A BYTE BOOK

>>$9.95

COMMODORE 64

PROGRAM BUILDING BLOCKS

Ewin and Shirley Gaby

COMMODORE 64

PROGRAM BUILDING BLOCKS

COMMODORE 64

PROGRAM BUILDING BLOCKS

Ewin and Shirley Gaby

McGraw-Hill Book Company

New York St. Louis San Francisco
Auckland Bogotá Guatemala Hamburg
Johannesburg Lisbon London Madrid
Mexico Montreal New Delhi Panama
Paris San Juan São Paulo Singapore
Sydney Tokyo Toronto

COMMODORE 64 PROGRAM BUILDING BLOCKS

A BYTE BOOK

1 2 3 4 5 6 7 8 9 SEMSEM 8 7 6 5 4

ISBN 0-07-022667-9

LIBRARY OF CONGRESS CATALOGING IN PUBLICATIONS DATA

Gaby, Ewin.
Commodore 64 program building blocks.

(A Byte book)
Includes index.
1. Commodore 64 (Computer)—Programming. 2. Subroutines (Computer programs) I. Gaby, Shirley.
II. Title. III. Title: Commodore sixty four program building blocks. IV. Series: Byte books.
QA76.8.C64G3 1985 001.64'2 84-17151
ISBN 0-07-022667-9

The editor for this book was Jeff McCartney; and the editing supervisor was Aliza Greenblatt. Production by Joyce Kaplan. Design by Patrice Fodero.

CONTENTS

	Preface	vii
CHAPTER 1	Introduction	1
CHAPTER 2	How to Use This Book	7
CHAPTER 3	Some Basic Modules	25
CHAPTER 4	Home and Personal	57
CHAPTER 5	Business	95
CHAPTER 6	More Business	135
CHAPTER 7	Statistics and the Classroom	173
CHAPTER 8	Sprites and Graphics	209
	Index	237

PREFACE

We want our computers to do work for us. We want to think, "Why not let the computer do this?", and then let the computer do it. We don't want to spend hours programming the beast in order to get it to do a little useful work.

Still, much of the work we want our computer to do can't be found in the available software. Does this mean we must learn all of the intricacies of the art of *programming* just to get a little work out of our computer? Perhaps not.

This book can help bridge the gap between available software and self-programmed software. In it you'll find another approach to programming. Here are program segments, called modular programs. Each modular program performs a specific function. Each module is like a building block, which can be used, along with other blocks, to build a complete program. By stringing together the functions you want, you can assemble a program to fit your needs.

Each chapter in this book deals with a different set of program subjects (such as home and business, or sprites and graphics, etc.). In each chapter, major programs are

presented in which the modules are used. Thus, you not only have a listing of a group of modules, but you're also given a number of useful programs which incorporate the modules.

The programs include such things as a card file (which can be converted into a grocery list or a tickler file) a complete monthly payroll program, a personnel effectiveness listing which makes a perfect grade book for teachers, a program which designs and moves sprites, a checkbook program which does everything except write the check for you, and many other useful and enjoyable programs for home, office, classroom, or just for your enjoyment. As you assemble the programs in the book, you can learn how each module works, and certainly will think of other programs in which the modules can be used.

The modules themselves are varied in character. Here you will find modules for menus, and for loading arrays from a disk or tape; modules which will save a program and then run it, which can move a cursor around the screen using a joystick or keyboard, which can search files to find the one that matches your input. There are over a hundred modules presented in this book, which can be used together to build useful and personal programs.

With all of these modules and programs, there is ample instruction. The first two chapters are devoted to discussing the conventions used and how to best begin to design your own programs. Subsequently, each chapter provides a text which describes each module in the chapter. The text is then followed by the listing of the modules for that chapter.

So, this is a book of building blocks for programs, and more! It's also a book of unique and useful programs which you can begin to use immediately. We hope it will help you to design programs which will fit your needs.

Chapter ONE

INTRODUCTION

The Commodore 64 is an exceptional computer. It has many of the attributes of a complex business or scientific computer, and yet can be afforded and understood by almost anyone. Because it is a complex computer while being simple enough to be understood, the Commodore 64 is an ideal tool for the person who wants to use a computer without learning "all about computers." It should be possible for anyone with a minimum of programming skills to use the Commodore. As users, we can develop our own simple programs and purchase software when the programming is beyond our skills.

Still, where do we draw the line between designing programs for ourselves to meet our specific needs and purchasing software which works well but may not fit our needs precisely? What "simple" programs can we design to fit our needs? What programs should we buy? What purchased programs can we modify (if possible) to meet our needs more specifically?

The answers to these questions depend on how complex our needs are, how much money we are willing to spend on software, and how much we know (or want to learn) about programming the Commodore 64. Although this book cannot answer these questions for you, there are a number of things that it can do.

WHAT THIS BOOK CAN DO FOR YOU

This book contains a number of useful subprograms, which we will call *modules*. These modules are designed to be assembled into larger, specific, useful programs for home, classroom, and business. They also can be used as parts of other programs, which we can design to meet our own specific needs.

That is really the purpose of this book—to provide a wide variety of modular programs which we can modify and connect together to make programs that fit our needs. This book is an inventory of parts from which we can assemble programs, like building with Tinkertoy pieces.

Chapter 2 discusses the specific ways in which this book can best be used. It addresses the logic behind the problem-solving process used to design programs, and how to use this process more effectively. Chapter 2 also looks at the difference between the careful, analytical development of a program and the less careful, but often faster trial-and-error approach.

Also in Chap. 2, a method is given for assembling modules into larger programs. This is done by using a device to fool the computer into chain-loading the modules. The first few programs are assembled as examples of how to use the modules from the book, and of the housekeeping work required to allow the modules to work properly together.

CONVENTIONS

The Commodore 64 has so many PRINT formatting functions available from the keyboard that it is necessary to provide a listing showing how we will indicate those functions in this book. To keep the function symbolism as simple as possible and yet descriptive and easily understood, we have used the notation shown in Table 1-1. A repeated function is indicated as, for example, [3x dwn], which calls for using the cursor down three times. Graphic symbols are indicated by shifted letters, such as [sft/X] or [cmd/P]. Color designation is indicated either by a control-key-shifted number such as [ctrl/6] or by a Commodore-key-shifted number such as [cmd/3]. The colors cor-

Table 1-1 **Notation**

THIS SYMBOL	MEANS
[home]	CLR/HOME (unshifted)
[clear]	Shifted CLR/HOME
[rvs]	Reverse on
[off]	Reverse off
[up]	Cursor up (up arrow)
[dwn]	Cursor down (down arrow)
[lft]	Cursor left (left arrow)
[rht]	Cursor right (right arrow)
[spc]	Space
[cmd/..]	Commodore key
[sft/..]	Shift key
[ctrl/..]	Control key (CTRL)
[2x ..]	Repeat two times (number of times repeated)

responding to the shifted numbers are given in the Commodore 64 User's Manual.

These symbols will always appear within quotation marks in an INPUT or PRINT statement. Each set of similar symbols will be separated from other symbols by brackets. For example, the line

```
20 PRINT"[clear][8x dwn][5x rht][ctrl/2][rev]HOT
   STUFF ![off][ctrl/7]"
```

will clear the screen, step down eight lines and over five spaces, print the words "HOT STUFF !" in reversed white (ctrl/2), and then turn off the reverse and print any following copy in blue [ctrl/7].

To enter this into the computer as a program line, you must follow the typed line as it is shown. After entering the line number, PRINT, and the quote marks, enter Shifted CLR/HOME [clear], enter the down cursor eight times [8x dwn], enter the right cursor five times [5x rht], enter the number 2 while holding down the CTRL key [ctrl/2], and enter the number 9 (rvs on) while holding down the CTRL key [rev]. All of this input is recorded by the computer as a string of symbols in the PRINT line. We can now continue to enter the rest of the line using the same techniques.

USING THE MODULES IN THIS BOOK

Most of the modules in this book are designed to be used in a specific program. The program demonstrates how the module works in conjunction with other modules. At the same time, the program itself is operational. This means that if it fits your needs, it can be used just as it is pre-

sented, or it can be modified to fit your needs more closely.

Where modules are to be used in the program given, the line numbers can usually be entered just as they are shown in the book. In the event that you are using a module in a program of your own design, it will be necessary for you to renumber the lines to fit the needs of your program. In such a case, care must be taken to be certain that all GOTOs and GOSUBs are directed to the correct line numbers.

Although Chap. 2 will discuss the methods for assembling modules into programs, it should be pointed out that almost any module will require some adjustment or "housekeeping" in order to be compatible with the other elements of your program. This housekeeping can be kept to a minimum by using conventions similar to those used in the modules of this book. After you have selected the modules that you believe can be assembled into the program that you are designing, look at the conventions used in the modules. Try to use the same conventions throughout your program, such as using N as the counter of a FOR/NEXT loop, so that the loop reads "FOR N = 1 TO 100," Use 'NEXT N'. Use the same input variable for all the nondata inputs. (The modules use Z$ for this type of input variable.)

CONCLUSION

You will find that the modules in this book are often unique and can easily be used to develop your own unique programs. At the same time, most of the modules are designed to fit into a specific program which can be used just as they are shown in the book.

This variety of program elements and program design gives you the flexibility to assemble your own programs without the need to be an expert programmer. The skills that you will need in order to assemble the modules into programs are minimal and will be supplemented by the instructions given in Chaps. 2 and 3.

These modules provide you with many complex subprograms which could take hours to design. If you were to take the time to enter all the modules onto a disk or tape, you could assemble your programs simply by chain-loading into the computer the modules you want. You may want to save each module as you use it so that you build up a library of modules from which to select. Eventually, you will be able to assemble programs using only the modules from the disk or tape.

Chapter TWO

HOW TO USE THIS BOOK

Most computer programs operate in a very simple manner. First, we input data or instructions. Then the computer uses this input to operate. Finally, the computer displays or stores the results of the operation. Computer programming is simple: input—operate—output. Consider

```
10 INPUT X          (input)
20 LET Y=X-30       (operate)
30 PRINT Y          (output)
```

The complications arise when we want to enter large amounts of data, manipulate the data in a number of different ways, and arrange the output so that the results are easily comprehended. Still, each step of even the most complicated computer program is simple. The number of steps and the interaction of the steps is what makes a program appear complicated.

Our job is to learn how to break down complicated requirements into simple steps. Once we have done this, we can simply assemble our program from the modular programs in this book, like placing beads on a string.

ANALYZING THE PROBLEM

Learning to describe our program requirements in simple steps is just like solving a problem. Describing these requirements requires the same problem-solving techniques that we use in our everyday lives. Some of us may not be aware that we use such techniques, but it is impossible to survive without them. For example, consider the following conversation:

"What time is your appointment?"
"10 o'clock." (input data)
"How long will it take to get there?"
"About 30 minutes." (input data)

Now you proceed to solve the problem. You think, 10 o'clock, minus 30 minutes travel time, equals 9:30. This is the *operation* part of the process.

"If you want to be on time, it will probably be a good idea to leave no later than 9:30." (output)

In this simple example, the problem is: "When must I leave in order to arrive on time for my appointment?" The first step in solving this problem is to determine the requirements (or the objectives) of the problem.

What is required?

To know the time to leave here in order to arrive there at 10 o'clock.

Any other requirements?

No.

The next step is to decide on some process or method for determining the time that you must leave in order to arrive at your destination on time. Doing this is almost always a trial-and-error determination. Experience, research, and consultation with others can often help to provide a number of possible solutions to the problem, but you must decide on the most likely solution and try it. This is the human part of problem solving. In complicated problems we will usually be able to think of a number of possible solutions. Then the choice must be made as to the most efficient method to use to solve the problem.

In our example, experience tells us that in order to be punctual, one must leave prior to the appointment time. Experience also tells us that if we are going to be on time, we must leave early enough to get from where we are to where we are going before the appointment time.

Another way to write this process might be

LEAVE TIME=APPOINTMENT TIME
– TRAVEL TIME

If we know the appointment time and the travel time, we can solve the problem of when we must leave. The *input* required is (1) appointment time and (2) travel time. The *operation* is to subtract travel time from appointment time. The *output* is the answer to this problem: Leave time.

This exercise may be trivial but the process used to make this decision is the same process as that used when solving any problem, from "Where did I put my shoes?" to "How does gravity affect time?" The process used to solve any problem can be described as follows:

1. Define the problem (what, exactly, do you need or want to know?).
2. Determine what appears to be the best method to use to solve the problem (using experience, research, help, and trial and error).
3. Obtain any input required for solving the problem.
4. Try using the method that you have selected.

Two more steps that we will want to add to this process are:

5. Test the results to see if the method chosen works well enough, or needs to be changed.
6. If the results were not adequate, go back to step 2, make adjustments, and try again.

You must test the results to see if they are adequate. In our example the test and the results might be:

"Did you get to your appointment on time?"
"Yes, what's for supper?"

The chosen method must have worked!

APPLYING PROBLEM SOLVING TO YOUR COMMODORE

How does this type of problem solving help us to program our Commodore 64? It helps because it is the same process that we will use to program our Commodore. The easiest way for us to use the modular programs in this book is to use a systematic problem-solving method. Such a method will help us to identify which modules we can use to meet our objectives.

USING THIS BOOK

This book is a compilation of modular programs which can be linked together to form other, larger, customized programs—programs that will solve your problems in the way you want them solved.

Consider how powerful these customized approaches can be. No one knows your needs better than you do. If from a collection of modules, you can assemble a program to fit your needs exactly, then you have designed your own program without having to write the individual elements of that program.

A large portion of this book consists of program modules designed for specific uses, such as programs for use around the home, programs for business, graphic programs, and more. All of these modules require the same basic format. They all require some form of input or interaction with the operator. When the computer has done its work, it needs some way to communicate the results to us in a meaningful form. The next two chapters describe some input and output modules.

STRINGING THE MODULES TOGETHER

The key to easy assembly of customized programs lies in how the modules can be entered into the program. We suggest that when you choose a module for use, you enter the module into your computer and save the module on tape or disk. In this way you will build a library of saved program modules.

When you are ready to assemble a program from modules, begin by entering the following two lines:

```
0 CLR:POKE 43,((-26627-FRE(8))/256-
  INT((-26627-FRE(8))/256))*256+1
1 POKE 44,(INT((-26627- FRE(8))/256))+8
```

This program allows us to move the bottom limit of memory up to cover the last line of BASIC. If, after running the program, we ask the computer to LIST, it will respond with nothing to list. However, any program that was present before running this program is still there—it is just below what the computer now sees as the bottom limit of the memory. Thus, if we load a module, the module loaded previously is not lost. If we load the module with the lowest line numbers first (and have no conflicting line numbers), we can string the modules together.

We string the modules together by first entering our little two-line loader program, and RUN 0 (this moves the memory up to cover line 1). Second, we LOAD the first module (let's say that this module has line numbers from 100 to 160). When this load is complete, we enter the following, using the immediate mode:

POKE 43,1:POKE 44,8

These POKEs reset the memory to normal and bring lines 0 and 1 back into the computer's memory. Now, we have lines 0 through 160 in the computer's memory.

Next, we change line 0's number so that it becomes line 170 and change line 1 to line 171. When this is done, we delete lines 0 and 1. What we have done is to move our two-line loader program to the end of the program listing.

To add the next module, we proceed as follows:

1. RUN 170 (change memory)
2. LOAD new module (load new)
3. POKE 43,1:POKE 44,8 (reset memory)

If we have more modules to add, we move the two-line loader to the end of the latest module, delete it from its previous position at lines 170 and 171, and continue the RUN–LOAD–POKE sequence.

Of course, we must always add modules whose line numbers do not conflict with those of modules already loaded. Also, the new module must always have higher line numbers than those already loaded. If this is not the case, then after loading the new module (step 2) it is possible to edit the line numbers of this new module to meet our requirements. After correcting the line numbers, we can reset memory with the POKE line (step 3).

STEP BY STEP

Using the modules in this book will require that we do the "housekeeping" or cleaning up of the loose ends needed to make the programs fit together. Let's use an example to

illustrate both the housekeeping needed and how a program might be assembled.

