AT(8,1):"Identifier no
    t found." :: GOTO 780
810 IDN$=SEG$(IDL$(RCODE),1,3):: P=VAL(IDN$)
820 FOR I=1 TO 10 :: BUFF$(I)=NT$(P,I):: NEXT I
830 CALL CHANGECARD(ID$,BUFF$(),0)
840 FOR I=1 TO 10 :: NT$(P,I)=BUFF$(I):: NEXT I
850 GOTO 760
860 RETURN
870 SUB DELAY(D)
880 FOR I=1 TO D :: NEXT I
890 SUBEND
900 SUB CHANGECARD(ID$,BUFF$(),FSW)
910 IF FSW=0 THEN CALL DRAWCARD
```

```
920 CALL WRITECARD(ID$,BUFF$()):: CALL GETRESPONSE
    (RESP)
930 IF RESP=4 THEN 970
940 IF RESP=1 THEN CALL READCARD(ID$,BUFF$(),"C")
950 IF RESP=5 THEN CALL PRINTCARD(ID$,BUFF$())
960 GOTO 920
970 SUBEND
980 SUB LISTENTRY(IDL$(),NMAX,RCODE)
990 P2=1 :: RCODE=0 :: DISPLAY AT(1,8)ERASE ALL:"L
    ist of Entries"
1000 DISPLAY AT(22,2):"SPACE  to continue"
1010 DISPLAY AT(23,2):"ENTER  to end list"
1020 FOR ROW=3 TO 18
1030 IF IDL$(P2)="" OR P2>NMAX THEN CALL BLANKLINE
     S(ROW,18):: P2=1 :: GOTO 1060
1040 DISPLAY AT(ROW,2):SEG$(IDL$(P2),4,25):: P2=P2
     +1
1050 NEXT ROW
1060 CALL GETKEY(R,"PRESS"):: IF R=32 THEN 1020 EL
     SE IF R<>13 THEN 1060
1070 SUBEND
1080 SUB MATCHID(ID$,IDL$(),NMAX,RCODE)
1090 RCODE=0 :: FOR I=1 TO NMAX
1100 IF ID$=SEG$(IDL$(I),4,255)THEN RCODE=I :: GOT
     O 1120
1110 NEXT I
1120 SUBEND
1130 SUB ADDID(ID$,IDL$(),NMAX,NEXTREC,RCODE)
1140 RCODE=0 :: FOR I=1 TO NMAX :: T$=SEG$(IDL$(I)
     ,4,25)
1150 IF ID$=T$ THEN RCODE=I :: GOTO 1240
1160 IF IDL$(I)="" OR ID$<T$ THEN P1=I :: GOTO 118
     0
1170 NEXT I :: RCODE=-1 :: GOTO 1240
1180 FOR I=P1 TO NMAX
1190 IF IDL$(I)="" THEN P2=I :: GOTO 1210
1200 NEXT I :: RCODE=-1 :: GOTO 1240
1210 FOR I=P2 TO P1+1 STEP -1 :: IDL$(I)=IDL$(I-1)
     :: NEXT I
1220 T$=STR$(NEXTREC):: T$=RPT$("0",3-LEN(T$))&T$
1230 IDL$(P1)=T$&ID$ :: IF P2<NMAX THEN IDL$(P2+1)
     =""
1240 SUBEND
1250 SUB BLANKLINES(BEGINL,ENDL)
1260 FOR I=BEGINL TO ENDL :: DISPLAY AT(I,1):" " :
     : NEXT I
1270 SUBEND
1280 SUB MAINMENU(FCODE)
1290 CALL SCREEN(12):: CALL DRAWBORDER
```

```
1300 DISPLAY AT(3,5):"Electronic Card File"
1310 DISPLAY AT(6,1):"Do you want to:"
1320 DISPLAY AT(8,2):"1  load data"
1330 DISPLAY AT(9,2):"2  save data"
1340 DISPLAY AT(10,2):"3  list entries"
1350 DISPLAY AT(11,2):"4  retrieve an entry"
1360 DISPLAY AT(13,2):"5  add an entry"
1370 DISPLAY AT(15,2):"6  return to BASIC"
1380 CALL GETKEY(R,"Enter your selection.");: IF R
     <49 OR R>55 THEN 1380 ELSE FCODE=R-48
1390 SUBEND
1400 SUB PRINTCARD(ID$,BUFF$())
1410 IF F=0 THEN OPEN #1:"RS232.BA=1200" :: F=1
1420 PRINT #1:TAB(10);"IDENTIFIER:";TAB(25);ID$:TA
     B(10);"NOTES:": :
1430 FOR I=1 TO 10
1440 IF BUFF$(I)<>"" THEN PRINT #1:TAB(15);BUFF$(I
     )
1450 NEXT I
1460 FOR I=1 TO 5 :: PRINT #1 :: NEXT I
1470 SUBEND
1480 SUB READCARD(ID$,BUFF$(),TRANS$)
1490 IF TRANS$<>"A" THEN 1530
1500 ACCEPT AT(4,2)SIZE(-25)BEEP:ID$ :: CALL CLEAN
     STR(ID$)
1510 IF ID$="" THEN DISPLAY AT(20,2)BEEP:"**Please
      enter identifier**" :: GOTO 1500
1520 DISPLAY AT(20,1):" "
1530 FOR I=1 TO 10
1540 ACCEPT AT(8+I,2)SIZE(-25)BEEP:T$ :: CALL CLEA
     NSTR(T$):: IF T$="" AND BUFF$(I)="" THEN 1560
      ELSE BUFF$(I)=T$
1550 NEXT I
1560 SUBEND
1570 SUB CLEANSTR(T$)
1580 T=POS(T$,CHR$(136),1):: IF T THEN T$=SEG$(T$,
     1,T-1)
1590 SUBEND
1600 SUB WRITECARD(ID$,BUFF$())
1610 IF FL$="" THEN FL$=RPT$(CHR$(136),25)
1620 DISPLAY AT(4,2)SIZE(25):ID$&FL$
1630 FOR I=1 TO 10 :: DISPLAY AT(8+I,2)SIZE(25):BU
     FF$(I)&FL$ :: NEXT I
1640 SUBEND
1650 SUB GETRESPONSE(RESPONSE)
1660 RESPONSE=0 :: CALL GETKEY(R,"PRESS ONE OF THE
      FOLLOWING")
1670 IF R=13 THEN RESPONSE=4 !ENTER
1680 IF R=82 THEN RESPONSE=1 !R
```

```
1690 IF R=80 THEN RESPONSE=5 !P
1700 IF RESPONSE=0 THEN 1660
1710 SUBEND
1720 SUB GETKEY(R,MSG$)
1730 DISPLAY AT(20,1):MSG$
1740 CALL SOUND(200,262,0):: K=0
1750 CALL KEY(3,R,S):: IF S<>1 THEN K=K+1 :: IF K<
     250 THEN 1750 ELSE 1740
1760 CALL KEY(5,R1,S):: DISPLAY AT(20,1):" "
1770 SUBEND
1780 SUB DRAWCARD
1790 CALL SCREEN(16):: IF F=0 THEN CALL CHAR(132,"
     000000FF"):: CALL CHAR(136,"0000000000427E"):
     : CALL COLOR(14,13,1):: F=1
1800 CALL DRAWBORDER
1810 CALL HCHAR(6,2,132,30):: CALL HCHAR(19,2,132,
     30):: CALL HCHAR(21,2,132,30)
1820 DISPLAY AT(2,1):"Identifier:"
1830 DISPLAY AT(7,1):"Notes:"
1840 DISPLAY AT(22,1):"R to revise.  P to print"
1850 DISPLAY AT(23,1):"ENTER when done."
1860 SUBEND
1870 SUB DRAWBORDER
1880 CALL CLEAR
1890 IF P1$="" THEN P1$="FF"&RPT$("00",7)&RPT$("01
     ",8)&RPT$("00",7)&"FF"&RPT$("80",8):: CALL CH
     AR(128,P1$):: CALL COLOR(13,5,1)
1900 CALL HCHAR(1,2,128,30):: CALL VCHAR(1,1,129,2
     4):: CALL HCHAR(24,2,130,30):: CALL VCHAR(1,3
     2,131,24)
1910 SUBEND
```

## Program 6-2. Disk Card File

```
100 REM DISK CARD FILE
110 ON BREAK NEXT :: ON WARNING NEXT :: OPTION BAS
    E 1
120 NEXTREC=0 :: NMAX=150
130 DIM SUBJECT$(150,2),IDLIST$(150),BUFFER$(20)
140 REM MAIN ROUTINE
150 CALL TITLESCREEN
160 CALL GETNAME(INDEXNM$,DATANM$,NEWFILE$)
170 IF NEWFILE$="N" THEN GOSUB 280
180 OPEN #3:"DSK1."&DATANM$,UPDATE,INTERNAL,RELATI
    VE,FIXED 254
190 CALL MAINMENU(FCODE,NEXTREC)
200 IF FCODE=7 THEN 230
210 ON FCODE GOSUB 440,470,500,640,970,770
220 GOTO 190
```

```
230 DISPLAY AT(3,1)ERASE ALL:"Are you sure (Y or N
    )?"
240 CALL GETYN(R$):: IF R$="N" THEN 190 ELSE GOSUB
     360
250 CALL CLEAR :: PRINT "Thank You": :"Program Com
    pleted"
260 CLOSE #3
270 END
280 REM LOAD INDEX FILE
290 DISPLAY AT(1,7)ERASE ALL:"Load Index File" ::
    DISPLAY AT(5,1):"Just a moment."
300 OPEN #2:"DSK1."&INDEXNM$,INPUT ,INTERNAL,VARIA
    BLE 254
310 INPUT #2:NEXTREC :: LIM=NEXTREC+1
320 FOR I=1 TO LIM :: INPUT #2:SUBJECT$(I,1):: INP
    UT #2:SUBJECT$(I,2):: NEXT I
330 FOR I=1 TO LIM :: INPUT #2:IDLIST$(I):: NEXT I
340 CLOSE #2
350 RETURN
360 REM SAVE INDEX FILE
370 DISPLAY AT(1,7)ERASE ALL:"Save Index File" ::
    DISPLAY AT(5,1):"Just a moment."
380 OPEN #2:"DSK1."&INDEXNM$,OUTPUT,INTERNAL,VARIA
    BLE 254
390 PRINT #2:NEXTREC :: LIM=NEXTREC+1
400 FOR I=1 TO LIM :: PRINT #2:SUBJECT$(I,1):: PRI
    NT #2:SUBJECT$(I,2):: NEXT I
410 FOR I=1 TO LIM :: PRINT #2:IDLIST$(I):: NEXT I
420 CLOSE #2
430 RETURN
440 REM LIST SUBJECTS
450 CALL LISTSUBJ(SUBJECT$(,),NMAX)
460 RETURN
470 REM LIST ENTRIES BY SUBJECT
480 CALL LISTENTRY(IDLIST$(),SUBJECT$(,),NMAX,RCOD
    E)
490 RETURN
500 REM BROWSE A SUBJECT
510 FSW=0 :: DISPLAY AT(1,7)ERASE ALL:"Browse a Su
    bject"
520 CALL SEARCHSUBJ(SUBJECT$(,),NMAX,P,RCODE)
530 LIM=LEN(SUBJECT$(P,2)):: P2=1 :: SUBJ$=SUBJECT
    $(P,1)
540 IDN$=SEG$(SUBJECT$(P,2),P2,3)
550 FOR I=1 TO NMAX
560 IF IDN$=SEG$(IDLIST$(I),1,3)THEN ID$=SEG$(IDLI
    ST$(I),4,25):: GOTO 580
570 NEXT I :: ID$="**Missing**"
```

```
580 R=VAL(IDN$)
590 CALL GETREC(R,BUFFER$())
600 CALL CHANGECARD(SUBJ$,ID$,BUFFER$(),USW,FSW)::
    FSW=FSW+1
610 IF USW THEN CALL PUTREC(R,BUFFER$())
620 P2=P2+3 :: IF P2<=LIM THEN 540
630 RETURN
640 REM BROWSE ENTRIES
650 FSW=0 :: DISPLAY AT(1,8)ERASE ALL:"Browse Entr
    ies"
660 CALL SEARCHID(IDLIST$(),NMAX,P,RCODE)
670 FOR I=P TO NMAX
680 IDN$=SEG$(IDLIST$(I),1,3):: ID$=SEG$(IDLIST$(I
    ),4,25)
690 IF IDN$="" THEN 760
700 CALL FINDSUBJ(IDN$,SUBJ$,SUBJECT$(,),NMAX)
710 R=VAL(IDN$)
720 CALL GETREC(R,BUFFER$())
730 CALL CHANGECARD(SUBJ$,ID$,BUFFER$(),USW,FSW)::
    FSW=FSW+1
740 IF USW THEN CALL PUTREC(R,BUFFER$())
750 NEXT I
760 RETURN
770 REM ADD AN ENTRY
780 CALL DRAWCARD
790 PAGE=1 :: SUBJ$,ID$="" :: FOR I=1 TO 20 :: BUF
    FER$(I)="" :: NEXT I
800 CALL WRITECARD(SUBJ$,ID$,BUFFER$(),PAGE)
810 CALL READCARD(SUBJ$,ID$,BUFFER$(),PAGE,"A")
820 CALL GETRESPONSE(RESP)
830 IF RESP=1 THEN 810
840 IF RESP=2 THEN PAGE=1 :: GOTO 800
850 IF RESP=3 THEN PAGE=2 :: GOTO 800
860 IF RESP=5 THEN CALL PRINTCARD(SUBJ$,ID$,BUFFER
    $()):: GOTO 820
870 DISPLAY AT(20,1):"* Working.  Just a moment. *
    "
880 CALL ADDID(ID$,IDLIST$(),NMAX,NEXTREC,RCODE)
890 IF RCODE>0 THEN DISPLAY AT(20,1):"*Identifier
    already present*" :: PAGE=1 :: GOTO 810
900 IF RCODE<0 THEN DISPLAY AT(20,1):"*Sorry, memo
    ry is full.*" :: CALL DELAY(2000):: GOTO 960
910 CALL ADDSUBJ(SUBJ$,SUBJECT$(,),NMAX,NEXTREC,RC
    ODE)
920 CALL PUTREC(NEXTREC,BUFFER$())
930 NEXTREC=NEXTREC+1
940 DISPLAY AT(12,1)ERASE ALL:"Do you want to add
    another  entry (Y or N)?"
950 CALL GETYN(R$):: IF R$="Y" THEN 770
```

```
960 RETURN
970 REM RETRIEVE AN ENTRY
980 ID$=""
990 DISPLAY AT(1,6)ERASE ALL:"Retrieve an Entry"
1000 DISPLAY AT(3,1):"Type the entry identifier."
     :: DISPLAY AT(20,1):"Hit ENTER when done."
1010 ACCEPT AT(5,1)SIZE(-25)BEEP:ID$ :: IF ID$=""
     THEN 1110
1020 CALL MATCHID(ID$,IDLIST$(),NMAX,RCODE)
1030 IF RCODE THEN 1050
1040 DISPLAY AT(8,1):"Identifier not found." :: GO
     TO 1010
1050 IDN$=SEG$(IDLIST$(RCODE),1,3):: P=VAL(IDN$)
1060 CALL FINDSUBJ(IDN$,SUBJ$,SUBJECT$(,),NMAX)
1070 CALL GETREC(P,BUFFER$())
1080 CALL CHANGECARD(SUBJ$,ID$,BUFFER$(),USW,0)
1090 IF USW THEN CALL PUTREC(P,BUFFER$())
1100 GOTO 990
1110 RETURN
1120 SUB DELAY(D)
1130 FOR I=1 TO D :: NEXT I
1140 SUBEND
1150 SUB GETREC(RECNO,T$())
1160 INPUT #3,REC 2*RECNO:R$,T$(1),T$(2),T$(3),T$(
     4),T$(5),T$(6),T$(7),T$(8),T$(9),T$(10)
1170 INPUT #3,REC 2*RECNO+1:R$,T$(11),T$(12),T$(13
     ),T$(14),T$(15),T$(16),T$(17),T$(18),T$(19),T
     $(20)
1180 SUBEND
1190 SUB PUTREC(RECNO,T$())
1200 R=RECNO*2 :: R$=STR$(R):: R$=RPT$("0",3-LEN(R
     $))&R$
1210 PRINT #3,REC R:R$,T$(1),T$(2),T$(3),T$(4),T$(
     5),T$(6),T$(7),T$(8),T$(9),T$(10)
1220 R=R+1 :: R$=STR$(R):: R$=RPT$("0",3-LEN(R$))&
     R$
1230 PRINT #3,REC R:R$,T$(11),T$(12),T$(13),T$(14)
     ,T$(15),T$(16),T$(17),T$(18),T$(19),T$(20)
1240 SUBEND
1250 SUB GETNAME(INDEXNM$,DATANM$,NEWFILE$)
1260 DISPLAY AT(1,5)ERASE ALL:"Electronic Card Fil
     e"
1270 DISPLAY AT(3,1):"What is the file name?"
1280 DISPLAY AT(4,1):"(May be up to 9 characters.)
     "
1290 ACCEPT AT(6,1)SIZE(-9)VALIDATE(UALPHA,DIGIT)B
     EEP:FNAME$
1300 INDEXNM$=FNAME$&"X" :: DATANM$=FNAME$&"D"
1310 DISPLAY AT(8,1):"Is it a new file? (Y or N)"
```

```
1320 CALL GETYN(NEWFILE$)
1330 DISPLAY AT(12,1):"The file name is ";FNAME$
1340 IF NEWFILE$="Y" THEN T$="New" ELSE T$="Old"
1350 DISPLAY AT(13,1):"The file is ";T$
1360 DISPLAY AT(15,1):"O.K. (Y or N)?"
1370 CALL GETYN(R$):: IF R$="N" THEN 1290
1380 SUBEND
1390 SUB FINDSUBJ(IDN$,SUBJ$,SUBJECT$(,),NMAX)
1400 FOR J=1 TO NMAX :: T$=SUBJECT$(J,2)
1410 FOR K=1 TO LEN(T$)STEP 3
1420 IF IDN$=SEG$(T$,K,3)THEN SUBJ$=SUBJECT$(J,1):
     : GOTO 1440
1430 NEXT K :: NEXT J :: SUBJ$="**Missing**"
1440 SUBEND
1450 SUB SEARCHID(IDLIST$(),NMAX,P,RCODE)
1460 RCODE=0 :: DISPLAY AT(3,1):"With which identi
     fier do youwish to begin the list?"
1470 ACCEPT AT(6,2)BEEP:ID$
1480 IF ID$="" THEN P=1 :: GOTO 1530
1490 DISPLAY AT(20,1):"Searching ..."
1500 FOR I=1 TO NMAX
1510 IF POS(SEG$(IDLIST$(I),4,25),ID$,1)<>0 THEN P
     =I :: GOTO 1530
1520 NEXT I :: P=1
1530 DISPLAY AT(20,1):" "
1540 SUBEND
1550 SUB CHANGECARD(SUBJ$,ID$,BUFFER$(),USW,FSW)
1560 USW=0 :: PAGE=1 :: IF FSW=0 THEN CALL DRAWCAR
     D
1570 CALL WRITECARD(SUBJ$,ID$,BUFFER$(),PAGE)
1580 CALL GETRESPONSE(RESP)
1590 IF RESP=4 THEN 1650
1600 IF RESP=1 THEN CALL READCARD(SUBJ$,ID$,BUFFER
     $(),PAGE,"C"):: USW=1
1610 IF RESP=2 THEN PAGE=1
1620 IF RESP=3 THEN PAGE=2
1630 IF RESP=5 THEN CALL PRINTCARD(SUBJ$,ID$,BUFFE
     R$())
1640 GOTO 1570
1650 SUBEND
1660 SUB LISTENTRY(IDLIST$(),SUBJECT$(,),NMAX,RCOD
     E)
1670 RCODE=0 :: DISPLAY AT(1,4)ERASE ALL:"Entries
     within Subject"
1680 CALL SEARCHSUBJ(SUBJECT$(,),NMAX,P,RCODE)
1690 DISPLAY AT(4,1):" " :: DISPLAY AT(6,1):" "
1700 DISPLAY AT(22,2):"SPACE  to continue"
1710 DISPLAY AT(23,2):"ENTER  to end list"
1720 DISPLAY AT(3,1):SUBJECT$(P,1)
```