Suppose that we have a classroom of students. Let's say that we have 30 students, and that we want to keep a record of the students' grades on five tests given during the term. How can this book help with such a problem?

Using the problem-solving steps that we have already discussed, we can define the problem as wanting to store five test grades of 30 students. While we are storing the test grades, we might as well determine each student's average test score for the term.

The best method to use for solving this problem would seem to be to store the scores in a two-dimensional array. One dimension will designate the test number (1, 2, 3, 4, or 5), and the other dimension will designate the student number (1 through 30). If we use an array variable A(5,30) and store all the test scores in it, we can find each test score (T) for each student (S) by printing A(T,S). By this we mean that if we want the score on test 2 (T = 2) for student 18 (S = 18), we just print A(2,18) and we get the score of student 18 on test 2.

Now, to assemble the program. First, we need to set up our variables so that the computer knows the size that we will need. Our first line will be

```
5 DIM A(5,30)
```

Next, we need to find an input module in Chap. 3 which will input data into a two-dimensional array. Module 3/19A should do. We need only to change the line numbers to fit our program.

The housekeeping must now be done. The array variable dimensioned in line 5 must be the same variable as that used in the input program. Is it? Well, both of them are two-dimensional and both are the variable A. The variable

is the same, but we must insert the dimension limits (5 and 30) into the program variable on line 5. We must use B and C as the variables for these limits. We add these limits to line 5 and add line 5 to Module 3/19A.

Now our program reads as follows:

```
 5 DIM A(5,30):B=5:C=30
10 PRINT"[clear]X=1"
20 FOR X=1 TO B:FOR Y=1 to C
30 PRINT TAB(5) X; "/";Y;"="; : INPUT A(X,Y)
40 NEXT Y:PRINT"X=" X+1
45 INPUT "ENTER * TO STOP, SPACE TO
   CONTINUE";Z$:
   IF Z$="*" THEN 60
50 NEXT X
```

Here the program simply goes through each test number asking us to input a score for each student. It does this by printing "X/Y=?", where X is the test number and Y is the student number. If we want to customize this program still further, we can change line 30 to read

```
30 PRINT "TEST #"X", STUDENT #"Y;:INPUT
   A(X,Y)
```

Notice that we have added line 45, which allows us to stop the input at any point.

Now that we have the data in an array, we need to save it. Module 3/7 will do that for us if we choose the subroutine that starts at line 1200. To do this, we change line 1070 of Module 3/7 to read

```
1070 GOSUB 1200
```

When entering Module 3/7, we need only to enter lines 1010 through 1080 and lines 1200 through 1230. In this module we must tell the computer which type of storage device we are using. We can do this by setting the variable DR equal to either 1 or 8.

If we are going to save the data on a Datassette recorder, then DR = 1. If we are using a disk drive, then DR = 8. This variable, DR, also can be added to line 5. Thus, for a disk drive,

```
5 DIM A(5,30):B=5:C=30:DR=8
```

Because line 40 of our program ends with GOTO 60, we should add our SAVE program, starting at line 60. Again, we must do our housekeeping. Do the variables used so far in the program match those in the module we are about to add?

In both modules the array variable is A(B,C), and B and C have already been set to correspond to the size of the two dimensions of A. The storage device has to be identified by the variable DR. It appears that we are ready. Thus

```
 60 INPUT "ENTER THE ARRAY NAME";FL$
 70 OPEN 2,DR,2,"0:"+FL$+",S,W"
 80 FOR X=1 TO B: FOR Y=1 TO C
 90 PRINT #2, A(X,Y)
100 NEXT Y:NEXT X
110 CLOSE 2:PRINT "DATA SAVED"
```

Entering the array name (line 60) allows us to give a different name to each set of data. We could name our saved test scores "class 1," "class 2," and so on. That is all there is to

this part of the program. We have identified our variables on line 5, input our students' scores using Module 3/19A (lines 10 through 50), and saved our data using Module 3/7 (lines 60 through 110). We can stop here and call this our input program. We can then write a second program to retrieve the data from our storage device and calculate the averages.

To do this, add line 115 to our existing program:

```
115 STOP
```

Now save the program on the storage device by typing

```
SAVE"INPUT PROGRAM"      (For Datassette)
```

or

```
SAVE"INPUT PROGRAM",8      (for disk)
```

With this program saved, we can erase it from the computer by entering NEW. Now, we are ready to start the output program. We will need the same variables, so we will use a similar line 5, but we can change the DIM statement to indicate the use of B and C.

```
5 B=5:C=30: DIM A(B,C): DR=8
```

To get the data out of storage, use Module 3/11 from Chap. 3. Let's start it on line 200.

```
200 REM OUTPUT PROGRAM
210 DR=8
220 INPUT"ENTER ARRAY NAME";FL$
```

```
230 OPEN 2,DR,2,"0:"+FL$+",S,R"
240 FOR X=1 TO B:FOR Y=1 TO C
250 INPUT#2,A(X,Y)
270 NEXT Y:NEXT X
280 CLOSE 2
```

Again, this simple module asks us for the name of the file we want, opens the file, reads the contents out into the array A(X,Y), and closes the file. We have retrieved our data from storage and have it in an array. Now, what are we going to do with it?

Module 3/23 from Chap. 3 will print out our array with line and column totals. This module is designed for use with a video screen. If we are using a printer, additional commands are needed. For the video version, the complete module reads as follows:

```
260 FOR Y=1 TO C: PRINT Y;"[spc]";:
    FOR X=1 TO B
265 PRINT TAB(X*34/(B+1));A(X,Y);
270 SUM(X)=SUM(X)+A(X,Y)
    ADD=ADD+A(X,Y):NEXT X
275 ADD=INT(10*ADD/B+0.5)/10
280 PRINT TAB(33) ADD:ADD=0
285 IF Y>23 THEN INPUT "ENTER TO SCROLL"
    ;Z$:PRINT"[up][17x spc][up]"
290 NEXT Y:PRINT"COLUMN TOTAL"
295 FOR X=1 TO B:PRINT TAB(X*
    34/(B+1)-2)SUM(X);:NEXT X
```

The line totals provide us with the sum of the scores for each student, but we wanted the student's average score.

The average (mean) score would be the sum of the student's scores divided by the number of tests. We already have the sum. ADD (which is accumulated in line 260 and printed in line 270) is the sum of the scores, and we have the number of tests—B. Thus the mean is simply ADD/B.

We need to limit this test average to a reasonable number of significant figures. That is what line 275 does:

```
275 ADD=INT(10*ADD/B+0.5)/10
```

Stating the average in this way will limit it to one decimal place. The +0.5 will cause the decimal to round off to the nearest tenth.

We have set this calculation equal to ADD, so it is simply inserted between lines 270 and 280. Line 270 sums the student's scores in the variable ADD. Line 275 changes ADD from a sum to an average and line 280 prints ADD.

If we also want to list the average score of the class on each test, we must use the column totals, divided by the number of students. First, we will need to print a title for the column averages as follows:

```
300 PRINT:PRINT"TEST AVERAGE"
```

Now, calculate each column average and print it. To get each column average, we take the column sum, SUM(X), and divide it by the number of students, C. We want to limit the answer to one decimal place and round to the nearest tenth. To print our average, we can use the same tab as that used to print the column totals (see line 295). Thus we have

```
310 FOR X=1 TO B:PRINT TAB(X*34/(B+1)-2)
    INT(10*SUM(X)/C+0.5)/10;:NEXT X
```

That is the entire program. All we did was to combine Module 3/11 with Module 3/23, provide initial data in line 5, and then customized the program by adding lines 300 and 310. Not too hard. If we do not want to do this extra work, we can simply plug in an averaging module from Chap. 5.

Finally, if we like, we can combine our two programs (the input program and the output program) into a single program. To combine the programs, add the following line to the input program:

```
7 PRINT "[clear]":INPUT
  "[7x dwn][3x rht]DATA INPUT
  OR OUTPUT(I/[rvs]O[off])";Z$:
  IF Z$<>"I" THEN 200
```

This line will let us choose between inputting new data or retrieving stored data. We can now change line 115 to read

```
115 INPUT"CONTINUE([rvs] Y [off] /N)";Z$:
    IF Z$="N" THEN STOP
120 GOTO 210
```

Now we have a program that will input test scores and store them, retrieve stored test scores and print them, average each student's scores, and give the class average on each test.

Using this basic program, we can add statistical modules and graphing outputs, if we like, to provide an analysis of the test results and different forms of output. These new modules can be added in the same way that the test score program was assembled, by doing the housekeeping to keep the variables correct.

If we add the statistical and graphing modules to this test score program, we will probably also want to add a way to select the part of the program that we want. The easy way to do that is to put a menu at the beginning of the program. See Menu Modules 3/14 and 3/15 in Chap. 3.

SUBROUTINES

Most of the modules in this book can be used as subroutines by adding RETURN as a last line. Doing this is an easy way to assemble a program. Using this method, your main program will become a series of GOSUBs separated by your housekeeping work. For example, the main program for our test score program might look like this:

```
 10 B=5:C=30:D=8:DIM A(B,C)
 20 PRINT"[clear]":INPUT"[4x dwn]
    [2x rht]DATA INPUT OR OUTPUT (I/[rvs]O
    [off])";Z$:IF Z$<>"I" THEN 80
 30 GOSUB 1000:REM to input program
 40 GOSUB 1500:REM to save program
 50 INPUT"CONTINUE ([rvs]Y
    [off]/N)";Z$:IF Z$="N" THEN STOP
 60 GOSUB 2010:REM to print program
 70 GOTO 90
 80 GOSUB 2000:REM to print program
 90 PRINT"CLASS AVERAGES":FOR X=1 TO B
100 PRINT TAB(X*34/(B+1)-2)
```

```
    INT(10*SUM(X)/C+0.5)/10;
110 TTL=TTL+INT(10*SUM(X)/C+0.5)/10:NEXT X
120 PRINT:PRINT TAB(5)"CLASS
    AVERAGE="TTL/B
130 STOP
```

In this example you will notice that lines 30, 40, 60, and 80 go to subroutines. These subroutines are constructed from the appropriate modules. Each of the modules being used as subroutines must be listed using the appropriate line numbers and ending with the command RETURN. The RETURN command must be placed in the module so that when the operation of the subroutine is completed, the operation will RETURN to the main program. This is often accomplished simply by making RETURN the last line of the subroutine. In some cases, however, there will be a line, other than the last line, which will determine when the subroutine is finished. That line may look something like this:

```
2015 IF D$(7)="0" THEN GOTO 2100
```

Here the value of D$(7) determines when the module program is complete and causes it to jump out of the module to line 2100. If you are using this module as a subroutine, simply replace GOTO 2100 with RETURN. For additional information about subroutines and how they work, see your User's Manual [or see Chap. 2 of the book *GOSUBS* by Gaby and Gaby (New York: McGraw-Hill, 1984)].

CONCLUSION

There are basically two ways to use the modular program segments given in this book. They can be used (in a very organized manner) by diagramming your main program and plugging in the modular segments where appropriate. Alternatively, a program can be assembled by using a step-by-step trial-and-error method, by selecting the module that seems to do what is needed, trying it, and then moving on to the next step. Either method will work, and each has advantages and disadvantages.

In assembling modules into programs, care must be taken to be certain that the variables and constants passed from module to program and from program to module are compatible and do not cause errors or confuse the computer. The program areas, which must be supplied in order to link the modules together, will be the keys to the program's clarity. These linking segments should be kept as simple and straightforward as possible. This housekeeping, which you must do, is not difficult but does require your close attention.

Above all, do not be afraid to try your ideas. You will not damage the computer, and except for the time lost, a "blown" program can be a positive learning experience. If you save your programs before you try them, even the loss of time can be minimized. Soon you will be amazed at how easily you are assembling exceptionally useful programs from the modules provided in this book. That is what this book is designed to help you do—assemble useful programs easily.

Chapter THREE

SOME BASIC MODULES

Before we proceed to the modules designed for building specific programs, we should discuss some generally useful core modules. There are a number of common operations that we will need in almost any program we write. The most obvious of these common operations are the saving and loading of programs and data.

Any program that you assemble will need to be saved on disk or tape if you wish to use it again. Once saved, the program will later need to be loaded into memory in order to use it. Data, too, must be saved and retrieved if it is to be preserved and reused. This chapter provides a number of different save and load modules from which you can choose the ones that best fit your needs.

This chapter also covers modules that can initialize your programs, menu modules with which to select optional functions, and modules that can change to fit your needs.

SAVE AND LOAD

The Commodore floppy disk unit is a useful, yet inexpensive storage device. The save and load modules in this section have been written to use this disk unit but can easily be converted for use with the Commodore Datassette. The modules in this section use the number 8 to identify the storage device as a disk. If you are using the Datassette tape, change the 8 to a 1 and the module will work for your tape. This conversion (from 8 to 1) is noted in each module so that you will not forget to make the change.

The easiest way to load a program is simply to type

```
LOAD " [program name] ",8
```

Modules 3/1, 3/2, and 3/3 can be used within one program to load a second program. Of course, loading the second program erases the first program from memory, but using these modules (3/1 through 3/3) allows us to "chain" programs.

Module 3/1 asks for a program name and LOADs that program into the computer. Module 3/2 does the same thing and in addition RUNs the program. Module 3/3 does not ask for a program name but LOADs and RUNs the program named by PR$.

As simple as these load programs seem, they can be very useful. For instance, a menu of available programs can be listed using Menu Module 3/14. The program name chosen from the menu can be loaded using Module 3/3. The only change required to combine Menu Module

3/14 and Load Module 3/3 is to make the name selected on line 180 of the menu module the same name as that used in the load module. To do this, change Z$ to PR$ in the menu module.

Although Module 3/3 will load any program named by PR$, it can also be used to load a program whose name never changes. You might use such a program to load a procedure program automatically if an inventory falls below a certain level. This same program could be used to load a want list program if the checkbook balance exceeds some specified amount. To use Module 3/3 in this way, simply add to it line 105 (given below) with the program name inserted as shown. When your program goes to this module, it will LOAD and RUN the new program which is named in line 105.

```
105 PR$=" [program name] "
```

SAVE modules are of limited use, because it is seldom that a program will need to save itself. Module 3/4 can, however, be placed at the end of a program that you are writing. Thus, by GOTO 9000, you can be certain that the program in progress is saved correctly. Module 3/5 can be used in the same manner as Module 3/4, but allows you to input different names for different versions of a program in progress.

Modules 3/4 and 3/5 both include verification of the program saved. They may be most useful for a program in which you are still making changes or in a program that you want to copy many times. Such a program might be an index program placed on each disk which is constantly changing as new items are added to the index.

SEQUENTIAL DATA STORAGE

Just as saving and loading programs is important, so is saving and retrieving data. The most common and easiest form of data storage is the sequential data file. This is the only type of data file that can be stored on a Datassette.

To store data in a sequential file, we must OPEN the file (using the file name, identifying the file as sequential, and indicating that we plan to write into the file) and write data into the file using PRINT#. When we have completed writing in the data file, we CLOSE the file.

To read the information stored in a sequential file, we must go through a similar set of steps. We must OPEN the file (using the file name, identifying the file as sequential, and indicating that we expect to read the file), read the data from the file (using either INPUT# or GET#), and when we are finished, CLOSE the file.

Once a sequential file is established, changes or additions can be made only by reading out the entire contents of the file, making any desired changes, and reentering the entire changed contents into the file. The sequential file is like a file folder with all the information stapled into the folder. We can find the folder by the file name, and we can read the data in the order in which it is placed (stapled) in the file, but we cannot change or add to the file without taking out the staple and restapling the changed data into the file folder.

There are two forms of statements to OPEN a file into which we expect to write data. One form is used when we are opening a new file and do not expect (or want) any other file, having the same name, to be present. We will call this form the *initial* form. The second form is used when there is an existing file and we wish to change the contents of that file. We will call this form the *replace* form.

Both the replace form and the initial form can be used to open a new file. If there is an existing file with the same name as the new file, the initial form will warn you of this and will not open the new file. Under the same conditions, the replace file will not warn you but will simply delete the existing file and write the new file in its place. You should be aware of this difference.