```
1730 P2=1 :: LIM=LEN(SUBJECT$(P,2))
1740 FOR ROW=5 TO 18
1750 IF P2>LIM THEN CALL BLANKLINES(ROW,18):: GOTO
      1810
1760 IDN$=SEG$(SUBJECT$(P,2),P2,3):: P2=P2+3
1770 FOR I=1 TO NMAX
1780 IF IDN$=SEG$(IDLIST$(I),1,3)THEN DISPLAY AT(R
     OW,2):SEG$(IDLIST$(I),4,25):: GOTO 1800
1790 NEXT I :: RCODE=-1 :: GOTO 1840
1800 NEXT ROW
1810 CALL GETKEY(R,"PRESS"):: IF R=13 THEN 1840 EL
     SE IF R<>32 THEN 1810
1820 IF P2<=LIM THEN 1740
1830 P=P+1 :: IF P<=NMAX AND SUBJECT$(P,1)<>"" THE
     N 1720
1840 SUBEND
1850 SUB SEARCHSUBJ(SUBJECT$(,),NMAX,P,RCODE)
1860 RCODE=0 :: DISPLAY AT(3,1):"With which subjec
     t do you{3 SPACES}wish to begin the list?"
1870 ACCEPT AT(6,2)BEEP:SBJ$
1880 IF SBJ$="" THEN P=1 :: GOTO 1930
1890 DISPLAY AT(20,1):"Searching ..."
1900 FOR I=1 TO NMAX
1910 IF POS(SUBJECT$(I,1),SBJ$,1)<>0 THEN P=I :: G
     OTO 1930
1920 NEXT I :: P=1
1930 DISPLAY AT(20,1):" "
1940 SUBEND
1950 SUB MATCHID(ID$,IDLIST$(),NMAX,RCODE)
1960 RCODE=0 :: FOR I=1 TO NMAX
1970 IF ID$=SEG$(IDLIST$(I),4,25)THEN RCODE=I :: G
     OTO 1990
1980 NEXT I
1990 SUBEND
2000 SUB ADDID(ID$,IDLIST$(),NMAX,NEXTREC,RCODE)
2010 RCODE=0 :: FOR I=1 TO NMAX :: T$=SEG$(IDLIST$
     (I),4,25)
2020 IF ID$=T$ THEN RCODE=I :: GOTO 2110
2030 IF IDLIST$(I)="" OR ID$<T$ THEN P1=I :: GOTO
     2050
2040 NEXT I :: RCODE=-1 :: GOTO 2110
2050 FOR I=P1 TO NMAX
2060 IF IDLIST$(I)="" THEN P2=I :: GOTO 2080
2070 NEXT I :: RCODE=-1 :: GOTO 2110
2080 FOR I=P2 TO P1+1 STEP -1 :: IDLIST$(I)=IDLIST
     $(I-1):: NEXT I
2090 T$=STR$(NEXTREC):: T$=RPT$("0",3-LEN(T$))&T$
2100 IDLIST$(P1)=T$&ID$ :: IF P2<NMAX THEN IDLIST$
     (P2+1)=""
```

```
2110 SUBEND
2120 SUB ADDSUBJ(SUBJ$,SUBJECT$(,),NMAX,NEXTREC,RC
     ODE)
2130 RCODE=0 :: FOR I=1 TO NMAX
2140 IF SUBJ$=SUBJECT$(I,1)THEN 2170
2150 IF SUBJECT$(I,1)="" OR SUBJ$<SUBJECT$(I,1)THE
     N P1=I :: GOTO 2200
2160 NEXT I :: RCODE=-1 :: GOTO 2300
2170 IF LEN(SUBJECT$(I,2))>240 THEN RCODE=-1 :: GO
     TO 2300
2180 T$=STR$(NEXTREC):: T$=RPT$("0",3-LEN(T$))&T$
2190 SUBJECT$(I,2)=SUBJECT$(I,2)&T$ :: GOTO 2300
2200 FOR I=P1 TO NMAX
2210 IF SUBJECT$(I,1)="" THEN P2=I :: GOTO 2230
2220 NEXT I :: RCODE=-1 :: GOTO 2300
2230 FOR I=P2 TO P1+1 STEP -1
2240 SUBJECT$(I,1)=SUBJECT$(I-1,1):: SUBJECT$(I,2)
     =SUBJECT$(I-1,2)
2250 NEXT I
2260 SUBJECT$(P1,1)=SUBJ$
2270 T$=STR$(NEXTREC):: T$=RPT$("0",3-LEN(T$))&T$
2280 SUBJECT$(P1,2)=T$
2290 IF P2<NMAX THEN SUBJECT$(P2+1,1)="" :: SUBJEC
     T$(P2+1,2)=""
2300 SUBEND
2310 SUB LISTSUBJ(SUBJ$(,),N)
2320 DISPLAY AT(1,7)ERASE ALL:"List of Subjects"
2330 DISPLAY AT(22,2):"SPACE  to continue"
2340 DISPLAY AT(23,2):"ENTER  to end subject list"
2350 P=1
2360 FOR ROW=3 TO 17
2370 IF SUBJ$(P,1)="" OR P>N THEN CALL BLANKLINES(
     ROW,17):: P=1 :: GOTO 2390 :: ELSE DISPLAY AT
     (ROW,I):SUBJ$(P,1):: P=P+1
2380 NEXT ROW
2390 CALL GETKEY(R,"PRESS"):: IF R=32 THEN 2360 EL
     SE IF R<>13 THEN 2390
2400 SUBEND
2410 SUB BLANKLINES(BEGINL,ENDL)
2420 FOR I=BEGINL TO ENDL :: DISPLAY AT(I,1):" " :
     : NEXT I
2430 SUBEND
2440 SUB MAINMENU(FCODE,N)
2450 CALL DRAWBORDER
2460 DISPLAY AT(3,5):"Electronic Card File"
2470 DISPLAY AT(6,1):"Do you want to:"
2480 DISPLAY AT(08,2):"1  list subjects"
2490 DISPLAY AT(09,2):"2  list entries by subject"
2500 DISPLAY AT(10,2):"3  browse a subject"
```

```
2510 DISPLAY AT(11,2):"4  browse entries"
2520 DISPLAY AT(12,2):"5  retrieve an entry"
2530 DISPLAY AT(14,2):"6  add an entry"
2540 DISPLAY AT(16,2):"7  return to BASIC"
2550 DISPLAY AT(23,1):"Number of entries";N
2560 CALL GETKEY(R,"Enter your selection."):: IF R
     <49 OR R>55 THEN 2560 ELSE FCODE=R-48
2570 SUBEND
2580 SUB PRINTCARD(SUBJ$,ID$,BUFFER$())
2590 IF F THEN 2610
2600 OPEN #1:"RS232.BA=1200" :: F=1
2610 PRINT #1:TAB(10);"SUBJECT:";TAB(25);SUBJ$
2620 PRINT #1:TAB(10);"IDENTIFIER:";TAB(25);ID$
2630 PRINT #1:TAB(10);"NOTES:"
2640 PRINT #1
2650 FOR I=1 TO 20
2660 IF BUFFER$(I)<>"" THEN PRINT #1:TAB(15);BUFFE
     R$(I)
2670 NEXT I
2680 FOR I=1 TO 5 :: PRINT #1 :: NEXT I
2690 SUBEND
2700 SUB READCARD(SUBJ$,ID$,BUFFER$(),PAGE,TRANS$)
2710 IF PAGE<>1 OR TRANS$<>"A" THEN 2780
2720 ACCEPT AT(3,2)SIZE(-25)BEEP:SUBJ$ :: CALL CLE
     ANSTR(SUBJ$)
2730 IF SUBJ$="" THEN DISPLAY AT(20,2)BEEP:"**Plea
     se enter subject**" :: GOTO 2720
2740 DISPLAY AT(20,1):" "
2750 ACCEPT AT(6,2)SIZE(-25)BEEP:ID$ :: CALL CLEAN
     STR(ID$)
2760 IF ID$="" THEN DISPLAY AT(20,2)BEEP:"**Please
      enter identifier**" :: GOTO 2750
2770 DISPLAY AT(20,2):" "
2780 IF PAGE=2 THEN BIAS=10 ELSE BIAS=0
2790 FOR I=1 TO 10
2800 ACCEPT AT(8+I,2)SIZE(-24)BEEP:T$ :: CALL CLEA
     NSTR(T$):: IF T$="" AND BUFFER$(BIAS+I)="" TH
     EN 2820 ELSE BUFFER$(BIAS+I)=T$
2810 NEXT I
2820 SUBEND
2830 SUB CLEANSTR(T$)
2840 T=POS(T$,CHR$(136),1):: IF T THEN T$=SEG$(T$,
     1,T-1)
2850 SUBEND
2860 SUB WRITECARD(SUBJ$,ID$,BUFFER$(),PAGE)
2870 IF FL$="" THEN FL$=RPT$(CHR$(136),25)
2880 DISPLAY AT(3,2)SIZE(25):SUBJ$&FL$ :: DISPLAY
     AT(6,2)SIZE(25):ID$&FL$
2890 IF PAGE=2 THEN BIAS=10 ELSE BIAS=0
```

```
2900 DISPLAY AT(8,22)SIZE(2):PAGE
2910 FOR I=1 TO 10 :: DISPLAY AT(8+I,2)SIZE(24):BU
     FFER$(BIAS+I)&FL$ :: NEXT I
2920 SUBEND
2930 SUB GETRESPONSE(RESPONSE)
2940 RESPONSE=0 :: CALL GETKEY(R,"PRESS ONE OF THE
      FOLLOWING")
2950 IF R=13 THEN RESPONSE=4 !ENTER
2960 IF R=82 THEN RESPONSE=1 !R
2970 IF R=49 THEN RESPONSE=2 !1
2980 IF R=50 THEN RESPONSE=3 !2
2990 IF R=80 THEN RESPONSE=5 !P
3000 IF RESPONSE=0 THEN 2940
3010 SUBEND
3020 SUB GETKEY(R,MSG$)
3030 DISPLAY AT(20,1):MSG$
3040 CALL SOUND(200,262,0):: K=0
3050 CALL KEY(3,R,S):: IF S<>1 THEN K=K+1 :: IF K<
     250 THEN 3050 ELSE 3040
3060 CALL KEY(5,R1,S):: DISPLAY AT(20,1):" "
3070 SUBEND
3080 SUB GETYN(R$)
3090 CALL GETKEY(R,"")
3100 IF R=89 THEN R$="Y" ELSE IF R=78 THEN R$="N"
     ELSE 3090
3110 SUBEND
3120 SUB DRAWCARD
3130 IF F=0 THEN CALL CHAR(132,"000000FF"):: CALL
     CHAR(136,"0000000000427E"):: CALL COLOR(14,13
     ,1):: F=1
3140 CALL DRAWBORDER
3150 CALL HCHAR(4,2,132,30):: CALL HCHAR(7,2,132,3
     0):: CALL HCHAR(19,2,132,30):: CALL HCHAR(21,
     2,132,30)
3160 DISPLAY AT(2,1):"Subject:"
3170 DISPLAY AT(5,1):"Identifier:"
3180 DISPLAY AT(8,1):"Notes:" :: DISPLAY AT(8,25):
     "of 2"
3190 DISPLAY AT(22,1):"R to revise. 1,2 for page #
     ."
3200 DISPLAY AT(23,1):"P to print. ENTER when done
     ."
3210 SUBEND
3220 SUB DRAWBORDER
3230 CALL CLEAR
3240 IF P1$="" THEN P1$="FF"&RPT$("00",7)&RPT$("01
     ",8)&RPT$("00",7)&"FF"&RPT$("80",8):: CALL CH
     AR(128,P1$):: CALL COLOR(13,5,1)
```

```
3250 CALL HCHAR(1,2,128,30):: CALL VCHAR(1,1,129,2
     4):: CALL HCHAR(24,2,130,30):: CALL VCHAR(1,3
     2,131,24)
3260 SUBEND
3270 SUB TITLESCREEN
3280 P1$=RPT$("01",8)&RPT$("00",7)&"FF"&RPT$("01",
     7)&"FFFF"&RPT$("01",7):: P5$="FF"
3290 CALL CHAR(128,P1$):: CALL CHAR(132,P5$):: CAL
     L COLOR(13,2,1)
3300 CALL CHAR(136,SEG$(P1$,1,16)&SEG$(P1$,49,16)&
     P5$):: CALL COLOR(14,2,9)
3310 CALL CLEAR
3320 CALL VCHAR(3,1,128,22):: CALL HCHAR(24,2,129,
     27):: CALL HCHAR(24,29,130)
3330 CALL VCHAR(3,29,128,21):: CALL HCHAR(3,29,131
     ):: CALL HCHAR(3,2,132,27)
3340 CALL HCHAR(2,3,128):: CALL HCHAR(1,5,128):: C
     ALL HCHAR(23,30,132)
3350 CALL HCHAR(21,31,132):: CALL HCHAR(2,4,138,26
     ):: CALL HCHAR(1,6,138,25)
3360 CALL VCHAR(3,30,136,20):: CALL VCHAR(2,31,136
     ,19):: CALL HCHAR(2,30,137):: CALL HCHAR(1,31
     ,137)
3370 DISPLAY AT(7,4)SIZE(20):"Electronic Card File
     "
3380 DISPLAY AT(12,10)SIZE(8):"for  the"
3390 DISPLAY AT(14,9)SIZE(10):"TI-99  4/A"
3400 DISPLAY AT(16,7)SIZE(14):"Home  Computer"
3410 CALL DELAY(2000)
3420 SUBEND
```

# Chapter 7

# Appointment Calendar

# Appointment Calendar

One of the more trying aspects of running a household is making sure that everybody is at the right place at the right time. There are church, school, and sports activities for the youngsters. There are club meetings, dinner parties, and socials for the parents. For the entire family there are assorted doctor and dentist appointments, outings, and computer club meetings.

How do you keep track of all these activities? You probably have a household appointment calendar. Does everyone in the family always know where it is? Is it faithfully updated?

Often the need arises for a record of appointments that were scheduled for last month or even several months ago. Who keeps records of such things? Don't we all just tear off and discard last month's calendar at the start of a new month?

## Appointment Calendar

In the last chapter you looked at an electronic card file application. This chapter presents a complementary program tailored to a specific purpose.

Your TI-99/4A, with the "Appointment Calendar" program, can help you keep track of your household's appointments. You can look at one month at a time. You can type in the appointments for the month. Then you can save all the appointments on tape or disk. (Appointment Calendar expects you to enter in the program your years of interest ahead of time. More will be said about this later.)

Later on, you can look at the appointments for that month. A desired day's appointments may be called up and reviewed. You may change the details for any appointments that may have already been recorded. Of course, you can add new appointments at any time.

So the Appointment Calendar program works just like the appointment calendar that you may now be keeping. Even some of the same operating procedures apply. The TI cannot

automatically record or announce upcoming events.

The Appointment Calendar program will introduce some new operating procedures. Household members may want you to explain how to use the program. Also, a convenient storage place for the appointment tapes or disks should be set up.

### System Requirements

The Appointment Calendar program is designed for a 16K 99/4A with Extended BASIC. Figure 7-1 shows the equipment that you will need.

Notice that the 32K memory and disk drive equipment are optional. The addition of memory will, with the proper program changes, allow for more appointments for a month. In a 16K system, there is room for about 50 appointments per month.

A disk drive does not introduce any additional capabilities as far as Appointment Calendar goes. However, the storage and retrieval of monthly appointment information will be much faster than with a cassette recorder.

The program does not require, nor can it use, a printer. The calendar and appointments are shown strictly on your TV or video monitor.

**Figure 7-1. Appointment Calendar System Requirements**

Required:

—TI-99/4A console
—Extended BASIC
—Cassette tape recorder

Optional:

—32K memory expansion
—Disk drive
—Disk controller

### Setting Up the Calendar

Appointment Calendar is not a general-purpose calendar program. It is not designed for displaying any arbitrary calendar such as June 1139. Rather, the program is intended to track appointments within a certain data range that you specify.

Before using Appointment Calendar, check that the proper calendars have been stored in the program. You can

store the calendars for several years. So, the job of updating Appointment Calendar should not be a difficult one. You will have to do it only once every several years.

There are two steps involved with setting up the calendar. First, locate the subprogram called DATELIM. There are two variables in this subprogram—Y1 and Y2. Set these variables to the lower and upper limits of the date range that you want. The following line established 1984 and 1985 as the years of interest:

```
2390 Y1=1984 :: Y2=1985
```

Notice that the full four-digit year number is used. This completes the first step.

The second step is a little more involved. Find the subprogram MONTHSET. Locate the DATA statements in this subprogram. There are DATA statements for the names of the months and the number of days in the months. Do *not* change these. You are looking for the DATA statements just after the remark:

```
2460 REM ** BEGIN CALENDAR DATA STATEMENTS **
```

The first DATA statement contains just one number. That number tells the number of years that Appointment Calendar should handle. Since we are dealing with two years, the DATA statement will read:

```
2480 DATA 2
```

Following this DATA statement, there should be one additional DATA statement per year (lines 2500 and 2510). Each of these DATA statements should contain 13 numbers:

Number 1 the four-digit year number
Numbers 2–13 the day number of the first of the month.

Numbers 2–13 require a bit of explanation. Day numbers range from 1 to 7. Sunday is day 1, Monday is day 2, and so on, with Saturday being day 7. Consider the first three months of 1984.

| Month | Day | Day Number |
|---|---|---|
| January 1 | Sunday | 1 |
| February 1 | Wednesday | 4 |
| March 1 | Thursday | 5 |

The complete DATA statement for 1984 is:

```
2500 DATA 1984,1,4,5,1,3,6,1,4,7,2,5,7
```

As a double-check, count the numbers in the DATA statement. There should be 13. In addition, there should be two such statements—one for 1984 and the other for 1985.

In this manner, you can set up Appointment Calendar for any number of years. The program as shown in the listing will work for 1984 and 1985. 1984 is treated as a leap year.

Save a copy of the program after you've changed the DATA statements. It is this customized copy that you will be working with in the future.

### Operating the Appointment Calendar

Appointment Calendar is another menu-driven program. The main menu screen lists the major functions. The menu uses the single keystroke technique. Choose a function and type in the selection number.

There are five major functions:

1 specify a month and year,
2 load appointments,
3 review appointments,
4 save appointments, and
5 return to BASIC.

Selection 1 is used to tell Appointment Calendar with which year and month you will be working. The program can display a calendar for only one month on the screen at a time. But how does Appointment Calendar know what month?

Selection 1 will display another screen. You will be asked for a four-digit year number and a two-digit month number. The year number must fall within the range that has been set by the DATA statements discussed earlier. Otherwise, Appointment Calendar will keep asking you for a year number. The month number must fall within the range 01 through 12 inclusive. After you've specified the year and month, the main menu will be displayed again.

Generally speaking, use Selection 1 only for the first time that you're making appointments for a month. Selection 1 erases *all* appointment information from memory. Be careful here.

Selection 2 loads appointments into the computer's memory. Any appointments already in memory are erased. Thus, you must load all the old appointments first before making any new ones. You may use any one of the following input devices:

1 tape,
2 disk, or
3 other.

If you choose option 2 or 3, you will be asked for a complete filename. For tape input, follow the loading instructions that will be displayed on the screen. The main menu will be displayed again after the appointments have been read in.

Loading appointments automatically sets the year and month. So if you use Selection 2 for loading appointments, you do not need Selection 1. You only need Selection 1 as a way to get started with a particular year and month in the first place.

Can Appointment Calendar be confused? No. Certain safeguards are built-in. For example, suppose you choose Selection 1 and set up February 1984. Next, you immediately use Selection 2. Either on purpose or by accident, you load January 1984's appointment tape in the computer. Will you end up with January's days showing up on February's calendar? Certainly not. Appointment Calendar will realize what happened. It will display a correct calendar and a correct set of appointments for January 1984. You will notice the mistake: The calendar will be labeled January instead of February.

Appointments are reviewed, changed, and added all through Selection 3. These functions will be discussed in a separate section.

Selection 4, saving appointments, is an important function. Always remember to save your appointments. This is especially true if you plan to look at another month. Selections 1 and 2 erase all appointments from memory. If you change any appointments or add any new ones, be certain that you've saved them. You may use tape, disk, or some other device for appointment storage.

Selection 5 provides a way to return to BASIC. Again, appointments should be saved before leaving Appointment Calendar.

### Review Appointments

Selection 3, review appointments, is the heart of Appointment Calendar. This selection shows you the calendar.

When you choose Selection 3, the calendar for the year and month set by Selection 1 or 2 is displayed. The calendar is shown in a grid produced by graphics characters. Figure 7-2

shows the calendar for January 1984. Notice that the grid has room for six weeks. This was done so that any month could be displayed. The calendar does not fit into nice, even, four-week patterns, so there will always be some empty squares on the calendar. Just ignore these.

The calendar is clearly labeled. The year and month are shown above the calendar. Double-check the year and month before you begin making appointments.

You will notice a set of instructions below the calendar. These represent the keys that are active. A special function is associated with each of these keys. A particular function is called up by typing the appropriate key.

### Selecting Dates

Before appointments can be reviewed, changed, or added, the particular appointment days must be selected. Look at Figure 7-2 again. There is a little arrow in the upper left corner. This arrow is the day selector or cursor. The idea is to move the arrow to each of the days of interest. When the arrow lands at the proper day, press ENTER. This tells Appointment Calendar to remember that day. You are going to do something with that day later.

To move the arrow around, use the following keys:

| | |
|---|---|
| D | move the arrow forward, |
| S | move the arrow backward, |
| E | move the arrow up, |
| X | move the arrow down, and |
| ENTER | select a day. |

Do not hold the FCTN key down when you're moving the arrow. The FCTN key is not necessary for arrow movement.

When you select a day, a little tag will appear in the upper left corner of the day's grid. This tag lets you know the days that have been selected. Suppose you press ENTER at the wrong date and make an incorrect selection. Just press ENTER again. You will see the tag disappear. In effect, ENTER toggles the tag on and off.

How can you tell on what days you already have appointments? Appointment Calendar uses another tag. The appointment tag is on the bottom left-hand corner of the day's grid. If you load appointments from tape, you can quickly spot the

appointment days. Now, if you want to see the specific appointments, select the day or days by means of the arrow, as described above.

You may return to the main menu at any time. Press FCTN BACK. This does not alter the appointments in any way. Nor does it affect the days that you've selected. They remain marked with the selection tag.

**Figure 7-2. Appointment Calendar Display**
**January 1984**