Module 3/6 is a complete program for putting data into a sequential file. The module will ask for a file name and will attempt to open that file with the initial open form. If the file already exists, the program asks if it should overwrite (or replace) the existing file. If the answer is yes, the file is opened using the replace form of open, and the module prepares to receive data.

Lines 1070 and 1075 provide the method for data input, using the string variable X$. Line 1070 can be changed to GOSUB to an input subroutine by substituting GOSUB (line number) for INPUT X$. You must supply the line number of the subroutine. This input subroutine can be chosen from the input modules discussed later in this chapter. If the subroutine begins at line 400, line 1070 would be changed to read

```
1070 X$="":GOSUB 400:IF X$=""GOTO 1080
```

Upon returning from this subroutine or upon completing the input, the module writes the data into the file and goes to line 1070 for more data. If the input is only a "return," the program jumps to line 1080 to close the file. You can replace the STOP in line 1090 with GOTO(line number) in order to continue on to other parts of the program.

Module 3/7 is much less complicated than Module 3/6. Module 3/7 does not ask for a file name and does not

check for existing files. (It uses the replace form of the opening statement.) Module 3/7 uses the string variable X$ to input alphanumeric data. It can be modified to GOSUB to an input subroutine. This module is for use in programs, where we are often updating or changing the same file or files. If we are working with a single file, that file name can be used in place of SF$ or we can change line 1010 to read

```
1010 DR=8:SF$=" [file name] "
```

Otherwise, SF$ must be furnished from the program that precedes the use of either of these modules.

Modules 3/8 and 3/9 are like 3/6 and 3/7 except that they do not ask for a file name and do not check for existing files. Both 3/8 and 3/9 open the file named by FL$, using the replace mode to open the file.

RETRIEVING SEQUENTIAL DATA

Reading the information from a sequential file can be done rather easily. We open the file, read the data, and close the file. Because we are not writing in the file, there is no possibility of destroying or losing data stored on the disk or tape.

Module 3/10 asks for a file name, OPENs that file (if available), GOSUBs to a read module, and then GOSUBs to an output module. When the "end of file" marker is read, the module CLOSEs and STOPs. Otherwise, it repeats the read/output sequence.

Module 3/11 reads a two-dimensional array. Like Module 3/10, it asks for a file name and OPENs the file

named. Then Module 3/11 reads in a complete array, A(B,C). It then CLOSEs and STOPs. If desired, a PRINT line can be inserted at line 2060. This line could be

```
2060 PRINT A(X,Y)
```

It is possible that we may not have used all the positions in the array. We may not have had enough data to fill the array at this time. In this case we will not wish to read the portion of the array that does not contain data, and therefore may wish to add line 2065 as follows:

```
2065 IF ST>0 THEN 2080
```

This line will stop the reading process at the "end of file" marker.

Module 3/12 is for alphanumeric data and can be modified for numeric data by changing the X$ to an X. This module asks for a file name, searches for it, and reads from it. When a piece of data is read, the program goes to an output subroutine (line 2050) and upon return from the subroutine looks for more data to read. If it finds the "end of file" marker (st>0), the module CLOSEs the file and STOPs. If desired, the GOSUB to an output subroutine (line 2050) can be replaced by a PRINT:

```
2050 PRINT X$
```

Module 3/13 does the same thing as Module 3/12 but does not ask for a file name. Module 3/13 can be used where the file name is specified somewhere else in the program. It also can be used when the file name is always

the same. In this case SF$ can be replaced by the file name in quotation marks, or line 2010 can be changed to read

```
2010 DR=8:SF$=" [file name] "
```

RANDOM FILES

Although random files are very useful, they can be used only with disk storage and cannot be used with the Datassette. For these reasons we will not include random file modules in this book.

MENU

A menu is an excellent way to branch a program to areas selected by the operator. The menu allows selection of one part of a program from a number of options. One menu can lead to other menus which can offer more finely discriminated choices. In this chapter we have two menu forms. The first (Module 3/14) requires that we change the module to include the names of the items that will appear on the menu. This is done by setting each item name variable (M0$ through MJ$) equal to the item name desired. Thus

```
10 M0$="BANK ACCOUNT":M1$="BUDGET"
11 M3$="SAVINGS":M4$="NET WORTH"
```

This might be your list of items for the menu. The maximum possible number of items is 20 (M9$ is the tenth item name, MA$ is the eleventh, MB$ is the twelfth, etc.).

In our example we have used only four items, so set N = 4 on line 20 and your menu is complete.

The second menu (Module 3/15) is much more sophisticated. It asks us for item names and actually changes itself to include our input into a DATA line. Once done, this changed data line becomes a permanent part of the module. In Module 3/15 you will find a DATA line filled with zeros (line 250). The program will replace these zeros with the menu item names that you input.

The total length of your menu item names and the commas that separate them can be no greater than 67 characters. This is because the DATA line will hold only 67 characters and the "−1." which indicates the end of the data.

As you input your item names, lines 50 and 60 will assemble the new DATA line, place commas between the names, and at the same time keep track of the total number of characters. If you exceed the maximum allowable number of characters, line 50 will inform you of this and your last entry will not be included.

When we have finished entering items, the item names are POKEd into the DATA line, the menu is printed, and the REM in line 10 is automatically changed to GOTO. Now that we have finished with the input part of this module, automatically changing the REM to GOTO in line 10 will cause the computer to go directly to the menu when the program is run.

As written, Module 3/15 can be used only with the line numbers shown, because the module looks for line 10 to change the REM. This limitation can be removed by changing line 170 so that the "10" is replaced by the "lo byte" of your REM line number and the "0" is replaced by the "hi byte" of this line number.

INPUT AND OUTPUT

Almost any program that we assemble will require some form of data or instructions input. Similarly, most programs will require some form of output that will help us make sense of the computer's work. There are so many possible ways in which data can be input and output that we could write a complete book just on these two topics.

In this chapter we have listed a number of typical input and output modules which will handle many of our needs. When required we can modify them to meet our needs more specifically. Additionally, some special-purpose input and output modules will be found in later chapters.

Input

Module 3/16 will accept the input of six digits and will place a decimal point between the fourth and fifth digits. This dollars-and-cents format is used in many programs. Module 3/16A does the same thing, except that the input is stored in a string rather than as a number. Using a string has the advantage of retaining zeros following the decimal (which the numeric input will not). Module 3/16A converts any initial zeros (zeros that precede whole numbers) into spaces.

Module 3/17 asks for the number of items to be in an array and then accepts numeric input for that many items. Module 3/17A does exactly the same thing except that it accepts alphanumeric input and stores it in a string array. Line 240 of either module can be changed to use GET rather than INPUT, if desired, and the length of the string input in Module 3/17A can be limited by changing line 240 and adding lines 244, 245, 246, and 247 as shown:

```
240 GET B$:IF B$=""THEN 240
244 PRINT B$
245 A$(X)=A$(X)+B$
246 IF LEN(A$(X))=L THEN PRINT:GOTO 250
247 GOTO 240
```

The variable L in line 246 can be input from the program or a constant can be substituted. For instance, if we always want the length of A$(X) to be 7, we should substitute the number 7 for the L in line 246.

```
246 IF LEN(A$(X))=7 THEN PRINT:GOTO 250
```

Modules 3/18 and 3/18A are numeric and alphanumeric input modules, respectively. These modules are inputs for two-dimensional arrays. The size of the first dimension is set at two and the module asks for and sets the size of the second dimension. The data is then input into either column A or column B (the first dimension of the array determines which column is being input).

Module 3/19 is also input to a two-dimensional array. It asks for the size of each of the two dimensions and then accepts input for each item.

The last input module, Module 3/20, is two-dimensional, with the first dimension set at four. It asks for the size of each of the four elements of the first dimension and then lists the input in four columns.

Output

There is more variety in how we can display our results than in how we can input the original data. The possibilities for output are almost unlimited. We can print lists,

tables, bar graphs, and pie graphs; in color or in black and white; with or without column heads; and on and on.

The output format should help us to comprehend easily the information being presented. How the information should be presented will differ with each program, so most of the output programs are listed in other chapters, together with the specialized programs to which they apply. Here we have listed three output modules for general use (3/21, 3/22, and 3/23).

Module 3/21 will print a page containing up to 24 lines of alphanumeric data. The variable used is A$(P,X), where P is the page number and X is the line number of that page. In this module, typed characters can fill the page. Any formatting, such as debit/credit columns or outline forms, must be typed in character by character.

In contrast, Module 3/22 prints up to 20 lines of four columns. Each column contains seven characters. The variable used is A$(X,Y), where X is the row and Y is the column. This is a very convenient format to use for dollars-and-cents entries. The seven characters allow us to place a decimal point as the fifth character, leaving four places to the left and two places to the right of the decimal.

Module 3/23 is the module used in Chap. 2 to illustrate how class scores might be kept. This output module is numeric only and uses the variable A(B,C), where B denotes row and C denotes column. The module will not accept more than 10 columns of data and proportions the space to fit the number of columns required. This proportioning of space also limits the size of the numbers that go into the columns. This module provides row average, column totals, and column averages.

CONCLUSION

Certainly there are many other input and output possibilities. Similarly, data storage and retrieval formats can be as varied as your design requires. Still, the modules listed in this chapter can be used 70 percent of the time and with minor modifications should cover most of our needs.

As we said earlier, a complete book could be written just to cover input, output, and data storage. This book deals more with the useful manipulation of data. The following chapters provide modules with which to build programs. Like a book on chess, we have now covered the standard opening moves and end-game play. We now move on to the middle game, with special openings and endings.

CHAPTER THREE
MODULES

MODULE 3/1 Asks for the program name and LOADs it. Line 110 specifies the storage method used. If a Datassette is being used, change the 8 to a 1.

```
100 REM ** 3/1-NAME & LOAD **
110 DR=8
120 PRINT "[clear][4x dwn][2x rht]
    ENTER PROGRAM NAME TO LOAD"
130 INPUT"[6x rht]";PR$
140 PRINT"[clear][3x dwn]LOAD"
    CHR$(34) PR$ CHR$(34)",DR"
150 PRINT"[home]":POKE 198,1:POKE 631,
    13:END
```

MODULE 3/2 Asks for the program name, LOADs it, and RUNs it.

```
100 REM ** 3/2-NAME,LOAD,RUN **
110 DR=8
120 PRINT "[clear][4x dwn][2x rht]
    ENTER PROGRAM NAME TO LOAD"
130 INPUT "[6x rht]";PR$
140 PRINT"[clear][3x dwn]LOAD"
    CHR$(34) PR$ CHR$(34)",DR"
150 PRINT"[home]":POKE 198,4:POKE 631,
    13:POKE 632,82:POKE 633,213:
    POKE 634,13:END
```

MODULE 3/3 LOADs and RUNs a program whose name is given by PR$.

```
100 REM **          3/3-LOAD & RUN          **
101 REM ** LOADS PROGRAM NAMED PR$ **
110 DR=8
120 PRINT"[clear][3x dwn]LOAD"
    CHR$(34) PR$ CHR$(34)",DR"
130 PRINT"[home]":POKE 198,4:POKE 631,
    13:POKE 632,82:POKE 633,213:
    POKE 634,13:END
```

MODULE 3/4 SAVEs and VERIFYs a program named by PR$. Line 9010 specifies disk drive. If Datassette is used, change the 8 in line 9010 to a 1.

```
9000 REM **          3/4-SAVE & VERIFY          **
9001 REM ** SAVES PROGRAM NAMED PR$ **
9010 DR=8
9020 SAVE PR$,DR
9030 OPEN 15,DR,15:INPUT#15,A,B$:CLOSE15
9040 IF A=0 THEN 9080
9050 IF A=63 THEN PRINT"[clear][3x
     dwn]";PR$:PRINT
     B$;",OVERWRITE(Y/N)";:INPUT Z$
9060 IF Z$="Y" THEN PR$="@0:"+PR$:GOTO
     9020
9070 GOTO 9090
9080 VERIFY PR$,DR
9090 END:REM (OR RETURN IF A SUBROUTINE)
```

MODULE 3/5 Asks for the name of the program to be SAVEd, SAVEs it using the name given, and VERIFYs that it has been SAVEd properly.

```
9000 REM ** 3/5-NAME,SAVE,VERIFY **
9010 DR=8
9020 PRINT "ENTER PROGRAM NAME TO SAVE"
9025 INPUT PR$
9030 SAVE PR$,DR
9035 OPEN 15,DR,15:INPUT#15,A,B$:CLOSE15
9040 IF A=0 THEN 9080
9050 IF A=63 THEN PRINT"[clear][3x
     dwn]";PR$:PRINT
     B$;",OVERWRITE(Y/N)";:INPUT Z$
9060 IF Z$="Y" THEN PR$="@0:"+PR$:GOTO
     9030
9070 GOTO 9090
9080 VERIFY PR$,DR
9090 END:REM (OR RETURN IF A SUBROUTINE)
```

MODULE 3/6 Opens a sequential file to write data. Accepts alphanumeric data. A "return" closes the file and ends the program.

```
1000 REM ** 3/6-WRITE SEQ-ALPHA/NUM **
1001 REM **        INPUT FILE NAME      **
1002 REM **        WRITE SEQ FILE       **
1003 REM **          "" = STOP          **
1010 DR=8
1020 INPUT"ENTER THE FILE NAME";FL$:
     FL$="0:"+FL$:OPEN 15,DR,15
```

```
1030 OPEN 2,DR,2,FL$+",S,W"
1040 INPUT#15,A,B$:IF A=0 GOTO 1070
1050 PRINT"[2x dwn]";FL$:PRINT B$;
     ", OVERWRITE (Y/N)";
1055 INPUT Z$:IF Z$<>"Y"GOTO 1080
1060 FL$="@"+FL$:CLOSE 2:GOTO 1030
1070 X$="":INPUT X$:IF X$="" GOTO 1080
1075 PRINT#2,X$:GOTO 1070
1080 CLOSE 2:CLOSE 15:STOP
```

MODULE 3/7 Asks for a file name, opens the file, and GOSUBs to one of a number of possible input subroutines. On RETURN it closes the file and STOPs (or GOTOs another part of the program).

```
1000 REM **  3/7-WRITE SEQ GOSUB  **
1001 REM **  INPUT NAME,OPEN FILE **
1002 REM **     GOSUB FOR INPUT    **
1010 DR=8
1020 INPUT"ENTER THE FILE NAME";FL$:
     FL$="0:"+FL$:OPEN 15,DR,15
1030 OPEN 2,DR,2,FL$+",S,W"
1040 INPUT#15,A,B$:IF A=0 GOTO 1070
1050 PRINT"[2x dwn]";FL$:PRINT B$;
     ", OVERWRITE (Y/N)";
1055 INPUT Z$:IF Z$<>"Y"GOTO 1080
1060 FL$="@"+FL$:CLOSE 2:GOTO 1030
1070 GOSUB 1100:REM ** (choose input
     subroutine that is wanted) **
1080 CLOSE 2:CLOSE 15:STOP:REM ** (or GOTO
     other program) **
```

```
1100 REM ** SUBROUTINE FOR ONE-
     DIMENSIONAL NUMERIC ARRAY INPUT
1110 FOR X=1 TO B
1120 PRINT#2,A(X)
1130 NEXT X:RETURN
1150 REM ** SUBROUTINE FOR ONE-
     DIMENSIONAL STRING ARRAY INPUT
1160 FOR X=1 TO B
1170 PRINT#2,A$(X)
1180 NEXT X:RETURN
1200 REM ** SUBROUTINE FOR TWO-
     DIMENSIONAL NUMERIC ARRAY INPUT
1210 FOR X=1 TO B:FOR Y=1 TO C
1220 PRINT#2,A(X,Y)
1230 NEXT Y:NEXT X:RETURN
1250 REM ** SUBROUTINE FOR TWO-
     DIMENSIONAL STRING ARRAY INPUT
1260 FOR X=1 TO B:FOR Y=1 TO C
1270 PRINT#2,A$(X,Y)
1280 NEXT Y:NEXT X:RETURN
```

MODULE 3/8 Opens a sequential file using the file name represented by FL$. Does not check for existing file and uses the replace mode of file opening.

```
1000 REM ** 3/8-WRITE OVER SEQ  **
1001 REM **    STRING INPUT     **
1002 REM **    "" = STOP        **
```

```
1020 DR=8
1030 OPEN 2,DR,2,"@0:"+FL$+",S,W"
1040 X$="":INPUT X$:IF X$=""THEN 1060
1050 PRINT#2,X$:GOTO 1040
1060 CLOSE 2:STOP
```

MODULE 3/9 Exactly like Module 3/8 except that Module 3/9 GOSUBs to an input subroutine.

```
1000 REM ** 3/9-WRITE OVER SEQ GOS **
1001 REM **      GOSUB INPUT          **
1002 REM **      "" = STOP            **
1020 DR=8
1030 OPEN 2,DR,2,"@0:"+FL$+",S,W"
1040 GOSUB XXXX:REM (to an input
     subroutine)
1080 CLOSE 2:STOP:REM ** (or GOTO
     other program) **
1100 REM ** SUBROUTINE FOR ONE-
     DIMENSIONAL NUMERIC ARRAY INPUT
1110 FOR X=1 TO B
1120 PRINT#2,A(X)
1130 NEXT X:RETURN
1150 REM ** SUBROUTINE FOR ONE-
     DIMENSIONAL STRING ARRAY INPUT
1160 FOR X=1 TO B
1170 PRINT#2,A$(X)
1180 NEXT X:RETURN
```

```
1200 REM ** SUBROUTINE FOR TWO-
     DIMENSIONAL NUMERIC ARRAY INPUT
1210 FOR X=1 TO B:FOR Y=1 TO C
1220 PRINT#2,A(X,Y)
1230 NEXT Y:NEXT X:RETURN
1250 REM ** SUBROUTINE FOR TWO-
     DIMENSIONAL STRING ARRAY INPUT
1260 FOR X=1 TO B:FOR Y=1 TO C
1270 PRINT#2,A$(X,Y)
1280 NEXT Y:NEXT X:RETURN
```

MODULE 3/10 Asks for file name, OPENs file, GOSUBs for input, GOSUBs for output, CLOSEs, and STOPs.

```
2000 REM **    3/10-READ GOSUB    **
2001 REM **    INPUT FILE NAME    **
2002 REM **    GOSUB INPUT        **
2003 REM **    GOSUB OUTPUT       **
2010 DR=8
2020 INPUT"ENTER THE FILE NAME";FL$
2030 OPEN 2,DR,2,"0:"+FL$+",S,R"
2040 GOSUB YY:REM (to read subroutine)
2050 GOSUB XX:REM (to output subroutine)
2060 IF ST>0 THEN 2080
2070 GOTO 2040
2080 CLOSE 2:STOP
```

MODULE 3/11 Asks for a file name, OPENs the file, INPUT$ a two-dimensional array, CLOSEs the file, and STOPs.

```
2000 REM **    3/11-READ ARRAY     **
2001 REM **    FL$ = FILE NAME     **
2010 DR=8
2020 INPUT"ENTER THE FILE NAME";FL$
2030 OPEN 2,DR,2,"0:"+FL$+",S,R"
2040 FOR X=1 TO B:FOR Y=1 TO C
2050 INPUT#2,A(X,Y)
2070 NEXT Y:NEXT X
2080 CLOSE 2: STOP
```

MODULE 3/12 Asks for file name, OPENs the file, INPUTs X$, GOSUBs, CLOSEs, and STOPs.

```
2000 REM ** 3/12-READ SEQ STRING **
2001 REM **    INPUT FILE NAME     **
2002 REM **    GOSUB OUTPUT        **
2010 DR=8
2020 INPUT"ENTER THE FILE NAME";FL$
2030 OPEN 2,DR,2,"0:"+FL$+",S,R"
2040 INPUT#2,X$
2050 GOSUB XX:REM (to output subroutine)
2060 IF ST>0 THEN 2080
2070 GOTO 2040
2080 CLOSE2:STOP
```

MODULE 3/13 OPENs the file named by SF$, INPUTs X$, GOSUBs to ouput, CLOSEs, and STOPs.

```
2000 REM ** 3/13-READ SEQ STRING **
2001 REM **    FILE NAME SF$     **
2002 REM **    GOSUB OUTPUT      **
2010 DR=8
2020 OPEN 2,DR,2,"0:"+SF$+",S,R"
2030 INPUT#2,X$
2040 GOSUB XX:REM (to output subroutine)
2050 IFST>0 THEN 2070
2060 GOTO 2030
2070 CLOSE2:STOP
```

MODULE 3/14 Manually enter up to 20 items into the menu format.

```
200 REM **      3/14-MENU          **
201 REM **  FOR UP TO 20  ITEMS **
210 DIM M$(20)
211 M$(1)="":M$(2)=""
212 M$(3)="":M$(4)=""
213 M$(5)="":M$(6)=""
214 M$(7)="":M$(8)=""
215 M$(9)="":M$(10)=""
216 M$(11)="":M$(12)=""
217 M$(13)="":M$(14)=""
218 M$(15)="":M$(16)=""
219 M$(17)="":M$(18)=""
```

```
220 M$(19)="":M$(20)=""
225 N=0:PRINT"[clear]"
230 TP$="[4x spc][30x cmd/+]"
240 PRINT TP$:FORX=1 TO N+2:PRINT"[4x
    spc][cmd/+]";
250 IF X<N+1 THEN PRINT TAB(23-LEN(M$(X)
    ))M$(X)"="X TAB(33)"[cmd/+]":GOTO
    260
255 PRINT TAB(33)"[cmd/+]"
260 NEXT X:PRINT TP$
270 PRINT"[2x up]"TAB(7);:INPUT
    "CHOOSE ONE";Z$
280 IF VAL(Z$)<1 OR VAL(Z$)>N THEN PRINT
    "[up]"TAB(18)"[4x spc][dwn]
    ":GOTO 270
```

MODULE 3/15 Accepts data for the menu, modifies itself by placing data into a DATA line. Changes line 10 to GOTO 200.

```
  3 REM ** 3/15-MENU/DATA LINE  **
 10 REM 200
 20 D$="":PRINT"[clear]":Z=0
 30 PRINT"ENTER ITEM #";Z+1;" (*=STOP)";
    :INPUT A$
 40 IF A$="*"THEN 80
 50 IF LEN(D$+A$)>68 THEN PRINT "OVERFLOW
    DATA BY"LEN(D$+A$)-68:GOTO 80
 60 D$=D$+A$+",":Z=Z+1:IF Z>10 THEN Z=Z-
    1:GOTO 80
```

```
 70 GOTO 30
 80 D$=D$+"−1.":PRINT"[dwn][5x rht]
    [rvs] WAIT [off]"
 90 N=PEEK(65)+256*PEEK(66)
100 IF PEEK(N+I)=131 THEN 120
110 I=I+1:GOTO 100
120 FOR X=1 to LEN(D$):POKE(N+I+X),ASC
    (LEFT$(D$,1))
130 D$=RIGHT$(D$,LEN(D$)−1):IF LEN(D$)=0
    THEN 150
140 NEXT X
150 REM ** REPLACE REM WITH GOTO **
160 N=PEEK(43)+255*PEEK(44):I=3
170 IF PEEK(N+I)=143 AND PEEK(N+I−1)=0
    AND PEEK (N+I−2)=10 THEN GOTO 190
189 I=I+1:GOTO 170
190 POKE N+I,137
200 PRINT"[clear]":FOR X=1 TO 10
210 READ B$:IF LEN(B$) >26 THEN B$=LEFT$
    (B$,26)
220 IF VAL(B$)=−1 THEN GOTO 240
230 PRINT"[dwn]";TAB(26−LEN(B$));B$
    ;" = "X:NEXT X
240 INPUT"[2x dwn][4x rht][rvs]
    CHOOSE ONE[off]";Z$:IF VAL(Z$)
    >X−1 THEN PRINT"[4x up]":
    GOTO 240
250 DATA0000000000000000000000000000000
    0000000[0s to fill two lines]00000
```

MODULE 3/16 INPUTs a number with six digits, two of which are decimals. Works very well for dollars and cents.

```
200 REM ** 3/16-INPUT XXXX.XX NUMB **
201 REM **   6 DIGIT NUMBER WITH     **
202 REM **    2 DECIMAL PLACES       **
203 REM **      V=NUMBER             **
210 V=0
220 FOR X=1 TO 7:IF X=5 THEN PRINT".";:
    GOTO 260
230 GET A$:IF A$< CHR$(48) OR A$> CHR$
    (57) THEN 230
240 PRINT A$;
250 V=10*V+VAL(A$)
260 NEXTX
270 PRINT:V=V/100
280 PRINT V
290 REM ** RETURN FROM SUBROUTINE ? **
```

MODULE 3/16A The same as Module 3/16 except that it places the number in a string.

```
200 REM ** 3/16A-INPUT XXXX.XX STRG **
201 REM **   6 DIGIT NUMBER WITH     **
202 REM **    2 DECIMAL PLACES       **
203 REM **      V$=NUMBER            **
210 V$="":FOR X=1 TO 7:IF X=5 THEN A$=
    ".":GOTO 260
```

```
220 GET A$:IF A$< CHR$(48) OR A$> CHR$
    (57) THEN 230
230 IF A$="0" AND VAL(V$)=0 AND X<4 THEN
    A$="[spc]"
240 PRINT A$;
250 V$=V$+A$
260 NEXTX
270 PRINT:PRINT V$
280 REM ** RETURN FROM SUBROUTINE ? **
```

MODULE 3/17 An input module for a one-dimensional numeric array that asks for the number of elements in the array, and then asks for input to fill the array.

```
200 REM ** 3/17-INPUT NUMB ARRAY **
201 REM **          A(N)=ARRAY          **
210 INPUT"ENTER NUMBER OF ELEMENTS IN
    ARRAY";N:DIM A(N)
220 FOR X=1 TO N
230 PRINT "A(";X;"[lft] )=";
240 INPUT A(X)
250 NEXT X
260 REM ** RETURN FROM SUBROUTINE ? **
```

MODULE 3/17A The same as Module 3/17 except that it sets up a string array.

```
200 REM ** 3/17A-INPUT STRING ARRAY **
201 REM **          A$(N)=ARRAY          **
210 INPUT"ENTER NUMBER OF ELEMENTS IN
    ARRAY";N:DIM A$(N)
```

```
220 FOR X=1 TO N
230 PRINT "A$(";X;"[lft] )=";
240 INPUT A$(X)
250 NEXT X
260 REM ** RETURN FROM SUBROUTINE ? **
```

MODULE 3/18 An input module for a two-dimensional numeric array. The first dimension of the array is 2. The module asks for the number of elements in the second dimension of the array, and then asks for input to fill the array.

```
200 REM ** 3/18-INPUT NUMB ARRAY **
201 REM **        A(2,N)=ARRAY        **
210 INPUT"ENTER NUMBER OF ELEMENTS IN
    ARRAY";N:DIM A(2,N)
220 FOR X=1 TO N
230 PRINT"A(";X;"[lft] )=";:INPUT
    A(1,X):PRINT"[up]",,
240 PRINT"B(";X;"[lft] )=";:INPUT
    A(2,X)
250 NEXT X
260 REM ** RETURN FROM SUBROUTINE ? **
```

MODULE 3/18A The same as Module 3/18 except that it uses a string array.

```
200 REM ** 3/18A-INPUT STRING ARRAY **
201 REM **        A$(2,N)=ARRAY        **
210 INPUT"ENTER NUMBER OF ELEMENTS IN
    ARRAY";N:DIM A$(2,N)
```

```
220 FOR X=1 TO N
230 PRINT"A(";X;"[lft] )=";:INPUT
    A$(1,X):PRINT"[up]",,
240 PRINT"B(";X;"[lft] )=";:INPUT
    A$(2,X)
250 NEXT X
260 REM ** RETURN FROM SUBROUTINE ? **
```

MODULE 3/19 An input module for a two-dimensional numeric array. The module asks for the number of elements in each dimension of the array, and then asks for input to fill the array. The module can be changed to be a string array by adding "$" to each "A."

```
200 REM ** 3/19-INPUT NUMB ARRAY  **
201 REM **   ENTER DIMENSION SIZE **
202 REM **        A(B,C)=ARRAY       **
210 INPUT"ENTER NUMBER OF ELEMENTS IN
    FIRST DIMENSION";B
220 INPUT"ENTER NUMBER OF ELEMENTS IN
    SECOND DIMENSION";C:DIM A(B,C)
230 PRINT"[clear]X=1"
240 FOR X=1 TO B:FOR Y=1 TO C
250 PRINT TAB(5)X"/"Y"=";:INPUT A(X,Y)
260 NEXT Y:PRINT "X="X+1:NEXT X
270 REM ** RETURN FROM SUBROUTINE ? **
```

MODULE 3/19A The same as Module 3/19 but without the dimension-size entry. The dimension sizes are B and C.

```
200 REM ** 3/19A-INPUT NUMB ARRAY  **
201 REM **          A(B,C)=ARRAY          **
205 DIM A(B,C)
210 PRINT"[clear]X=1"
220 FOR X=1 TO B:FOR Y=1 TO C
230 PRINT TAB(5)X"/"Y"=";:INPUT A(X,Y)
240 NEXT Y:PRINT "X="X+1
250 NEXT X
270 REM ** RETURN FROM SUBROUTINE ? **
```

MODULE 3/20 An input module for a two-dimensional numeric array. The first dimension of the array is 4. The module asks for the number of elements in each of the second dimensions of the array, and then asks for input to fill the array. The input is listed in columns.

```
200 REM ** 3/20-INPUT NUMB ARRAY  **
201 REM **  N(X)=SIZE OF ELEMENTS  **
202 REM **        A(4,N(X))=ARRAY        **
210 FOR X=1 TO 4:PRINT"ENTER SIZE OF DIM
    #";X;:INPUT N(X):NEXT X:PRINT"[clear]"
220 PRINT "DIM 1","DIM 2","DIM 3",
    "DIM 4"
225 PRINT"[cmd/J]"TAB(10)"[cmd/J]"
    TAB(10)"[cmd/J]"TAB(10)"[cmd/J]"
230 LET X=1
235 Z=0:PRINT
```

```
240 IF N(1)>X-1 THEN PRINT"[up]";:
    INPUT A(1,X):Z=Z+1
250 IF N(2)>X-1 THEN PRINT"[up]",:
    INPUT A(2,X):Z=Z+1
260 IF N(3)>X-1 THEN PRINT"[up]",,:
    INPUT A(3,X):Z=Z+1
270 IF N(4)>X-1 THEN PRINT"[up]",,,
    :INPUT A(4,X):Z=Z+1
280 IF Z=0 THEN 300
290 LET X=X+1:GOTO 235
300 STOP:REM ** (or RETURN or GOTO)
```

MODULE 3/21 Prints pages of alphanumeric data stored in array A$(P,X). The page number (P) must be specified.

```
4000 REM ** 3/21-PAGED STRING DATA   **
4001 REM **  1ST DIM = PAGE NO (P)    **
4002 REM **  2ND DIM = LINE NO (X)    **
4003 REM ** PRINTS 20 LINES PER PAGE**
4010 PRINT"[clear]";TAB(30);"[rvs]
     PAGE[lft]"P"[lft]
     [off]"
4020 FOR X=1 TO 20
4030 IF A$(P,X)="" THEN 4060
4040 PRINT A$(P,X)
4050 NEXT X
4060 INPUT"NEXT PAGE (Y/N)";Z$:IF Z$="N"
     THEN STOP:REM ** (or GOTO ?) **
4070 P=P+1:GOTO 4010
```

MODULE 3/22 Prints up to 20 lines of four columns. Each column contains seven characters.

```
2000 REM ** 3/22-4COL X 20 LINES      **
2001 REM **  7 CHARACTERS/BOX         **
2002 REM **  STRING ARRAY=A$(X,Y)     **
2003 REM **   X=ROW  Y=COLUMN         **
2010 PRINT"[clear][rvs]NO.[3x spc]
     COL.1[4x spc]COL.2[4x spc]COL.3
     [4x spc]COL.4 [off]"
2020 FOR X=1 TO 20:PRINT X;:FOR Y=1 TO 4
2030 PRINT TAB(9*(Y-1)+5)"[lft]"A$
     (X,Y)"[lft]";:NEXT Y
2040 PRINT: NEXT X
```

MODULE 3/23 Prints an array by row and column. Both the number of columns (not to exceed 10) and the number of rows are input as variables of the array.

```
250 REM ** 3/23-NUMERIC ARRAY      **
251 REM **   VARIABLE NUMBER OF    **
252 REM **     ROWS AND COLUMNS    **
260 FOR Y=1 TO C:PRINT Y"[spc]";:FOR X=1
    TO B
265 PRINT TAB(X*34/(B+1))A(X,Y);
270 SUM(X)=SUM(X)+A(X,Y):ADD=ADD+A(X,Y):
    NEXT X
275 ADD=INT(10*ADD/B+0.5)/10
280 PRINT TAB(33) ADD:ADD=0
```

```
285 IF Y>23 THEN INPUT"ENTER TO SCROLL";
    Z$:PRINT"[up][17x spc]
    [up]"
290 NEXT Y:PRINT "COLUMN TOTAL"
295 FOR X=1 TO B:PRINT TAB(X*34/(B+1)-2)
    SUM(X);:NEXT X
```

Chapter FOUR

HOME AND PERSONAL

Many programs that are useful around the home also have uses in the office. Similarly, many office programs can have personal uses. In this chapter and the next, you'll find many modules that can be helpful at the office, around the home, and for personal use.