```
→1    2    3    4    5    6    7
 8    9   10   11   12   13   14
15   16   17   18   19   20   21
22   23   24   25   26   27   28
29   30   31

move arrow to desired date
D        forward   S        back
E        up        X        down
ENTER    mark date
PROC'D   update appointments
BACK     return to menu
```

## Adding and Changing Appointments

After all the days have been selected and marked, press FCTN PROC'D. This takes Appointment Calendar out of the calendar mode and puts it in the appointment mode. The calendar display will be replaced by an appointment display. You can look at each appointment for each selected day, or go from selected day to selected day while skipping over the appointment details.

Figure 7-3 shows what the appointment display looks like. The name of the day (Monday) and the day of the month (1) are at the top of the display. Then there are four items of information for each appointment:

—time,
—place,
—person being met, and
—subject or purpose of the appointment.

**Figure 7-3. Appointment Display**

```
                Monday 1
Time:           2:30
Place:          Medical Building
Person:         Dr. Drill
Subject:        Cavities

1 Appointment for the day
AID             make a new appt.
BACK            return to menu
ENTER           review next appt.
PROC'D          review next date
REDO            change appt. info.
```

Following this is a count of the number of appointments made for the day. Last, the active function keys are listed.

At this point, Appointment Calendar will recognize the following function keys:

| | |
|---|---|
| AID | make a new appointment, |
| BACK | return to the menu, |
| ENTER | review the next appointment, |
| PROC'D | review the next selected date, and |
| REDO | change appointment information. |

These functions are invoked by holding down the FCTN key and then pressing one of the keys listed above.

Notice that there is no delete appointment key. A record is kept of everything. You can always mark an appointment as cancelled or rescheduled. If an entry is made by mistake, you can say something to that effect in the subject field.

Let's take a brief look at each of these functions.

**AID** lets you make a new appointment. The cursor is positioned at the Time field. Type in all the appropriate information for the four fields. When you have finished adding an appointment, the appointment counter will be incremented by one.

**BACK** returns you to the main menu. Use this option when you are finished reviewing or adding appointments.

**ENTER** shows you the next appointment, if any, for the day. If you keep pressing ENTER, you will cycle through all of the day's apointments.

**PROC'D** takes you to the next selected date. This is a way of quickly looking at each selected day. If all of the

selected days have been displayed, PROC'D will take you back to the main menu.

**REDO** lets you make changes to any appointment information. The date of the appointment, of course, cannot be changed this way. If the date changes, the appointment should be added on the new date.

## Restrictions

There are some restrictions as to the number of appointments that can be recorded. These restrictions are due to memory limitations. In a 16K TI, about 50 appointments a month can be stored. You can store 15 appointments per day as long as you don't exceed 50 appointments overall.

With more memory the storage capacity can be increased quite a bit. To increase the allowable number of appointments, increase the first dimension of the two-dimensional array APPT$. Set MAXAPPT so that it matches the new first dimension of APPT$.

To increase the permitted number of appointments per day, change the variable DAYAPPT. It was set to 15 arbitrarily so that all 50 appointments would not be used in one day.

## Summary of Operation

Appointment Calendar is a very useful program for keeping track of household appointments. The main menu screen provides the means of invoking the major functions of setting the year and month, loading and saving appointments, and reviewing appointments. The appointment review function is where most of the appointment work is done. A calendar for the year and month is displayed. You select the days of interest by manipulating a cursor. Then you proceed to the appointment review mode. Each of the selected days can be examined. You can add or change appointments as necessary. When you are finished, return to the main menu and save the appointments.

## How It Works

The structure chart for Appointment Calendar is shown in Figure 7-4. The program is composed of a series of subroutines and subprograms. The main program dimensions the arrays and initializes a few variables. The main menu is implemented as a subprogram. The menu subprogram is called over and over again until Appointment Calendar is instructed to return

to BASIC. The major processing functions are subroutines which are called from the main program.

All of the subroutines in Appointment Calendar have REM statements as their first line. The REM statements give the name of the subroutine and follow Figure 7-4 fairly closely.

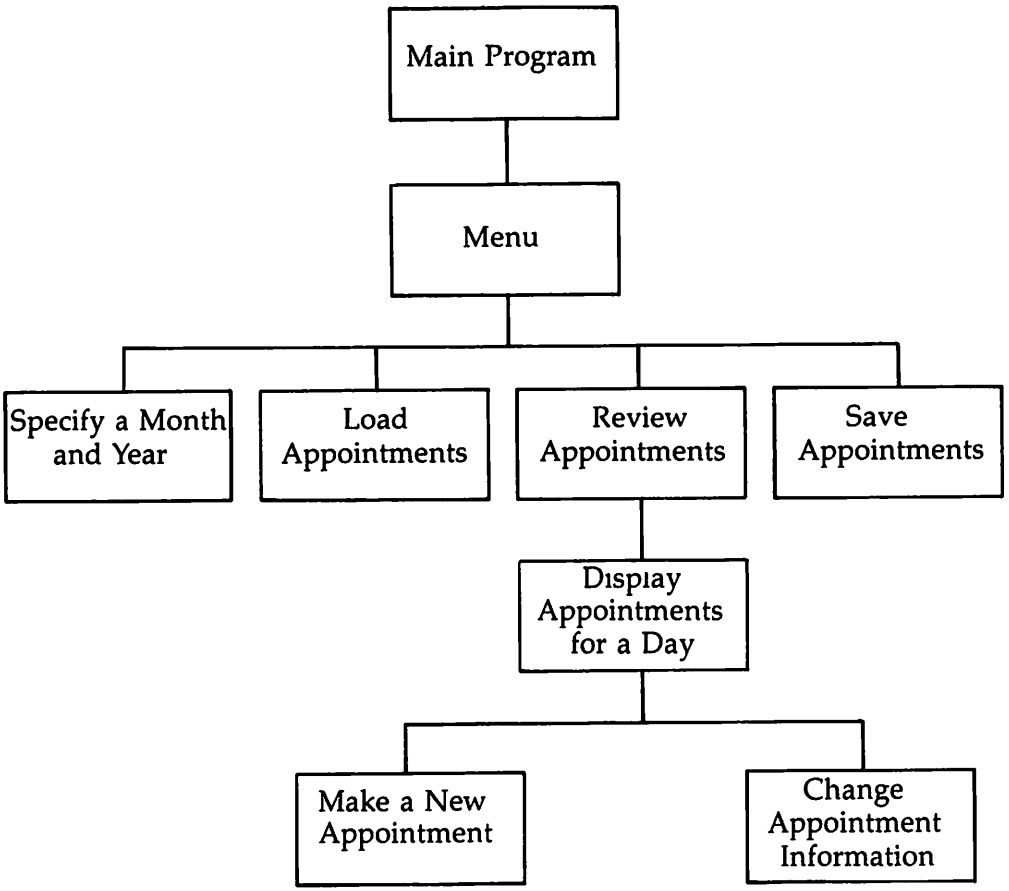
There are many subprograms in Appointment Calendar. In general, they are called from several places. Figure 7-5 lists all of the subprograms and gives a brief description of what they do. No attempt has been made to describe the subprogram parameters except in a few instances.

Two arrays used by Appointment Calendar are APPT$ and MARKLIST$. APPT$ is dimensioned as 50 by 3. The appointment information is stored in APPT$. The second dimension is used as follows:

| | |
|---|---|
| APPT$(X,0) | time |
| APPT$(X,1) | place |
| APPT$(X,2) | person |
| APPT$(X,3) | subject |

MARKLIST$, dimensioned 5 by 6 by 1, holds calendar information. The first dimension is for the weeks, which may range in number from zero to five. The second dimension represents the days of the week, which are numbered zero to six. Subscript zero of the third dimension tells whether or not a day has been marked for review. Similarly, the first subject of the third dimension tells if there is an appointment for the day.

**Figure 7-4. Appointment Calendar Program Structure**



### Figure 7-5. Appointment Calendar Subprograms

| | |
|---|---|
| DELAY— | A FOR-NEXT delay loop. |
| CLEARLIST— | Clears MARKLIST$; sets the number of appointments to zero. |
| GETDEVICE— | Obtains a device name for an input or an output operation. |
| SETCAL— | Gets the year and month from the user; calls DATELIM and MONTHSET. |
| DAYNAME— | Given a week number (0–5), day number (0–6), and the day number of the first of the month, returns the name of the day. |
| MARKCAL— | Moves the cursor around the calendar; marks selected dates. |
| SETCSR— | Moves the cursor (a sprite) to a specified week and day. |
| LABELCAL— | Prints the dates on the calendar. |
| DRAWGRID— | Draws the grid for the calendar by means of user-defined graphics characters. |
| GETKEY— | Displays a prompt message; waits for a key; periodically beeps a reminder tone. |
| MAINMENU— | Displays the main menu screen; gets the function selection from the user. |
| DATELIM— | Returns the beginning and ending years of Appointment Calendar's range. The limits are defined by the user. |
| MONTHSET— | Given a year and month, returns the day number of the first of the month minus one, the number of days in the month, and the name of the month. Contains DATA statements for setting up the calendar. The DATA statements are set by the user. |

### Program 7-1. Appointment Calendar

```
100 REM APPOINMENT CALENDAR
110 ON BREAK NEXT
120 DIM MARKLIST$(5,6,1),APPT$(50,3),A$(3)
130 NEXTAPPT=0 :: MAXAPPT=50 :: DAYAPPT=15
140 REM MAIN PROGRAM LOOP
```