CARD FILE MODULES

The first series of modules in this chapter all relate to a card file program. With them we can build a number of different types of files. For example, we can build

1. Alphabetized information cards
2. Recipe files
3. Christmas card and gift records
4. Name and address directories
5. And many others

Some of the same elements used in the card file can also be used to construct:

1. Message center programs
2. Calendar/reminders
3. Grocery list programs
4. Appointment books
5. And many others

The cards come in two forms. The first form (Module 4/2A) provides a card that contains 14 lines with 28 characters per line. This first form displays the contents of the card called for and displays the titles of the four cards that follow the called card. This feature can be most helpful when searching the card file.

The second card file format (Module 4/2B) provides a single card whose size can be adjusted to fit our needs. This card has a maximum size of 20 lines with 36 characters per line. It has a minimum size of eight lines with 20 characters per line. The parameters of the card may be changed to meet our requirements. This variable format (Module 4/2B) is suggested for programs such as the message center, calendar/reminder, and other uses requiring more data space than is provided on the smaller format of Module 4/2A.

Whatever card format we choose for our program, we will want to write on the cards (enter data), erase any mistakes (delete/backspace), and store the card file (save file). Later, we will want to retrieve the file (load file), search the file (search), display the desired card (display), and perhaps erase the entire card contents (delete card).

We need a menu when we want to choose among entering new data, searching the file, or deleting informa-

tion. There are menu modules listed which provide menus for a card file, a message center, a calendar/reminder, a grocery list, and a telephone directory.

The line numbers of all the modules are arranged so that the modules can be used just as they appear. Modules are provided for data security, and to search for cards by "last name" rather than by the entire title.

Let's put together a simple card file. First, we must choose our card format. The small card will give us adequate space (392 characters per card) and will provide easy scanning of the information stored. Second, we should enter into the computer Initial Module 4/1. On line 20 of this module, we change LI = 20 to LI = 14 and change WI = 36 to read WI = 28. These changes match the number of lines (LI) and the number of characters per line (WI) of the small card. Now, enter Small Card Module 4/2A into the computer.

Third, we choose the card file menu (Module 4/3A) and enter it. If we wanted a Christmas card list or recipe file rather than a card file, we would design the Christmas card menu or the recipe file menu and enter it into the computer.

Fourth, enter Print Module 4/4, Write Module 4/5, and Delete/Backspace Module 4/6. Entering these modules into the computer will give us an almost complete card file system. These three modules are used with all the card files. The card format module chosen and the parameters given in the initial module will control how the print, write, and delete modules function.

Now, add Card File Search Module 4/8A and Common Search Module 4/8C. With these additions we have a fully functional card file. We can write on the cards, erase our errors, search the file for information, and print that information on the screen. Pretty good!

Finally, unless we want to use the computer only as an expensive scratchpad, we need to be able to save and retrieve the cards. To do this, we add Module 4/9 (error channel), Module 4/10 (save cards), and Module 4/11 (load cards). These modules will allow us to save and retrieve card files under different names. Thus we could have both personal and business card files on the same disk and call them up as needed. We could have a card file named "Christmas," another named "Telephone," and a third named "Clients." Each file could be retrieved and used separately by the one-card file program that we have assembled.

OTHER USES FOR CARD FILE MODULES

In addition to the many uses for the card file programs, a number of the card file modules can be used in other programs. The large card format can be used for family records, message centers, and even as a grocery list reminder.

Grocery Lists

The grocery list can be built up over a period of time by adding new items as needed. The new items are then saved as part of the total list. Before printing out our shopping list we can scan the computer list and enter an asterisk beside the items that we want to print. When we print, only the items for this week's shopping list are printed.

The asterisk function is provided by Module 4/12A. Grocery List Menu Module 4/12B includes the add-on

and save functions. The number of characters per line should be set so that WI = 21 (the minimum size). The grocery list can be assembled using the following modules:

4/1	Card initial
4/2B	Single card form
4/5A	New card
4/7	Bottom line/ erase card
4/9	Error channel
4/10	Save file
4/12A	Asterisk adder
4/12B	Grocery list menu

Once we understand how to make the needed changes, it is surprising how easy it is to assemble card files to meet our individual needs. If we have all the modules on a disk, we just lay out our design and put it together from the disk. There will usually be some adjustments to be made in LI and WI. Often, there will be some other changes, but with a little trial-and-error manipulation the modules will usually work well.

Family Records

For instance, family records are easily set up. Here, again, we will use the large card format. Set up the normal card system using the basic modules. Because we may want to change or add to the data on the card from time to time, we should include Add-On Module 4/13. If we require security, we can use Security Module 4/8B rather than the normal Search Module 4/8A.

```
4/1    Initial
4/2B   Single card form
4/3A   Menu
4/4    Print card copy
4/5A   Write
4/6    Delete/backspace
4/7    Erase card
4/8B   Security
4/8C   Common search
4/9    Error channel
4/10   Save
4/11   Load
4/13   Add-on
```

When we use the completed program, we may want to decide on the titles wanted, such as "Inventory," "Life Insurance," "John Smith Medical," and so on. We make card 1 a list of these titles so that we do not forget what titles we have used. We may want to name card 1 "Title Index." Use these names for the headings of the other cards. After heading each card, type in the data on the lines under each title.

Now we can find those records! Using search, we can find the "Life Insurance" card (or cards), which will list our family's policies. If we search, using "Insurance" as a last name, we can scan through our life, auto, house, and other insurance listings. The card file will furnish a good method for record storage and retrieval.

The message center program is another program that can have many uses. Not only can we leave messages for

others with this program, we can also leave notes and reminders for ourselves. If we use Search Module 4/8A, all messages are open, so that everyone has access to all messages. Alternatively, we can use Security Module 4/8B. Although this module does not provide a very sophisticated form of security, it does increase the difficulty for accessing messages meant for others.

The security module asks for your last name and then asks for your code number. The code is the sum of the computer's character codes for the first three letters of your last name. Thus the computer would take the name Smith and sum ASC(S) + ASC(m) + ASC(i) to arrive at a code number of 83 + 77 + 73 = 233. This number will be the code number for the name Smith. Not a very sophisticated security system, but it is certainly better than no security system.

If we do not want to look up the ASC numbers to determine our correct codes, the following subprogram will provide the code for any name entered:

```
7000 REM ** SECURITY CODE **
7010 INPUT"ENTER YOUR LAST NAME";NM$
7020 LET NM%=ASC(NM$)+ASC(MID$(NM$,2,1))
     +ASC(MID$(NM$,3,1))
7030 PRINT NM%:STOP
```

An appointment/reminder calendar can be made by including Calendar Initial Module 4/14, Add-On Module 4/13, Up-Date Module 4/15, and Menu Module 4/16. The calendar module will title all the cards with consecutive dates. We can then access any date so as to place a reminder on a specific card. As we access the current day's

date, the entire set of cards is moved up to "throw away" the out-of-date cards.

You will find many uses for the modules in these card file programs. They are useful, basic information storage programs.

Loans, Savings and Interest

As you have noticed by now, the menu module is used a lot. It allows us to choose among a number of possible options. The modules in this section may all be grouped in a single program and accessed through a menu. They may be used individually or as part of a larger financial program.

Each module has a specific function. Although some are only two or three lines in length, they are, nonetheless, complete programs. Each asks for an input, performs its function, and then prints the resulting output. In each case we can easily modify the module to use input coming from previous program modules. This is done either by deleting the lines ending in 5 through 9 (where the input program is requested) or by having the previous program GOTO the line ending in 10 (the first operation line of the module).

Similarly, the output of the module can, if desired, be used as input for other modules or printed in a format that we specify. All that is required is to replace the output or print line of the module with a GOTO line, which will direct the program to the next module or printing format.

It is possible that we would want a program that would allow us to input a number of different loan parameters and then print the different results for comparison. In such a case the module might be used as a subroutine. Consider the following:

```
100 INPUT"ENTER AMOUNT TO BE
    BORROWED";P
110 INPUT"ENTER LENGTH OF LOAN
    IN DAYS";T
120 INPUT"WHAT RATE OF
    INTEREST";R:R=R/100
130 GOSUB 410:PRINT"RATE ="
    100*R"% INTEREST=$"I
    "MATURITY VALUE=$"MV
140 PRINT:GOTO 120
400 REM MATURITY VALUE
410 I=P*R*T/365
420 MV=P+I
430 RETURN
```

Notice that we have used Module 4/17 as the subroutine, starting at line 400. We have removed lines 405, 406, and 407 (the input lines) and have replaced line 430 (the print output line) with RETURN. As long as we continue to input interest rates, the program will print out interest and maturity values for the principal and the term we have specified.

Obviously, more complicated programs can be devised easily. For example, a series of possible rates can be entered by the computer, and the results can be listed to show the change in maturity value over this range of possible interest rates. Or the computer could enter a range of payment terms (i.e., FOR T = 30 TO 360 STEP 30) to show the effects of different terms on the maturity value. The possible variations are many, and all the modules in this section can be modified similarly.

In all these modules, be careful to note that the inter-

est is expressed as a decimal. If you wish to enter interest as a percent (i.e., 15 percent rather than 0.15), you must divide the input by 100. This can be done in Module 4/17 by changing line 406 to read

```
406 INPUT"ENTER % INTEREST
    RATE";R:R=R/100
```

Module 4/18 calculates interest on a declining balance when the monthly payment and rate of interest are fixed. The input required is the original amount owed (P), the annual rate of interest expressed as a decimal (R), the amount of the monthly payment (S), and the number of months over which we want the calculation made (N). The module then calculates: the total interest for N months, the number of payments made (if the loan is paid off prior to N months), the amount of the last payment (if the loan is paid off prior to N months), and the outstanding balance (if the loan is not paid off prior to N months). This method of calculating interest is used on many "revolving account" payments made on consumer appliances, furniture, and similar items.

Module 4/19 calculates the bank discount of a payable note. If we know the discount rate charged by the bank, we can calculate the present value (the payoff value) of the loan. The module input requires the parameters of the loan: the principal or amount of the loan (P), the rate of interest (R), and the length of the note in days (T). Then the module needs the discount rate (R2) and the discount period in days (T2). The discount period is the time between now and the end of the note period. For instance, if we have a 90-day note and want to pay it off at the end of 75 days, the discount period is 15 days (90 − 75 = 15).

Module 4/20 will work out the maturity value and

amount of interest paid on a compound-interest note. Put in the initial amount of the loan (P), the annual rate of interest (R), the number of times per year that the note is compounded (C), and the number of years to maturity (N). The module will then calculate the total interest to be paid and the maturity value of the note.

Modules 4/21 and 4/22 calculate the number of payments needed to pay off a note. Module 4/21 calculates the number of payments needed to pay off an interest-bearing note if the number of the payments is known. Thus if we have an amount budgeted for a loan, we can put into the computer the amount of the loan (P), the annual rate of interest (R), and the monthly amount that we have budgeted to pay (S). The computer will then tell us the number of months needed to pay off the note (N) and the amount of the last payment (LS). Neat!

Module 4/22 is the opposite of Module 4/21. If we know the number of months over which we plan to pay off a note, Module 4/22 will calculate what our monthly payment will be. If we want to decide whether to pay off the car in 24 months, 36 months, or 48 months, this module will give us the different monthly payments for the various periods. Input the amount to be financed (P), the annual rate of interest (R), and the number of months for the term of the loan. Then the computer will give us "S," the monthly payment required.

Modules 4/23 and 4/24 calculate the amounts that can be saved in a savings account, an annuity, an IRA (Individual Retirement Account), and so on, in a certain period of time. If we want to end up with a certain amount of money by a certain date, Module 4/23 will tell us how much we must deposit per month to meet that goal. On the other hand, if we want to deposit a certain amount into our account each month, Module 4/24 will tell us how much

we will have in the account at the end of the period of time that we select.

We must input the number of months (N) and the annual rate of interest paid (R) into both of the modules. We must also input the required end value (SF) into Module 4/23. The computer will then output the monthly deposit required (D). Module 4/24 is just the opposite of Module 4/23. Module 4/24 requires the monthly deposit as input, and outputs the value of the fund after N months.

CONCLUSION

In this chapter we have covered a large number of ways in which a set of card file modules may be used. The possibilities have by no means been exhausted. These modules can be used for almost any information storage application. They require only your imagination and ingenuity to assemble.

We also have covered a number of savings and payment schedules to be calculated. These modules (4/17 to 4/24) may best be assembled into a large program with a menu that allows selection of the wanted calculation. These modules will be useful to almost anyone at some time or other.

In the next chapter we design the elements required to assemble a checkbook register.

CHAPTER FOUR
MODULES

MODULE 4/1 Contains the initial parameters of the card files. The size of LI and WI set the number of lines and the number of characters per line, respectively.

```
10 REM ** 4/1-CARD INITIAL                **
11 REM **  LI=NO. OF LINES                 **
12 REM **  WI=NO. OF CHARACTERS            **
13 REM ** 400=LOAD CDS 1000=MENU **
20 LI=20:WI=36:FOR I=LI TO20:ST$=ST$+"
   [dwn]":NEXT I:ST$="[home]"+ST$
30 FOR I=1 to WI:D$=D$+CHR$(32):CT$=CT$
   +CHR4(183):CB$=CB$+CHR$(175):NEXT I
40 CT$=CHR$(207)+CT$+CHR$(208):CB$=CHR$
   (204)+CB$+CHR$(186)
50 CS$=CHR$(180)+D$+CHR$(170)
60 DIM C$(100,LI)
70 GOSUB 400:REM ** TO LOAD DATA **
80 GOTO 1000:REM ** TO PROGRAM START **
```

MODULE 4/1Y Begins at card 1 and cycles (flips) through the cards until a card is reached that has no data on it. The module program then returns to the menu (GOTO 1000).

```
2000 REM ** 4/1Y-FLIP CARDS       **
2010 FOR N=1 TO 100:SG=0
2020 IF C$(N,1)="" THEN 1000
```

```
2030 GOSUB 800:REM ** TO CARD WRITE **
2040 IF SG=3 THEN SG=0:RETURN
2050 IF LEFT$(Z$,1)="*" THEN 1000
2060 NEXT N:GOTO 1000
```

MODULE 4/2A The format for a small card (28 characters × 14 lines). It includes display of the titles of the four cards that follow in sequence. WI and LI must be changed in Initial Module 4/1 to WI = 28 and LI = 14.

```
1000 ** 4/2A-SMALL CARD FORM     **
1001 **    NEEDS CB$,CT$,& CS$    **
1002 **        FURNISHED IN #4/1    **
1010 CC=14:PRINT"[clear]":SS$=CHR$
     (166)+CHR$(170)
1015 FOR X=1TO5
1020 PRINT TAB(11-2*X) CT$;:IF X=1 THEN
     PRINT:GOTO 1040
1030 FOR Y=1 TO X-1:PRINT SS$;:NEXT Y:
     PRINT
1040 PRINT TAB(11-2*X) CS$;:IF X=1 THEN
     PRINT: GOTO 1060
1050 FOR Y=1 TO X-1:PRINT SS$;:NEXT Y:
     PRINT
1060 NEXT X
1070 FOR X=1 TO 7
1080 PRINT TAB(1) CS$ SS$ SS$ SS$ SS$
1090 NEXT X
1100 FOR X=1 TO 4
1110 PRINT TAB(1) CS$;:IF X=4 THEN PRINT
     CHR$(183);CHR$(183):GOTO 1150
```

```
1120 FOR Y=3 TO X STEP -1:PRINT SS$;:
     NEXT Y:PRINT CHR$(183);CHR$(183)
1130 PRINT TAB(1) CS$;:FOR Y=3 TO X
     STEP -1:PRINT SS$;:NEXT Y:PRINT
1140 NEXT X
1150 PRINT"[rht]" CB$ "[home]
     [10x dwn][2x rht]]";
```

MODULE 4/2B An adjustable format for a card. The card size and shape can be changed in Module 4/1 (the initial module) by entering the desired number of characters per line into WI and the desired number of lines per card into LI.

```
1000 REM ** 4/2B-SINGLE CARD FORM  **
1001 REM ** NEEDS ST$,CT$,CS$ & CB$ **
1002 REM **       FURNISHED IN 4/1      **
1003 REM **           100=MENU          **
1010 CC=0:PRINT"[clear]" ST$ "[rht]"
     CT$
1020 FOR X=1 TO LI+1:PRINT"[rht]"
     CS$:NEXTX
1030 PRINT"[rht]" CB$;
1050 GOTO 100:REM ** TO MENU MODULE **
```

MODULE 4/3A Prints on a card format a menu for a card file (including Christmas cards and address files). The values on line 150 can be changed to fit your program.

```
100 REM **       4/3A-CARD MENU        **
101 REM **  (2000=FLIP 3000=SELECT **
102 REM **   4000=NEW 5000=SAVE)    **
```

```
110 FLAG=0:PRINT ST$"[dwn][rvs]
    CHOOSE ONE [off]"
120 PRINT:PRINT"[4x rht] 1= FLIP
    CARDS":PRINT"[4x rht] 2= SELECT
    CARD"
130 PRINT"[4x rht] 3= NEW CARD"
    PRINT"[4x rht] 4= SAVE FILE"
140 INPUT"[dwn][10x rht]";Z$
150 ON VAL(Z$) GOTO 2000,3000,4000,5000
160 STOP
```

MODULE 4/3B Prints on a card format a menu for a message center. The values on line 340 can be chagned to fit your program.

```
300 REM ** 4/3B-MESSAGE CENTER MENU **
310 FLAG=0:PRINT ST$"[dwn][2x rht]
    [rvs] CHOOSE ONE [off]"
320 PRINT:PRINT"[4x rht] 1= RECEIVE
    MESSAGE":PRINT"[4x rht] 2= LEAVE
    MESSAGE"
330 PRINT"[4x rht] 3= SAVE MESSAGES"
    :PRINT"[4x rht] 4= LOAD MESSAGES"
340 INPUT"[dwn][10x rht]";Z$:
    ON VAL(Z$) GOTO 2500,3500,5000,5500
```

MODULE 4/4 Prints the card copy on the card format.

```
800 REM ** 4/4-PRINT CARD COPY **
810 GOSUB 550:PRINT"[home]":IF LI<14
    THEN 830
```

```
820 FOR B=1 TO 4:PRINT TAB(12-2*B);
    C$(N+5-B,1);"[dwn]":NEXT B
830 PRINT ST$:FOR I=1 TO LI:PRINT
    "[2x rht]";C$(N,I):NEXT I
840 GOSUB 500:RETURN
```

MODULE 4/5A Writes on the card what you type. Entering the INST/DEL causes the program to GOSUB to the delete/backspace subroutine in line 600.

```
4000 REM ** 4/5A-WRITE ON CARD      **
4001 REM **   GOSUB 600 TO DELETE  **
4005 GOSUB 550: REM ** CLEAR CARD
4010 FOR N=1 TO 100:IF C$(N,1)=""
     THEN 4025
4020 NEXT N:PRINT"[3x rht][rvs]
     NO NEW CARDS [off]":STOP
4025 GOSUB 200:REM ** TO MOD.4/5C **
4030 REM ** CARD BODY INPUT  ***
4035 PRINT ST$"[4x rht] RETURN=LINE
     END, £=PAGE END":PRINT ST$"[2x dwn]
     [2x rht]";
4040 FOR L=2 TO LI:PRINT:PRINT
     "[2x rht]";
4045 GET A$:IF A$="" THEN 4045
4050 IF A$=CHR$(13) THEN 4075
4055 IF A$=CHR$(20) THEN GOSUB 600:
     GOTO 4045
4060 C$(N,L)=C$(N,L)+A$
4065 IF A$="£" THEN 4080
```

```
4070 PRINT A$ "[cmd/+][lft]";:
     IF LEN(C$(N,L))<WI THEN 4045
4075 PRINT"[2x rht]";:NEXT L
4080 PRINT:GOSUB 500:IFZ$<>"*" THEN 1000
4090 STOP
```

MODULE 4/5B Writes the card title on the card as you type then prompts with "TITLE:", "COMPANY:", "ADDRESS:", "CTY, ST, ZIP:", and "TELEPHONE:". Data is entered and printed on the card as you type. Entering the INST/DEL causes the program to GOSUB to the delete/backspace subroutine at line 600.

```
4000 REM ** 4/5B-WRITE ADDRESS CARD **
4001 REM **    GOSUB 600 TO DELETE    **
4005 LI=8:GOSUB 550: REM ** CLEAR CARD
4010 FOR N=1 TO 100:IF C$(N,1)=""
     THEN 4030
4020 NEXT N:PRINT"[3x rht][rvs]
     NO NEW CARDS [off]":STOP
4030 GOSUB 200:REM ** TO MOD.4/5C **
4035 PRINT ST$"[4x rht] RETURN=LINE
     END, £=PAGE END":PRINT ST$"[2x dwn]
     [2x rht]";
4040 FOR L=2 TO LI
4045 IF L=2 THEN C$(N,2)=D$:PRINT
     C$(N,L):GOTO 4085
4050 IF L=3 THEN C$(N,L)="[6x rht]
     TITLE:[spc]"
4055 IF L=4 THEN C$(N,L)="[4x rht]
     COMPANY:[spc]"
```

```
4060 IF L=5 THEN C$(N,L)="[4x rht]
     ADDRESS:[spc]"
4065 IF L=6 THEN C$(N,L)="[rht]
     CTY,ST,ZIP:[spc]"
4070 IF L=7 THEN C$(N,L)="[2x rht]
     TELEPHONE:[spc]"
4075 IF L=8 THEN C$(N,8)="£":GOTO 4085
4080 PRINT C$(N,L);:GOSUB 4100
4085 NEXT L:GOSUB 500:IF Z$="*" THEN
     STOP
4090 GOTO 1000
4100 GET A$:IF A$="" THEN4100
4110 IF A$=CHR$(13) THEN PRINT:RETURN
4120 IF A$=CHR$(20) THEN GOSUB 600:
     GOTO4100
4130 PRINT A$;:C$(N,L)=C$(N,L)+A$
4140 IF LEN(C$(N,L))<WI THEN 4100
4150 PRINT:RETURN
```

MODULE 4/5C The subroutine that places a title line on the top of each card. It is used in both Modules 4/5A and 4/5B.

```
200 REM ** 4/5C-CARD TITLE INPUT ***
235 IF LI=14 THEN GOSUB 550
240 PRINT ST$"[2x dwn][2x rht]
    [rvs]" D$;"[off]" ST$
    "[5x rht] ENTER CARD TITLE,
    £=END":N$=STR$(N)+"[spc]"
245 IF N<10 THEN N$="[spc]0"+RIGHT$
    (N$,2)
```

```
250 C$(N,1)=N$:PRINT:PRINT
    "[2x rht]";C$(N,1);
255 GET A$:IF A$="" THEN 255
260 IF A$=CHR$(20) THEN L=1:GOSUB 600:
    GOTO 255:REM ** DELETE/BACKSPACE **
265 IF A$<>"£" THEN 275
270 FOR I=LEN(C$(N,1)) TO WI-1:C$(N,1)
    =C$(N,1)+"[spc]:PRINT"[spc]";:NEXT I
    :GOTO 285
275 PRINT A$;:C$(N,1)=C$(N,1)+A$:IF
    LEN(C$(N,1))=WI THEN 285
280 GOTO 255
285 PRINT ST$"[4x rht]" D$:PRINT
290 RETURN
```

MODULE 4/6 Deletes the last character printed and backs up one space to the next-to-last character printed.

```
600 REM ** 4/6-DELETE & BACKSPACE **
601 REM **          SUBROUTINE          **
610 IF LEN(C$(N,L))=0 THEN L=L-1
620 FOR K=WI TO 39:PRINT"[lft];:
    NEXT K
630 C$(N,L)=LEFT$(C$(N,L),LEN(C$(N,L))
    -1)
640 PRINT"[lft][spc][lft]";
650 RETURN
```

MODULE 4/7 A combination of two subroutines which are used often in these card file programs. The first (lines 500 through 510) is copy for input on the bottom line of

the card. The second (lines 550 through 570) erases the card.

```
500 REM ** 4/7-BOTTOM LINE/ERASE **
510 INPUT"[2x rht][rvs] RETURN
    [off] TO CONTINUE, *=STOP";Z$
    :RETURN
550 REM ** ERASE CARD COPY **
560 IF CC=14 THEN PRINT"[home]":FOR
    X=1 TO 4:PRINT TAB(12-2*X)D$
    "[dwn]":NEXT X
570 PRINT ST$:FOR X=1 TO LI+1:PRINT
    "[2x rht]"D$:NEXT X:RETURN
```

MODULE 4/8A Prepares to search the card file by card number, title (or part of title and the symbol "?"), or by last word (last name) of title.

```
3000 REM ** 4/8A-SEARCH FOR CARD **
3001 REM ** MUST BE USED WITH **
3002 REM ** 4/8C (COMMON SEARCH) **
3005 GOSUB 550
3010 PRINT ST$"[2x dwn][7x rht]
    [rvs] SELECT CARD [off]"
3020 PRINT"[dwn][2x rht] ENTER
    NUMBER";CHR$(13);"[10x rht] OR"
3030 PRINT"[2x rht] ENTER TITLE
    (?=UNKNOWN)"CHR$(13);"[10x rht]
    OR"
3040 PRINT"[2x rht] ENTER * AND
    LAST NAME"
```

```
3050 FLAG=0:LN=0:F$="":SR$="":PRINT
     "[dwn][5x rht]";
3055 PRINT"[cmd/+][lft]"
3060 GET A$:IF A$="" THEN 3060
3065 IF A$=CHR$(13) THEN LN=LEN(F$):
     GOSUB 3200:GOTO 1000
3070 PRINT A$;:IF A$="*" AND F$="" THEN
     FLAG=1:PRINT"[spc]";:GOTO 3055
3080 IF A$="?"THEN LN=LEN(F$):
     GOSUB3200:GOTO 1000
3090 F$=F$+A$:GOTO 3055
```

MODULE 4/8B A simple security system which asks for a last name and a code number. If the code number is the sum of the ASC values of the first three letters of the name, the name is entered as the subject of the card search. Module 4/8C must be used with this system.

```
3000 REM ** 4/8B-SECURITY FOR SEARCH **
3001 REM **   MUST BE USED WITH 4/8C   **
3010 GOSUB 550
3020 PRINT ST$"[3x dwn]
     [4x rht]";:INPUT"ENTER YOUR
     LAST NAME";F$
3030 IF RIGHT $(F$,1)="[spc]" OR
     RIGHT$(F$,1)=CHR$(170) THEN F$=
     LEFT$(F$,LEN(F$)-11):GOTO 3030
3040 INPUT"[2x dwn][4x rht]
     ENTER YOUR CODE";N1$
3050 IF VAL(LEFT$(N1$,4))=ASC(F$)+
     ASC(MID$(F$,2,1))+ASC(MID$(F$,3,1))
     THEN 3070
```

```
3060 PRINT"[2x rht][rvs] INVALID
     CODE - PROGRAM ABORTED ! [off]":
     STOP
3070 FLAG=1:LN=LEN(F$):GOSUB 3200:
     GOTO 1000
```

MODULE 4/8C The search subroutine used with Modules 4/8A and 4/8B. After the parameters are set in the preceding module, Module 4/8C does the actual search.

```
3200 REM ** 4/8C-COMMON SEARCH     **
3210 GOSUB 550:FOR Y=1 TO 100
3215 IF C$(Y,1)="" THEN 3260
3220 IF LEN(F$)<4 AND VAL(F$)<100 AND
     VAL(F$)>0 THEN 3300
3230 IF FLAG=0 THEN SR$=MID$(C$(Y,1),
     5,LN):GOTO 3250
3240 IF FLAG=1 THEN LG=WI:REM LAST NAME
     ROUTINE
3241 IF MID$(C$(Y,1),LG,1)="[spc]" THEN
     LG=LG-1:GOTO 3241
3242 FOR M=LG-1 TO 1 STEP -1
3243 IFMID$(C$(Y,1),M,1)="[spc]"
     THEN 3245
3244 NEXT M:GOTO 3250
3245 LET SR$=MID$(C$(Y,1),M+1,LN)
3250 IF F$=SR$ THEN 3280
3260 NEXT Y:GOSUB 550:REM GOSUB TO #4/7
3270 PRINT ST$"[4x dwn][2x rht]
     [rvs] NOT FOUND, ENTER RETURN
     [off]":INPUT"[2x rht]";Z$:
     GOTO1000
```

```
3280 SG=3:N=Y:GOSUB 800:IF LEFT$(Z$,1)
     ="*" THEN RETURN:REM ** 800=#4/4 **
3290 GOTO 3260
3300 SR$=MID$(C$(Y,1),2,2):IF VAL(SR$)=
     VAL(F$) THEN SG=3:N=Y:GOSUB 800:
     RETURN:REM ** 800 = #4/4 **
3310 GOTO 3260
```

MODULE 4/9 Opens the error channel and reads the error message. If the file named to be saved exists, the module asks "overwrite?". If the error is of a different nature, the error is printed and the program CLOSEs the files and RETURNs.

```
5200 REM ** 4/9-READ ERROR CHANNEL **
5210 INPUT#15,A$,B$:IF VAL(A$)=0 THEN
     RETURN
5220 IF A$="63" THEN PRINT"[dwn][2x
     rht]"B$;:INPUT", OVERWRITE? [rvs] Y
     [off] /N [dwn][8x lft]";Z$
5230 IF LEFT$(Z$,1)="N" THEN 5260
5240 IF A$<>"63" THEN PRINT"[4x rht]" A$
     "[spc]" B$:GOTO 5260
5250 CLOSE 2:FL$="@"+FL$:RETURN
5260 CLOSE 2:CLOSE 15:RETURN
5270 GOTO 1000
```