```
150 CALL MAINMENU(FCODE,NEXTAPPT):: ON FCODE GOSUB
     180,220,340,450,160 :: GOTO 150
160 CALL CLEAR
170 END
180 REM SPECIFY A MONTH & YEAR
190 CALL CLEARLIST(MARKLIST$(,,),NEXTAPPT)
200 CALL SETCAL(MM$,YY$,BEGINMM,NDAYS)
210 RETURN
220 REM LOAD APPOINTMENTS
230 CALL GETDEVICE("Load Appointments",F$)
240 IF F$="CS1" THEN OPEN #1:"CS1",INPUT ,INTERNAL
    ,FIXED ELSE OPEN #1:F$,INPUT ,INTERNAL,VARIABL
    E
250 CALL CLEARLIST(MARKLIST$(,,),NEXTAPPT)
260 INPUT #1:MM$,YY$,BEGINMM,NDAYS
270 INPUT #1:W,D,T$ :: IF W>=0 THEN MARKLIST$(W,D,
    1)=T$ :: GOTO 270
280 INPUT #1:NEXTAPPT
290 FOR I=0 TO NEXTAPPT
300 FOR J=0 TO 3 :: INPUT #1:APPT$(I,J),:: NEXT J
310 NEXT I
320 CLOSE #1
330 RETURN
340 REM REVIEW APPOINMENTS
350 IF MM$="" THEN 440
360 CALL DRAWGRID :: CALL LABELCAL(MM$&" "&YY$,BEG
    INNM,NDAYS)
370 FOR W=0 TO 5 :: FOR D=0 TO 6
380 IF MARKLIST$(W,D,0)="Y" THEN CALL HCHAR(W*2+4,
    D*4+3,135)
390 IF MARKLIST$(W,D,1)<>"" THEN CALL HCHAR(W*2+6,
    D*4+3,136)
400 NEXT D :: NEXT W
410 CALL SETCSR(0,0):: CALL MARKCAL(MARKLIST$(,,),
    R):: CALL SETCSR(0,11)
420 IF R=15 THEN 440
430 IF R=12 THEN GOSUB 590
440 RETURN
450 REM SAVE APPOINTMENTS
460 CALL GETDEVICE("Save Appointments",F$)
470 IF F$="CS1" THEN OPEN #1:"CS1",OUTPUT,INTERNAL
    ,FIXED 192 ELSE OPEN #1:F$,OUTPUT,INTERNAL,VAR
    IABLE 254
480 PRINT #1:MM$;YY$;BEGINMM;NDAYS
490 FOR W=0 TO 5 :: FOR D=0 TO 6
500 IF MARKLIST$(W,D,1)<>"" THEN PRINT #1:W;D;MARK
    LIST$(W,D,1)
510 NEXT D :: NEXT W
520 PRINT #1:-1;-1;"FINI"
```

```
530 PRINT #1:NEXTAPPT
540 FOR I=0 TO NEXTAPPT
550 FOR J=0 TO 3 :: PRINT #1:APPT$(I,J);:: NEXT J
    :: PRINT #1
560 NEXT I
570 CLOSE #1
580 RETURN
590 REM REVIEW MARKED DATES
600 DISPLAY AT(4,1)ERASE ALL:"Time:"
610 DISPLAY AT(6,1):"Place:"
620 DISPLAY AT(8,1):"Person:"
630 DISPLAY AT(10,1):"Subject:"
640 DISPLAY AT(13,4):"Appointments for the day"
650 DISPLAY AT(18,1):"AID{5 SPACES}make a new appt
    ."
660 DISPLAY AT(19,1):"BACK{4 SPACES}return to menu
    "
670 DISPLAY AT(20,1):"ENTER{3 SPACES}review next a
    ppt."
680 DISPLAY AT(21,1):"PROC'D  review next date"
690 DISPLAY AT(22,1):"REDO{4 SPACES}change appt. i
    nfo."
700 FOR W=0 TO 5 :: FOR D=0 TO 6
710 IF MARKLIST$(W,D,0)<>"Y" THEN 920
720 CALL DAYNAME(W,D,BEGINMM,D$)
730 DISPLAY AT(1,1):" " :: DISPLAY AT(1,(28-LEN(D$
    ))/2):D$
740 NAPPT=LEN(MARKLIST$(W,D,1))/2
750 DISPLAY AT(13,1)SIZE(2):USING "##":NAPPT
760 REM DAY
770 P=1
780 T$=SEG$(MARKLIST$(W,D,1),P,2)
790 IF T$="" THEN DISPLAY AT(4,10):" " :: DISPLAY
    AT(6,10):" " :: DISPLAY AT(8,10):" " :: DISPLA
    Y AT(10,10):" " :: GOTO 850
800 P1=VAL(T$)
810 DISPLAY AT(4,10):APPT$(P1,0)
820 DISPLAY AT(6,10):APPT$(P1,1)
830 DISPLAY AT(8,10):APPT$(P1,2)
840 DISPLAY AT(10,10):APPT$(P1,3)
850 CALL GETKEY(R,"Press the desired FCTN key.")
860 IF R=15 THEN 940
870 IF R=1 THEN GOSUB 950
880 IF R=13 THEN P=P+2 :: IF P<=2*NAPPT THEN 780 E
    LSE 770
890 IF R=12 THEN 920
900 IF R=6 THEN GOSUB 1040
910 GOTO 850
920 NEXT D
```

```
930 NEXT W
940 RETURN
950 REM MAKE A NEW APPT
960 IF NEXTAPPT>MAXAPPT THEN DISPLAY AT(24,1):"**
    CALENDAR IS FULL **" :: CALL DELAY(1000):: GOT
    O 1030
970 IF NAPPT>DAYAPPT THEN DISPLAY AT(24,1):"** DAY
     IS FULL **" :: CALL DELAY(1000):: GOTO 1030
980 GOSUB 1080 !READ SCREEN
990 FOR J=0 TO 3 :: APPT$(NEXTAPPT,J)=A$(J):: NEXT
     J
1000 T1$=STR$(NEXTAPPT):: IF NEXTAPPT<10 THEN T1$=
     "0"&T1$
1010 IF NAPPT<=DAYAPPT THEN MARKLIST$(W,D,1)=MARKL
     IST$(W,D,1)&T1$
1020 NEXTAPPT=NEXTAPPT+1 :: NAPPT=NAPPT+1 :: DISPL
     AY AT(13,1)SIZE(2):USING "##":NAPPT
1030 RETURN
1040 REM CHANGE APPT INFO
1050 GOSUB 1080 !READ SCREEN
1060 FOR J=0 TO 3 :: APPT$(P1,J)=A$(J):: NEXT J
1070 RETURN
1080 REM READ SCREEN
1090 ACCEPT AT(4,10)SIZE(-10)BEEP:A$(0):: IF A$(0)
     ="" THEN 1090
1100 FOR J=1 TO 3
1110 ACCEPT AT(2*J+4,10)SIZE(-19)BEEP:A$(J):: IF A
     $(J)="" THEN 1110
1120 NEXT J
1130 RETURN
1140 SUB DELAY(N)
1150 FOR I=1 TO N :: NEXT I
1160 SUBEND
1170 SUB CLEARLIST(A$(,,),N)
1180 FOR W=0 TO 5 :: FOR D=0 TO 6 :: A$(W,D,0)=""
     :: A$(W,D,1)="" :: NEXT D :: NEXT W :: N=0
1190 SUBEND
1200 SUB GETDEVICE(MSG$,DNAME$)
1210 DISPLAY AT(1,(28-LEN(MSG$))/2)ERASE ALL:MSG$
1220 DISPLAY AT(8,1):"Which device do you want?"
1230 DISPLAY AT(10,3):"1  Tape"
1240 DISPLAY AT(12,3):"2  Disk"
1250 DISPLAY AT(14,3):"3  Other"
1260 CALL GETKEY(R,"Type your selection ->")
1270 IF R<49 OR R>51 THEN 1260
1280 IF R=49 THEN DNAME$="CS1" :: GOTO 1320
1290 IF R=50 THEN DISPLAY AT(16,1):"Type the disk
     file name.": :"DSK1."
```

```
1300 IF R=51 THEN DISPLAY AT(16,1):"Type the compl
     ete file name."
1310 ACCEPT AT(18,1)SIZE(-20)VALIDATE(UALPHA,DIGIT
     ,"=.")BEEP:DNAME$ :: IF DNAME$="" THEN 1310
1320 SUBEND
1330 SUB SETCAL(MM$,YY$,BEGINMM,NDAYS)
1340 CALL DATELIM(Y1,Y2)
1350 DISPLAY AT(1,4)ERASE ALL:"Specify a Month & Y
     ear"
1360 DISPLAY AT(4,1):"What year?{3 SPACES}1983"
1370 DISPLAY AT(6,1):"What month?{4 SPACES}01"
1380 DISPLAY AT(10,1):"- Type the 4 digit year."
1390 DISPLAY AT(12,1):"- Type the 2 digit month":"
       number."
1400 ACCEPT AT(4,14)SIZE(-4)VALIDATE(DIGIT)BEEP:YY
1410 IF YY<Y1 OR YY>Y2 THEN 1400 ELSE YY$=STR$(YY)
1420 ACCEPT AT(6,16)SIZE(-2)VALIDATE(DIGIT)BEEP:MM
1430 IF MM<1 OR MM>12 THEN 1420
1440 CALL MONTHSET(YY,MM,BEGINMM,NDAYS,MM$)
1450 SUBEND
1460 SUB DAYNAME(WKNO,DYNO,BEGINMM,DAY$)
1470 IF F$="Y" THEN 1510
1480 DATA SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY
     ,FRIDAY,SATURDAY,Y
1490 DIM DN$(6):: RESTORE 1480
1500 FOR I=0 TO 6 :: READ DN$(I):: NEXT I :: READ
     F$
1510 D=WKNO*7+DYNO-BEGINMM+1
1520 DAY$=DN$(DYNO)&" "&STR$(D)
1530 SUBEND
1540 SUB MARKCAL(MARKLIST$(,,),R)
1550 DISPLAY AT(17,1):"move arrow to desired date"
1560 DISPLAY AT(19,3):"D  forward{4 SPACES}S  back
     "
1570 DISPLAY AT(20,3):"E  up{9 SPACES}X  down"
1580 DISPLAY AT(22,1):"ENTER{3 SPACES}mark date"
1590 DISPLAY AT(23,1):"PROC'D  update appointments
     "
1600 DISPLAY AT(24,1):"BACK{4 SPACES}return to men
     u"
1610 X,Y=0
1620 DX,DY=0 :: CALL KEY(3,R,S):: IF S=0 THEN 1620
1630 IF R=12 OR R=15 THEN SUBEXIT
1640 IF R=13 THEN 1770
1650 IF R=69 THEN DY=-1
1660 IF R=88 THEN DY=1
1670 IF R=68 THEN DX=1
1680 IF R=83 THEN DX=-1
1690 IF DX=0 AND DY=0 THEN 1620
```