MODULE 4/10 Asks for the name of a card file to be saved, checks the error channel, and saves the file.

```
5000 REM ** 4/10-SAVE CARD FILE    **
5010 GOSUB 550:PRINT ST$"[3x dwn]
```

```
      [3x rht][rvs] INPUT NAME OF
      FILE TO SAVE":PRINT:Z$="":FL$=""
5020  INPUT"[2x rht]";FL$:IF LEN(FL$)
      >15 THENFL$=LEFT$(FL$,16)
5030  IF RIGHT$(FL$,1)="[spc]"THEN FL$=
      LEFT$(FL$,LEN(FL$)-1):GOTO 5030
5040  FL$="0"+FL$
5050  GOSUB 550:PRINT ST$"[3x dwn]
      [3x rht][rvs] IF FILE IS
      READY TO SAVE"
5060  PRINT"[3x rht] THEN ENTER
      [rvs] RETURN [off]"
5070  Z$="":GET Z$:IF Z$="" THEN 5070
5080  IF Z$<>CHR$(13) THEN 1000
5100  OPEN 15,8,15:Z$="":A$=""
5110  OPEN 2,8,2,FL$+",S,W":GOSUB 5200:
      IF A$="63" AND Z$="Y" THEN 5110
5120  IF VAL(A$)>0 THEN GOSUB 500:
      IF Z$<>"*" THEN 1000
5125  IF Z$="*" THEN STOP
5130  FOR J=1 TO 100:IF C$(J,1)=""
      THEN 5180
5140  FOR Y=1 TO LI:IF C$(J,Y)="" THEN
      C$(J,Y)="£"
5150  PRINT#2,C$(J,Y):IF
      RIGHT$(C$(J,Y),1)=CHR$(92) THEN 5170
5160  NEXT Y
5170  NEXT J
5180  CLOSE 2:CLOSE 15:PRINT"[dwn]
      [2x rht] SAVE COMPLETE
```

```
      [dwn]":GOSUB 500:
      IF LEFT$(Z$,1)="*" THEN STOP
5190 GOTO 1000
```

MODULE 4/11 Asks for the name of the card file to be opened, OPENs the file, checks the error channel to be certain that the file is available, LOADs the file, and CLOSEs it when ST does not equal zero.

```
400 REM **    4/11-LOAD CARD DATA    **
410 PRINT ST$"[clear][3x dwn][3x rht]ENTER
    CARD FILE NAME":INPUT"[4x
    rht]";FL$:FL$="0:"+FL$
420 OPEN 15,8,15
430 OPEN 2,8,2,FL$+",S,R":GOSUB 5200:IF
    A$="62" THEN 1010:REM * 62=FILE NOT
    FOUND
440 FOR X=1 TO 100:FOR Y=1 TO LI
450 IF X>1 AND ST<>0 THEN 470
460 INPUT#2,C$(X,Y)
465 IF RIGHT$(C$(X,Y),1)=CHR$(92)
    THEN 460
470 NEXT Y
480 NEXT X
490 CLOSE 2: CLOSE 15:RETURN
```

MODULE 4/12A Adds an asterisk to the end of any card line that is to be printed. Although it is described in the grocery list program, it has many other uses.

```
1100 REM ** 4/12A—ASTERISK ADDER **
1110 FOR N=1 TO 100:SG=3:IF C$(N,1)=""
     THEN 1000
1115 GOSUB 2030:IF Z$="*" THEN 1125
1120 NEXT N:GOTO 1000
1125 FOR X=3 TO WI:SP$=SP$+"[rht]":
     NEXT X
1127 PRINT ST$"[2x dwn][2x rht]"
     SP$ "[cmd/+]": L=2
1130 GET A$:IF A$="" THEN 1130
1133 IF A$=CHR$(13) THEN 1120
1135 IF A$="[dwn]" THEN PRINT
     "[up][2x rht]" SP$ "[spc]"
     :L=L+1:GOTO 1150
1140 IF A$="[up]" THEN PRINT
     "[up][2x rht]" SP$ "[spc]
     [2x up]":L=L-1:GOTO 1150
1145 IF A$="*" THEN PRINT"[up]"
     SP$ "*"
1146 IF A$="*" THEN C$(N,L)=LEFT$(C$
     (N,L),18)+"*":GOTO 1130
1150 PRINT SP$ "[2x rht][cmd/+]"
1160 GOTO 1130
```

MODULE 4/12B A menu for the grocery list program. This module includes the facility to add new items, to save the list, and to print the marked items.

```
1000 REM ** 4/12B-GROCERY LIST MENU **
1010 FLAG=0:PRINT ST$"[2x dwn]
     [2x rht][rvs] CHOOSE ONE
     [off]"
```

```
1020 PRINT:PRINT"[3x rht] 1=SCAN
     LIST":PRINT"[3x rht] 2=ADD
     TO LIST"
1025 PRINT"[3x rht] 3=PRINT MARKED
     LIST":PRINT"[3x rht] 4=SAVE
     LIST ITEMS
1030 INPUT "[3x dwn][10x rht]";Z$:IF VAL(Z$)<1
     OR VAL(Z$)>4 THEN 1000
1035 IF VAL(Z$)=4 THEN FL$="@0:GROCERY"
1040 ON VAL(Z$) GOTO 1100,1095,1050,5100
1050 OPEN 2,4:CMD 2:PRINT"[rvs]
     GROCERY LIST. [off][dwn]"
1060 FOR X=1 TO 100:FOR Y=1 TO 20:
     IF C$(X,Y)="" THEN 1090
1070 IF RIGHT$(C$(X,Y),1)<>"*" THEN 1080
1075 PRINT C$(X,Y)
1080 NEXT Y
1090 NEXT X:PRINT#2:CLOSE2:GOTO 1000
1095 FOR N=1 TO 100:IF C$(N,1)=""
     THEN 1097
1096 NEXT N:PRINT"NO CARDS":GOTO 1000
1097 GOSUB 550:C$(N,1)="[rvs] CARD
     #"+ STR$(N):PRINT ST$"[2x dwn]
     [2x rht]"C$(N,1):GOTO 4035
```

MODULE 4/13 Allows information to be added to cards.

```
1500 REM ** 4/13-ADD ON CARD COPY **
1510 FOR S=2 TO LI:IF C$(Y,S)=""
     THEN 1550
```

```
1520 IF RIGHT$(C$(Y,S),1)="£" THEN 1540
1530 NEXT S:GOTO 1000
1540 C$(Y,S)=LEFT$(C$(Y,S),LEN
     (C$(Y,S))-1)+"/":S=S+1
1550 PRINT ST$"[4x rht]RETURN=LINE
     END, £=PAGE END":PRINT ST$"
     [2x rht]";:N=Y
1560 FOR L=1 TO S:PRINT"[dwn]";:
     NEXT L
1570 FOR L=S TO LI:GOTO 4050:REM TO 4/5A
```

MODULE 4/14 Asks for today's date and whether or not this entry is an initial entry. If it is initial, the cards are dated from the date given. If not initial, the program loads data and goes to Module 4/15.

```
1700 REM ** 4/14-CALENDAR INITIAL **
1705 INPUT"[clear]ENTER TODAY's
     DATE (XX/XX)";MD$:
     MD=VAL(LEFT$(MD$,2))
1707 FL$="0:CALENDAR"
1710 Z$="":INPUT"NORMAL OR INITIAL?
     ([rev] N [off] /I)";Z$:IF
     Z$<>"I" THEN GOSUB 415:GOTO 1800
1713 DIM DY$(7):FOR X=1 TO 7:READ
     DY$(X):NEXT X
1715 INPUT"ENTER TODAY'S DAY NUMBER
     (1=MONDAY)";DY
1720 RD=VAL(RIGHT$(MD$,2)):Z=1
1725 FOR X=Z TO 100:MN=31
1730 IF MD=4 OR MD=6 OR MD=9 OR MD=11
     THEN MN=30
```

```
1735 IF MD=2 THEN MN=28
1740 FOR Y=RD TO MN
1745 C$(X,1)="[4x spc]"+STR$(MD)+"/"+
     RIGHT$(STR$(Y),2)+"[2x spc]"+DY$(DY)
1750 X=X+1:IF X=101 THEN 1780
1755 DY=DY+1:IF DY=8 THEN DY=1
1760 NEXT Y
1770 RD=1:X=X-1:MD=MD+1:IF MD=13
     THEN MD=1
1775 NEXT X
1780 GOTO 1000
1790 DATA"MONDAY-1","TUESDAY-2",
     "WEDNESDAY-3","THURSDAY-4"
1791 DATA"FRIDAY-5","SATURDAY-6",
     "SUNDAY-7"
```

MODULE 4/15 Takes the data entered as "today" and resets the calendar cards so that the first card is today. Doing this erases all cards prior to today. (If desired, this module can be made a part of Module 4/14.)

```
1800 REM ** 4/15-CALENDAR RESET **
1810 PRINT"[2x rht][rvs][2x spc]
     WAIT, RESETTING CALENDAR [2x spc]
     [off]"
1820 HL$="[4x spc]"+STR$(VAL(LEFT$
     (MD$,2)))+"/"+RIGHT$(STR$(VAL
     (RIGHT$(MD$,2))),2)
1825 IF LEFT$(C$(X,1),9)=HL$ THEN 1850
1830 NEXT X:PRINT"OUT OF RANGE":GOSUB
     500:IF Z$="*" THEN STOP
```

```
1840 GOTO 1000
1850 Z=1:FOR R=X TO 100:IF C$(R,1)="" THEN 1890
1860 FOR S=1 TO LI
1870 C$(Z,S)=C$(R,S):NEXT S
1880 Z=Z+1:IF Z=101 THEN 1000
1885 NEXT R:GOTO 1000
1890 DY=1+RIGHT$(C$(100,1)):MD=VAL(LEFT$(C$(100,2)))
1895 RD=1+VAL(MID$(C$(100,4,2))):GOTO 1725
```

MODULE 4/16 The menu for the calendar card file.

```
1900 REM ** 4/16-CALENDAR MENU    **
1910 FLAG=0:PRINT ST$ "[dwn][2x rht][rvs] CHOOSE ONE [off]"
1920 PRINT:PRINT"[4x rht] 1=FLIP THROUGH DATES":PRINT"[4x rht] 2=GO TO SPECIFIC DATE"
1925 PRINT"[4x rht] 3=FIND SPECIFIC DAYS"
1930 PRINT"[4x rht] 4=LOAD CARDS":PRINT"[4x rht] 5=SAVE CARDS & QUIT"
1940 INPUT"[dwn][10x rht]";Z$:IF VAL(Z$)=1 THEN 2000
1950 IF VAL(Z$)=2 THEN 1970
1955 IF VAL(Z$)=3 THEN FLAG=1:GOTO 1985
```

```
1960 IF VAL(Z$)=4 THEN FL$="0:CALENDAR":
     GOTO 415:REM ** TO LOAD MODULE **
1965 IF VAL(Z$)=5 THEN FL$="CALENDAR":
     GOTO 5040:REM ** TO SAVE MODULE **
1970 GOSUB 550:PRINT ST$:INPUT"[3x dwn]
     [2x rht] ENTER REQUIRED
     DATE (XX/XX)";Z$:Z$=LEFT$(Z$,5)
1975 F$=STR$(VAL(LEFT$(Z$,3)))+
     MID$(Z$,3,1)+RIGHT$(STR$(VAL
     (RIGHT$(Z$,2))),2)
1980 LN=LEN(F$):GOSUB 3200:IF Z$="*"
     THEN 1500
1981 GOTO 1000
1985 GOSUB 550:PRINT ST$:INPUT"([3x dwn]
     [2x rht] ENTER DAY NUMBER
     (1=MONDAY)";Z$
1990 F$=DY$(VAL(Z$)):GOTO 1980
```

MODULE 4/17 Asks for the principal amount, the annual interest (expressed as a decimal), and the time in days. It then prints the maturity value and interest paid.

```
400 REM ** 4/17-MATURITY VALUE  **
405 INPUT"ENTER PRINCIPAL";P
406 INPUT"ENTER ANNUAL INTEREST RATE
    (DECIMAL)";R
407 INPUT"ENTER TIME OF NOTE IN DAYS";T
410 I=P*R*T/365
420 MV=P+I
```

```
430 PRINT"MATURITY VALUE=";MV
440 PRINT"TOTAL INTEREST PAID=";I
```

MODULE 4/18 Calculates interest on a declining balance. Inputs principal, interest rate, monthly payments, and number of months for the calculation. Outputs total interest for the number of months, number of payments made, outstanding balance (if any), and amount of last payment made.

```
500 REM ** 4/18-MONTHLY PAYMENTS **
505 INPUT"ENTER PRINCIPAL";P
506 INPUT"ENTER ANNUAL INTEREST RATE
    (DECIMAL)";R
507 INPUT"ENTER NO. OF MONTHS TO PAY";N
508 INPUT"ENTER MONTHLY PAYMENT
    AMOUNT";S
510 I=0:LS=S
520 FOR X=1 to N:I=I+P*R/12:IF P<=S
    THEN 550
530 P=P+P*R/12-S
540 NEXT X:X=X-1:GOTO 560
550 LS=(INT(100*(P+P*R/12)+0.5))/100
560 PRINT:PRINT"NUMBER OF PAYMENTS
    MADE=";X
570 PRINT"AMOUNT OF LAST PAYMENT= $";LS
580 PRINT"TOTAL INTEREST PAID=$";I
590 PRINT"OUTSTANDING BALANCE= $";P
```

MODULE 4/19 Input principal, interest rate, time of note in days, discount rate, and discount period. Output is discount on note, maturity value of note, and present value of note.

```
600 REM ** 4/19-BANK DISCOUNT **
605 INPUT"ENTER PRINCIPAL";P
606 INPUT"ENTER ANNUAL INTEREST RATE
    (DECIMAL)";R
607 INPUT"ENTER TIME OF NOTE
    (IN DAYS)";T
608 INPUT"ENTER DISCOUNT RATE
    (DECIMAL)";R2
609 INPUT"ENTER DISCOUNT PERIOD IN
    DAYS";T2
610 MV=P+P*R*T/360
620 D=MV*R2*T2/360
630 PV=MV-D
650 PRINT:PRINT"DISCOUNT ON NOTE= $";D
660 PRINT"MATURITY VALUE OF NOTE=
    $";MV
670 PRINT"PRESENT VALUE OF NOTE=
    $";PV
```

MODULE 4/20 Calculates the maturity value and interest paid on a compound-interest note.

```
700 REM ** 4/20-COMPOUND INTEREST **
705 INPUT"ENTER PRINCIPAL";P
706 INPUT"ENTER ANNUAL INTEREST RATE
    (DECIMAL)";R
```

```
707 INPUT"ENTER NO. OF CONVERSIONS/
    YEAR";C
708 INPUT"ENTER NUMBER OF YEARS";N
710 MV=P*(1+R/C) ↑ INT(N*C)
720 I=MV-P
750 PRINT:PRINT"MATURITY VALUE OF
    NOTE = $";MY
760 PRINT"TOTAL INTEREST PAID= $";I
```

MODULE 4/21 Calculates the number of payments needed to pay off an interest-bearing note when the amount of payment is known.

```
100 REM ** 4/21-MONTHS TO PAY **
105 INPUT"ENTER PRINCIPAL";P
106 INPUT"ENTER ANNUAL INTEREST RATE
    (DECIMAL)";R
107 INPUT"ENTER MONTHLY PAYMENT
    AMOUNT";S
110 N=0
115 P=P+P*R/12:P=P-S:N=N+1
120 IF P<=S+1 THEN GOTO 140
130 GOTO 115
140 N=N+1:LS=P
150 PRINT:PRINT"NUMBER OF PAYMENTS
    TO MAKE=";N
570 PRINT"AMOUNT OF LAST PAYMENT= $";LS
```

MODULE 4/22 Calculates the monthly payment required to pay off a note when the number of payments is known.

```
200 REM ** 4/22-AMOUNT OF PAYMENT **
205 INPUT"ENTER PRINCIPAL";P
206 INPUT"ENTER ANNUAL INTEREST RATE
    (DECIMAL)";R
207 INPUT"ENTER NO. OF MONTHS TO PAY";N
210 R=R/12
220 V=1/(1+R)↑N
230 S=P*R/(1-V)
250 PRINT:PRINT"REQUIRED MONTHLY
    PAYMENT= $";S
```

MODULE 4/23 Calculates how much must be saved each month to reach a certain amount of savings in a certain amount of time.

```
300 REM ** 4/23-SINKING FUND **
305 INPUT"ENTER ANNUAL INTEREST RATE
    (DECIMAL)";R
306 INPUT"ENTER NO. OF MONTHS TO PAY";N
307 INPUT"ENTER AMOUNT TO BE
    ACCUMULATED";SF
310 D=SF*R/(12*(1+R/12)↑N-12)
350 PRINT:PRINT"REQUIRED MONTHLY
    PAYMENT= $";D
```

MODULE 4/24 Calculates the future value of a savings account that is being added to on a regular basis.

```
400 REM ** 4/24-FUTURE VALUE **
405 INPUT"ENTER ANNUAL INTEREST RATE
    (DECIMAL)";R
```

```
406 INPUT"ENTER NO. OF MONTHS TO PAY";N
407 INPUT"ENTER AMOUNT OF MONTHLY
    DEPOSIT";D
410 SF=12*D*((1+R/12)↑N-1)/R
450 PRINT:PRINT"VALUE OF FUND IN ";N;
    "MONTHS= $";SF
```

Chapter FIVE

BUSINESS

There are many uses in business for the card file modules covered in Chap. 4. With some ingenuity, they can be used to design ledger pages, tickler files, job ticket files, and any number of other business files which require retrieval of small units of data from a randomly stored set.