```
1700 X=X+DX :: Y=Y+DY
1710 IF X>6 THEN X=0
1720 IF X<0 THEN X=6
1730 IF Y>5 THEN Y=0
1740 IF Y<0 THEN Y=5
1750 CALL SETCSR(X,Y)
1760 GOTO 1620
1770 REM MARK DATE
1780 R=Y*2+5 :: C=X*4+5
1790 CALL GCHAR(R,C,CH):: IF CH=32 THEN 1620
1800 R=Y*2+4 :: C=X*4+3 :: CALL GCHAR(R,C,CH)
1810 IF CH=128 THEN CH2=135 :: MARKLIST$(Y,X,0)="Y
     " ELSE CH2=128 :: MARKLIST$(Y,X,0)="N"
1820 CALL HCHAR(R,C,CH2):: GOTO 1620
1830 SUBEND
1840 SUB SETCSR(X,Y)
1850 IF F$="" THEN CALL SPRITE(#1,134,05,200,1)::
     F$="Y"
1860 R=Y*2+5 :: DOTR=(R-1)*8+1
1870 C=X*4+3 :: DOTC=(C-1)*8+1
1880 CALL LOCATE(#1,DOTR,DOTC)
1890 SUBEND
1900 SUB LABELCAL(T$,MMBEGIN,NDAYS)
1910 K=0 :: D=1
1920 FOR R=5 TO 15 STEP 2 :: FOR C=2 TO 26 STEP 4
1930 IF K>=MMBEGIN AND D<=NDAYS THEN DISPLAY AT(R,
     C)SIZE(2):USING "##":D :: D=D+1
1940 K=K+1
1950 NEXT C :: NEXT R
1960 DISPLAY AT(1,(28-LEN(T$))/2):T$
1970 SUBEND
1980 SUB DRAWGRID
1990 IF F$="Y" THEN 2100
2000 CALL CHAR(128,"000000FF")!HORIZONTAL LINE
2010 CALL CHAR(129,RPT$("01",8))!VERTICAL LINE
2020 CALL CHAR(130,"010101FF01010101")!CROSS
2030 CALL CHAR(131,"000000FF01010101")!TOP ANGLE
2040 CALL CHAR(132,"01010FF")!BOTTOM ANGLE
2050 CALL CHAR(133,RPT$("80",8))!RIGHT EDGE
2060 CALL CHAR(134,"000406FF0604")!CURSOR
2070 CALL CHAR(135,"000000FFFCFCFCFC")!MARKER
2080 CALL CHAR(136,"00FCFCFF")!APPT
2090 CALL COLOR(13,16,1,14,16,1):: F$="Y"
2100 CALL CLEAR
2110 FOR C=2 TO 26 STEP 4 :: CALL VCHAR(4,C,129,13
     ):: NEXT C
2120 FOR R=4 TO 16 STEP 2 :: CALL HCHAR(R,3,128,28
     ):: NEXT R
2130 FOR R=6 TO 14 STEP 2 :: FOR C=6 TO 26 STEP 4
     :: CALL HCHAR(R,C,130,1):: NEXT C :: NEXT R
```

```
2140 FOR C=6 TO 26 STEP 4 :: CALL HCHAR(4,C,131,1)
     :: NEXT C
2150 FOR C=6 TO 26 STEP 4 :: CALL HCHAR(16,C,132,1
     ):: NEXT C
2160 CALL VCHAR(4,31,133,13)
2170 DISPLAY AT(3,2):"SUN MON TUE WED THU FRI SAT"
2180 SUBEND
2190 SUB GETKEY(R,MSG$)
2200 DISPLAY AT(24,1):MSG$
2210 CALL SOUND(200,262,6):: K=0
2220 CALL KEY(3,R,S):: IF S<>1 THEN K=K+1 :: IF K<
     250 THEN 2220 ELSE 2210
2230 CALL KEY(5,R1,S):: DISPLAY AT(24,1):" "
2240 SUBEND
2250 SUB MAINMENU(FCODE,N)
2260 DISPLAY AT(1,5)ERASE ALL:"Appointment Calenda
     r"
2270 DISPLAY AT(5,1):"Do you want to:"
2280 DISPLAY AT(7,3):"1  specify a month & year"
2290 DISPLAY AT(9,3):"2  load appoinments"
2300 DISPLAY AT(11,3):"3  review appoinments"
2310 DISPLAY AT(13,3):"4  save appoinments"
2320 DISPLAY AT(15,3):"5  return to BASIC"
2330 DISPLAY AT(18,1):USING "### Appointments ente
     red.":N
2340 CALL GETKEY(R,"Type your selection -> ")
2350 IF R<49 OR R>53 THEN 2340
2360 FCODE=R-48
2370 SUBEND
2380 SUB DATELIM(Y1,Y2)
2390 Y1=1984 :: Y2=1985
2400 SUBEND
2410 SUB MONTHSET(YY,MM,BEGINMM,NDAYS,MM$)
2420 DIM M$(12),ND(12),D(12):: IF M$(1)<>"" THEN 2
     460
2430 DATA JANUARY,31,FEBRUARY,28,MARCH,31,APRIL,30
     ,MAY,31,JUNE,30
2440 DATA JULY,31,AUGUST,31,SEPTEMBER,30,OCTOBER,3
     1,NOVEMBER,30,DECEMBER,31
2450 RESTORE 2430 :: FOR I=1 TO 12 :: READ M$(I),N
     D(I):: NEXT I
2460 REM ** BEGIN CALENDAR DATA STATEMENTS **
2470 REM YEARS OF DATA
2480 DATA 2
2490 REM year and day number (1-7) of 1st day of m
     onth
2500 DATA 1984,1,4,5,1,3,6,1,4,7,2,5,7
2510 DATA 1985,3,6,6,2,4,7,2,5,1,3,6,1
2520 REM
```

```
2530 RESTORE 2480 :: READ Y :: FOR I=1 TO Y
2540 READ Y1 :: FOR J=1 TO 12 :: READ D(J):: NEXT
     J
2550 IF Y1=YY THEN BEGINMM=D(MM)-1
2560 NEXT I
2570 NDAYS=ND(MM):: MM$=M$(MM)
2580 IF MM=2 AND(YY=1984)THEN NDAYS=NDAYS+1
2590 SUBEND
```

# Chapter 8

# Putting It All Together

# Putting It All Together

This book has presented several major application programs. Yes, each program works well by itself. Wouldn't it be nice, though, if all the programs were put together into one package? Then all the programs would be in one place and accessible from one menu screen. This would make things very convenient.

There are other benefits as well. As time goes by, you will develop or acquire additional programs. Would you like to put these new programs in with your other home information programs?

How can this be done? Clearly, all the programs could be combined into one great big program. This would require a great deal of memory, and you would have to be sure that all the line numbers were correct.

There are ways to overcome these obstacles. This chapter will explore some additional programming and disk system techniques that will result in a set of programs that will be called a "Home Information System." It must be pointed out, however, that all the programs in this chapter require at least one disk drive. If you do not have a disk drive, you can nevertheless learn some of the techniques. Who knows, maybe that urge to expand your system will overcome you.

## Description

Extended BASIC's ability to RUN a program from another program was briefly examined in Chapter 3. Using this technique, it's possible to set up a small menu program. The selections on the menu will turn out to be other programs. Thus, when a selection is made, the proper program will be brought into memory with the RUN "DSK1.XXXXX" statement. When the program is finished, it will do another RUN. This time, the menu program will be brought back. So another program may be selected.

It is therefore possible, for example, to perform a spreadsheet analysis, plot the results, and go back and do the spreadsheet again. All this can be done almost at the press of a button. There will be no need to type in the NEW, OLD, or RUN commands.

A second feature of Extended BASIC will also come in handy. When Extended BASIC is first started, it will look for a file called "DSK1.LOAD". That is why the disk drives turn on whenever you select Extended BASIC from the TI's menu. If DSK1.LOAD is found, Extended BASIC will try to run the program. In practical terms, this gives us an automatic start-up feature.

Even though it is possible to RUN a program from a program, large amounts of data can be passed between programs only by means of files. After a while, the free space on a diskette will diminish toward zero. The Disk Manager Command Module is the obvious way to monitor disk space. Unplugging Extended BASIC and plugging in the Disk Manager can be somewhat disruptive. Luckily, the disk directory information can be read from Extended BASIC just as though the directory were another file. So, you can write your own program that will tell us how much space we have left.

This chapter will look at three programs:

—a DSK1.LOAD program for automatic start-up,
—an application menu program, and
—a disk directory or catalog program.

These programs, plus selected application programs, will form our Home Information System.

## DSK1.LOAD

Program 8-1 shows a start-up program. Type it in and SAVE it using the name DSK1.LOAD. This particular program just displays a title screen and then invokes the system menu.

You can do other things with DSK1.LOAD. Perhaps you would like a more comprehensive set of operating instructions. DSK1.LOAD is a good place to put them. The instructions will be displayed as soon as Extended BASIC comes up. Since the instructions are in DSK1.LOAD, they don't take up valuable memory space from other programs.

DSK1.LOAD is also a good place for defining custom graphics characters. Maybe you would like true lowercase letters instead of TI's small capital letters. You can define the

characters in DSK1.LOAD. Whenever a program is run from another program, the character definitions remain in effect. The custom characters will return to their normal values only when you get back to Extended BASIC.

The same holds true for screen colors. You can set the screen to a particular color in DSK1.LOAD. That screen color will remain until you get back to Extended BASIC. If you are using a green or amber monochrome monitor, you might want to issue a

```
CALL SCREEN(15)
```

statement, which sets the screen color to white. This yields very nice results on a monochrome monitor.

There are several ways that you can make use of DSK1.LOAD. Try some of them. Program 8-1 represents the simplest approach. See if you can customize it for your own needs.

### System Menu

The system menu program, Program 8-2, uses the menu subprogram that was discussed in Chapter 2. The system menu presents a selection of programs. When you make a selection, the proper program is executed with the RUN statement. Type the program in and save it with the name DSK1.SYSTEMMENU.

The programs that you can select from the system menu are listed below. The programs must be stored on disk with the indicated names.

| Program | Program Name |
|---|---|
| 4-1 | DSK1.TINYPLAN |
| 5-1 | DSK1.BARCHARTS |
| 5-2 | DSK1.BARCHARTS2 |
| 6-2 | DSK1.DCARDFILE |
| 7-1 | DSK1.CALENDAR |
| 8-3 | DSK1.CATALOG |

Of course, you may add your own programs to the list.

All of these programs require a minor modification so you will see the system menu again. This way you can run the same program again with perhaps a different set of data. Maybe you just want to run a different program. Most programs finish with an END statement. The END statement returns control to Extended BASIC.

Locate the END statement in each program. Replace the END statement with these two lines:

```
DISPLAY AT(12,1) ERASE ALL:"Just a moment ... "
RUN "DSK1.SYSTEMMENU"
```

This way you'll see the system menu again when a program finishes.

### System Catalog

The system catalog program is shown as Program 8-3. The purpose of the system catalog program is to help keep track of data files. Consider the program a specialized version of the Disk Manager.

Program 8-3 will list the data files that are present on any drive—1, 2, or 3. The name of each file and its size in both sectors and bytes will be shown. The total number of bytes still available on the diskette will also be displayed.

In addition to simply displaying filenames, the system catalog program gives you the option of deleting data files. When the program is first started, it will ask you if you want to selectively delete files. If you do, the data filenames will be displayed one by one. You will have to specify whether you want to keep the file or delete it. If you do not choose to selectively delete files, a list of all data filenames will be presented. You cannot delete files in this mode.

Notice that the system catalog program displays only data files. The program intentionally hides program files. This was done to safeguard programs from being accidentally deleted.

The program works by reading the diskette directory. The directory works like a table of contents. Located in the directory are the names of the files stored on the diskette. The directory contains information about the files such as their type, size, and record length.

The directory is a RELATIVE INTERNAL file with the null string as the filename. The first record in the directory contains information about the diskette itself. Then, there is one directory record for each file. Program 8-3 simply reads the directory and displays its contents.

### Error Handling

You have seen how a nice program-to-program flow can be set up. Unfortunately, errors can disrupt this flow. Suppose you select the Bar Chart program. You already have the data

file that contains the X,Y pairs for plotting. What happens if you type in the filename incorrectly? Chances are pretty good that you will see an I/O error message on the screen. Extended BASIC will then take over. Our system menu has lost control.

Chapter 2 discussed how Extended BASIC can intercept errors. You might like to add a few lines of error-handling logic at the beginning of each of the application programs. The application programs all begin with line number 100. Thus, there is room for some additional lines.

Program 8-4 shows the error-handling statements. Type these statements in and save them on disk with the MERGE format. Then you can easily add them to the application programs with the MERGE command. Do not put these statements in any program until you are satisfied that the program is working properly in the first place. Do not put away your debugging tools until you are sure you are finished with them.

Program 8-4 does nothing more than display the various error codes. It waits until you press a key before resuming. At that point, the application program is started from the very beginning. This is not a very graceful recovery. Since the program is started from the beginning again, any work in progress up to the error will be lost. Errors should be few and far between. If most errors are due to miskeyed input filenames, not much will be lost. Generally, the application programs ask for the input filename before going very far with any processing. Again, experiment.

## Program 8-1. DSK1.LOAD

```
100 REM DSK1.LOAD
110 ON BREAK NEXT
120 !@P-
130 DISPLAY AT(4,5)ERASE ALL:"C O M P U T E ! ' S"
140 DISPLAY AT(6,11):"H O M E"
150 DISPLAY AT(8,4):"I N F O R M A T I O N"
160 DISPLAY AT(10,9):"S Y S T E M"
170 DISPLAY AT(19,1):"FOR:":" TI-99/4A EXTENDED BA
    SIC"
180 RUN "DSK1.SYSTEMMENU"
190 END
```

## Program 8-2. System Menu

```
100 REM SYSTEM MENU
110 ON BREAK NEXT
120 DATA 7
130 DATA "1  Tiny Plan Spreadsheet"
140 DATA "2  Bar Charts (Video)"
150 DATA "3  Bar Charts (Printer)"
160 DATA "4  Card File"
170 DATA "5  Appointment Calendar"
180 DATA "6  System Catalog"
190 DATA "7  Extended BASIC"
200 READ N
210 FOR I=1 TO N :: READ CHOICE$(I):: NEXT I
220 CALL MENU("APPLICATION MENU",CHOICE$(),N,FUNC)
230 IF FUNC=0 THEN 220
240 DISPLAY AT(12,1)ERASE ALL:"Just a moment ..."
250 ON FUNC GOTO 260,270,280,290,300,310,320
260 RUN "DSK1.TINYPLAN"
270 RUN "DSK1.BARCHARTS"
280 RUN "DSK1.BARCHARTS2"
290 RUN "DSK1.DCARDFILE"
300 RUN "DSK1.CALENDAR"
310 RUN "DSK1.CATALOG"
320 CALL CLEAR
330 END
340 SUB MENU(TITLE$,CHOICE$(),N,FUNC)
350 C=(28-LEN(TITLE$))/2
360 DISPLAY AT(1,C)ERASE ALL:TITLE$
370 DISPLAY AT(4,1):"DO YOU WANT:"
380 R=6 :: FOR I=1 TO N
390 DISPLAY AT(R,1):CHOICE$(I):: R=R+2
400 NEXT I
410 R=R+1 :: DISPLAY AT(R,1)BEEP:"TYPE YOUR SELECT
    ION ->"
420 CALL KEY(0,R2,S):: IF S<>1 THEN 420
430 IF R2=13 THEN FUNC=0 :: GOTO 460
440 IF R2<49 OR R2>48+N THEN 420
450 FUNC=R2-48 :: DISPLAY AT(R,24)SIZE(1):CHR$(R2)
460 SUBEND
```

## Program 8-3. System Catalog

```
100 REM SYSTEM CATALOG
110 ON BREAK NEXT
120 REM DEFINE VARIABLES
130 FILENAME$,DRIVE$,DSKNAME$,ACTION$,DODELETE$=""
140 FILETYPE,SECTORSUSED,RECLEN,CHARSUSED,LINENO,D
    OCCOUNT,R,S,DISKSECTORS=0
150 IMAGE "### ########## ### #####  #"
160 CALL KEY(0,R,S)
```

```
170 !@P-
180 REM GET DISK NUMBER AND DELETE OPTION
190 DISPLAY AT(11,1)ERASE ALL:"Do you want to sele
    ctively  delete files (y or n)? n"
200 ACCEPT AT(12,24)SIZE(-1)VALIDATE("yn")BEEP:DOD
    ELETE$
210 IF DODELETE$="" THEN 200
220 DISPLAY AT(15,1):"Which drive (1-3)?  1"
230 ACCEPT AT(15,21)SIZE(-1)VALIDATE("123")BEEP:DR
    IVE$
240 IF DRIVE$="" THEN 230
250 REM OPEN CATALOG AND GET HEADER
260 OPEN #1:"DSK"&DRIVE$&"."&"",INPUT ,RELATIVE,IN
    TERNAL
270 INPUT #1:DSKNAME$,FILETYPE,DISKSECTORS,SECTORS
    USED
280 REM BUILD DISPLAY
290 DISPLAY AT(1,1)ERASE ALL:"DATA FILES FOR ";DSK
    NAME$
300 DISPLAY AT(3,1):" #  File Name Sectr Char DEL"
310 DISPLAY AT(22,1):SECTORSUSED*256;" chars free
    on disk"
320 FOR LINENO=1 TO 15
330 INPUT #1:FILENAME$,FILETYPE,SECTORSUSED,RECLEN
340 IF LEN(FILENAME$)=0 THEN 450
350 IF FILETYPE=5 OR FILENAME$="DCARDFILE" THEN 33
    0 !SKIP OVER PROGRAM FILES
360 DOCCOUNT=DOCCOUNT+1
370 DISPLAY AT(4+LINENO,1):USING 150:DOCCOUNT,FILE
    NAME$,SECTORSUSED,SECTORSUSED*256,"n"
380 IF DODELETE$="n" THEN 420
390 ACCEPT AT(4+LINENO,27)SIZE(-1)VALIDATE("yn")BE
    EP:ACTION$
400 IF ACTION$="" THEN 390
410 IF ACTION$="y" THEN DELETE "DSK"&DRIVE$&"."&FI
    LENAME$
420 NEXT LINENO
430 DISPLAY AT(24,1):"Press any key for more."
440 CALL KEY(0,R,S):: IF S<>1 THEN 440 ELSE 320
450 FOR LINENO=LINENO TO 15
460 DISPLAY AT(4+LINENO,1):" "
470 NEXT LINENO
480 DISPLAY AT(24,1):"Press any key to quit."
490 CALL KEY(0,R,S):: IF S<>1 THEN 490
500 REM END CATALOG LIST
510 DISPLAY AT(14,1)ERASE ALL:"Just a moment, plea
    se."
520 RUN "DSK1.SYSTEMMENU"
530 END
```

## Program 8-4. Error Recovery

```
10 ON ERROR 20
15 GOTO 100
20 DISPLAY AT(4,1)ERASE ALL:"AN ERROR HAS OCCURRED
   "
25 CALL ERR(CODE,TYPE,SEVERITY,LINENUM)
30 DISPLAY AT(6,1):"CODE"
35 DISPLAY AT(6,14)BEEP:CODE
40 DISPLAY AT(8,1):"TYPE"
45 DISPLAY AT(8,14)BEEP:TYPE
50 DISPLAY AT(10,1):"SEVERITY"
55 DISPLAY AT(10,14)BEEP:SEVERITY
60 DISPLAY AT(12,1):"LINE/FILE #"
65 DISPLAY AT(12,14)BEEP:LINENUM
70 DISPLAY AT(17,1)BEEP:"PRESS ANY KEY TO CONTINUE
   "
75 CALL KEY(0,R,S):: IF S<>1 THEN 75
80 RETURN 10
```

# Index

*COMPUTE!'s Guide to Extended BASIC Home Applications on the TI-99/4A* explains how you can use many TI Extended BASIC programming techniques to create useful programs for your home and office. Whether you're an advanced programmer, or just starting to use your computer, you'll find clear, easy-to-understand explanations of how to get more from your TI-99/4A. You'll find this book full of powerful home application programs, ready to type in and run.

Here's a sample of what's inside:

- A complete electronic spreadsheet for both disk and tape
- How to get a program to load automatically from disk
- An electronic filing system
- An explanation of the use of data files
- A string sort subprogram
- A program to create your own bar charts
- How to recover from input errors
- A program to keep track of your family's appointments
- And many other techniques and subroutines you can easily add to your own programs

ISBN 0-942386-41-8