Similarly, the modules in Chap. 4 which calculate sinking-fund parameters, simple- and compound-interest loans, and so on, have many uses in business. The first few modules in this chapter continue to deal with calculations.

DEPRECIATION

The first three modules in this chapter represent three different methods for calculating depreciation. If desired, they can be combined into a single program. All that is required to combine these modules are an input, a menu, and an output. The input can be taken from lines X05

through X10 of any of the three modules. The menu can be either Module 3/14 or 3/15 from Chap. 3. The output can be line X90 from any of the three modules.

Module 5/1A calculates depreciation using a *straight-line* method. If the life expectancy is 10 years, the depreciation for each year is one-tenth of the difference between the cost and the salvage value. That is straight-line depreciation.

Module 5/1B uses a *sum-of-the-years-digits'* method of depreciation. This method calculates the depreciation by multiplying the depreciable value (cost less salvage value) by the ratio of remaining years to the sum (factorial) of the original years.

Finally, the *double-declining-balance* method of depreciation (Module 5/1C) uses a changing rate of depreciation based on a percentage of the balance. When the balance falls below the salvage value, the rate becomes zero.

TRENDS

Module 5/2 calculates a moving-average trend. The number of data points being used as the averaged group can be adjusted to broaden the smoothing effect of the averaging. Lines 760 through 790 print a horizontal graph of the averaged points. This simple module provides some very powerful information. In essence, each point in the output represents an average. This average is calculated using the number of points that we specify in line 705 as G. For example, if we use this module with monthly sales data and set G = 3, we are asking the computer to make a three-month average every month. The first average is for months 1, 2, and 3. The second average is for months 2, 3, and 4. The third average is for months 3, 4, and 5; and so

on. The module proceeds to calculate this moving average for each month furnished in the data array and stores the results in a new array, C(Q). The results are a series of values, each of which is an average sales trend for the month represented.

The more data points we include in each average (the bigger we make G), the less detail we are able to see and the fewer values we get. If we set G equal to N (the number of datum in our data array), we get a single value that is the average of all the data. The plotting routine given in lines 760 through 790 is useful but limited. There are other plotting routines that can easily be substituted.

CHECKBOOK PROGRAM

Modules that can be used to develop checkbook or budget programs require a listing and summing capacity that tends to be oriented toward business computing. The usefulness of the checkbook or budget programs, however, extend into the home and personal areas.

The first thing that a checkbook or budget program must be able to do is list items. To do this, the program must store the items in a string variable such as D$. We choose a string variable because we will want to store both numbers (such as the check number, date, and amount) and comments (such as to whom made, account, etc.). We will make D$ an array by dimensioning it to an appropriate size.

For a checkbook program, let's assume that we will not write more than 100 checks in a month (if that is not enough, we can increase the size of the array). We will dimension the array (in Module 5/3) to handle the 100 checks by using the initial line

```
20 DIM D$(100)
```

This line will set aside 100 string variables, numbered from D$(1) to D$(100). It actually sets aside 101 variables, because there is a D$(0), but for our purposes we will say that it sets aside 100. Each of these D$ variables will be able to hold all the information relating to a single check or to a single budget item.

In this module, line 25 dimensions another string variable, R$(100). This variable is used only if later we add reconciliation (Modules 5/8A, 5/8B, and 5/9) to the program. If reconciliation is not added, line 25 may be deleted from Module 5/3.

In the checkbook, a D$ variable would contain the check number, check date, payee, and amount of the check. In a budget program, the variable would contain much the same data, but instead of a check number, we might have a budget account number. In any case, the variables will act as the storage medium for the data. The data is entered using a fixed order and format.

In a checkbook program, the order and format might be something like this:

```
CHECK NO. DATE    PAYEE          AMOUNT
     1111 01/14   JOHN SMITH     0015.75
```

This is the string that will be entered into one of the D$ variable strings. If this is the first check that we enter, then,

```
D$(1)="1111  01/14  JOHN SMITH  0015.75"
      (---) ↑ (---)(-)(----------) ↑ (-------)
        4   1   5   4      10      1     7
```

As shown, the format that we have chosen for our data is a 4-position check number followed by a space, a 5-position

date followed by 4 spaces, a 10-position payee name followed by a space, and a 7-position amount. This format requires a total of 33 character spaces. We could have made the format tighter by not including the blank spaces between the elements of the data. Including the spaces, however, will let us use the data just as it is stored, without having to process it before printing.

Now that we have dimensioned an array and decided on a format, we need to consider what other functions will be needed. We will need an input module to insert data into our string array in the proper format. We will need a print module to list and sum our data. We will need to provide some method for correcting any mistakes that we may make (unless we never make any). Finally, we will need a way to save and load the data to and from a storage tape or disk.

Combining the modules we have just listed will provide us with a minimum checkbook program. This minimum program can duplicate the functions of our handwritten checkbook. It will accept our input, list our checks and deposits, print a running balance, and save and retrieve the list for later use. The modules required are:

3/14	Menu (modified for checkbook)
5/3A	Checkbook initial
5/3B	Bottom line (choices)
5/4	Load checks (modified 3/11)
5/5A	Input withdrawal
5/5B	Input deposit
5/5C	Delete
5/6A	Print check register
5/6B	Convert sum to SM$
5/7	Save file (modified 3/9)

The menu module will select the program function that we wish to use. The elements we will initially want to list in the menu are (1) list checks, (2) scroll to end of list, and (3) quit the program. The LOAD and SAVE functions are built into other modules, so they do not need to be listed separately.

If we use one of the menu modules from Chap. 3, we must change the line numbers to begin at 1000 and add a few lines to make it conform to the checkbook program. When printing the checkbook register, the checkbook program scrolls to the end of the checks if the Z$ from the menu input equals 11. The following lines include all the additions and changes needed to make Module 3/14 work with the checkbook program. Notice that only lines 1005 and 1090 are new.

```
1005 IF COPY=1 THEN RETURN
1011 M$(1)="LIST CHECKS":M$(2)="(SCROLL TO
     END)"
1012 M$(3)="QUIT PROGRAM":M$(4)=""
1025 N=3:PRINT"[clear]"
```

(other lines from Module 3/14)

```
1090 IF VAL(Z$)=2 THEN Z$="11":GOTO 2000
1095 ON VAL(Z$) GOTO 2000,2000,1097
1097 STOP
```

Loading Module 5/4 LOADs the file named "Ledger." Input Modules 5/5A and 5/5B are the core of the program. Module 5/6 (the list module) prints the checkbook and any input. When we have completed our checkbook work, we SAVE the file under the name "Ledger" (5/7). The other modules listed provide other needed functions, such as delete and other options.

After we have this much of our checkbook assembled and running, we will want to add a reconciliation module. Adding this module will easily allow us to check our records against the bank. The modules below will accomplish this reconciliation. We must also change the menu. Change the lines

```
1012 M$(3)="RECONCILE":M$(4)"QUIT
     PROGRAM"
1025 N=4:PRINT"[clear]"
```

(other line from Module 3/14)

```
1095 ON VAL(Z$) GOTO 2000,2000,5000,1097
```

Module 5/8B is not necessary but provides an opportunity to check the new balance forward before committing the results to the computer. Module 5/9 actually does the job and if desired, saves the record of the canceled checks. We also have the option to add the module that will retrieve the records of our canceled checks (5/10).

5/8A	Reconcile
5/8B	Test reconcile
5/9	Cancel and save
5/10	Retrieve canceled checks

Finally, we can add the "accounts" modules, if applicable. These modules will help us to track more than one checkbook on the same account. The modules do this by separating the checks into subaccounts by check number. The accounts menu module is simply one of our old friends from Chap. 3. The line numbers of this menu should begin at 3000. Because the line numbers should be in the 3000s, Module 3/14 may be easier to use. In either case, there are only four choices to be made: account

#1 = 1, account #2 = 2, account #3 = 3, and deposit = 4. If desired, the account menu can be changed to print names rather than account numbers. The account DATA line must contain the check number limits for the different accounts. Of course, there can be no overlap in these numbers. One addition to the menu module must be made. The last line should be

```
3090 Z=VAL(Z$)
```

The two modules to be added are

```
3/14  Accounts menu
5/12  Accounts program and data
```

Of course, we will need to change the main menu (back at lines 1000, etc.) to add this function to those functions already available. This change is made as follows:

```
1012 M$(3)="RECONCILE":M$(4)"ACCOUNTS"
1013 M$(5)="QUIT PROGRAM":M$(6)=""
1025 N=5:PRINT"[clear]"
```

(other lines from Module 3/14)

```
1095 ON VAL(Z$) GOTO 2000,2000,5000,3000,1097
```

Finally, we can add Printer Module 5/12 to print the check register and the accounts registers. This, too, will require changing the main menu, as follows:

```
1013 M$(5)="PRINT":M$(6)="QUIT PROGRAM"
1025 N=6:PRINT"[clear]"
```

(other lines from Module 3/14)

```
1093 IF VAL(Z$)=5 THEN Z$="20":GOTO 1200
```

```
1095 ON VAL(Z$) GOTO 2000,2000,5000,3000,1200,
     1097
```

With this printer module added to the program, we have a fully functional checkbook program and have many of the modules necessary to build a budget program.

BUDGET PROGRAM

The budget program is designed to work with the checkbook program. Using your ingenuity, the modules in these two programs can certainly be combined in other configurations for other uses. Here we will assemble the modules into a program that you can follow and use.

The budget program will take the check file, called "Ledger," from the checkbook program and allow you to assign each check in that file to a budget account. It will also allow you to divide a check between two or more accounts. It will then save the budget file under any file name you choose. Later, you can retrieve the budget file, divide the entries by accounts, and print out the accounts.

To do these things, we must assemble the following modules:

5/13	Budget initial
5/4	Read disk (modified)
5/14	Assign category
5/15A	Divide accounts
5/15B	Convert V to V$ (modified 5/6B)
5/16	List array
5/17	Write data to storage (modified 3/7)
5/18	List from storage (modified 3/13)
5/19	Print accounts
5/20	Manual scroll

Module 5/13 dimensions the string array and asks the initial questions. The answers to the questions determines to which module the program will go next. If desired, the questions could be replaced by a menu. If the choice is to assign account numbers to the checks, the computer is sent to Module 5/4 (read disk) and then to Module 5/14 (assign category). If the choice is to print, we are asked for a file name. If ITEMS are to be printed, the computer is directed to Module 5/18 (list from storage). If ACCOUNTS are to be printed, Module 5/19 (print accounts) is used.

Modules 5/17 and 5/18 are both derivatives of modules given in Chap. 3. The modifications made to Module 3/7 in order to produce Module 5/17 are quite minor, whereas the changes are somewhat greater to change Module 3/13 into Module 5/19. Although the modifications may be of interest for the purpose of learning how other modifications may be made for other uses, Chap. 5's modules can be used as shown.

Module 5/14 is the pivotal module of this budget program, and modifications of this module can be of use in many other programs that you may design. Initially, the module appears somewhat like a menu (lines 1000 through 1090), in that it is a list of items to be selected by entering a number. The remainder of the module, however, presents a check to be assigned to an account. Lines 1110 and 1120 print the check in reversed type, and line 1130 asks for an account number. The account number, name of the check payee, and the check amount are stored in the array as D$(X,1), where X is the check's position in the checkbook listing. All checks are stored as D$(X,1). Those checks that are not divided among more than one account have no other entry. Those that are divided have the additional entries stored as D$(X,2), D$(X,3), D$(X,4), and so on.

The accounts are listed in the portion of the module that we said looked like a menu. Of course, the names of the accounts may be changed to meet your specific needs. To do this, simply insert the names of your accounts in place of the names that are given in this book. If you like neatness, you can also change the spaces between the new names so that the new names fall into columns. There are two names that should not be changed. These are 15 = DIVIDE and E = ERROR. These two names have special functions which are built into the module.

Entering E as an account number allows an error to be corrected. The computer jumps to line 1180 of the module program. Here the program is pushed back to the previous entry so that corrections can be made to that entry.

Entering 15 will allow us to divide a check amount into more than one account. This feature might be used, for example, with the check paying a subcontractor. The single check might be paying for material, labor, travel, and insurance. To divide the entry into the different accounts, the module calls for the subroutine of Module 5/15A. Here the division is made and the program returns to Module 5/14.

As we have said, the check is listed as D$(X,1), so Module 5/15A begins to list its divisions at D$(X,2). A maximum of eight divisions can be made. There are only 10 places available [D$(X,1) through D$(X,10)]. The first of the 10 places is taken by the check itself and the last used place must be a zero so that the computer can recognize when it has reached the end of the divisions.

The total amount of the check is displayed and we are asked to "INPUT AMOUNT TO BE ASSIGNED". When we enter the amount, it is displayed together with the request to "INPUT CATEGORY FOR THAT AMOUNT". The amount assigned is stored in the array

and is subtracted from the total amount of the check. If there is still a positive difference between the check and the divided amounts, the computer asks for another amount to be assigned. If the sum of the divided amounts equals or exceeds the amount of the check, control is returned from Module 5/15A to Module 5/14.

If we do not wish to allow the amounts of the divisions to exceed the amount of the check, we can change line 230 to read

```
230 INPUT"[2x dwn]AMOUNT";V:AD=AD+V:
    IF VAL(V$)<V THEN AD=AD+VAL(V$):
    GOSUB 400:REM 5/15B
```

When all the accounts have been entered for a divided check, Q is entered to indicate that we are ready to QUIT. When this is done, the computer jumps to line 290, marks the end of the divisions by assigning a zero to the next element of the array, and returns to Main Program Module 5/14.

Module 5/15B is very similar to Module 5/6B. Module 5/6B is changed to fit the specific needs of this program, and is renumbered 5/15B. If, in Module 5/6B, we substitute V for SUM and V$ for SM$, the similarity becomes more obvious. The primary difference is that Module 5/15B will round values off upward to two places.

The remaining modules in this chapter are for listing, saving, or loading. Module 5/16 lists a two-dimensional array but does so in the unusual manner required by the budget program. It prints only the "column 1" element of each "row X" in the array [D$(X,1)] unless the contents of that row X/column 1 element is 15. If the element is 15, the element is not printed; instead, the program jumps to line 1270 and prints the additional column elements of

row X until an element is reached whose contents are 0. The program then jumps back to line 1250 and continues to print the first column elements of the row X's that follow.

If there are more than 24 items to be printed (which would cause the first items to be scrolled off the screen), then instead of scrolling the first items off the screen, the program goes to the subroutine in Module 5/20. Module 5/20 holds the printing and allows us to scroll one line at a time, manually.

Of course, Module 5/16 can be modified for other uses. One such modification might be to change the 15 in line 1230 to some other "flag" that suits our needs. Similarly, the 0 in line 1270 can be changed if some other flag would fit our needs better.

Module 5/17 SAVEs the two-dimensional array on disk or tape. Module 5/17 is derived from Module 3/7 and differs only because all the elements of the array do not need to be saved.

Module 5/18 prints all the items in storage. It is the opposite of the save module (5/16). The module OPENs the file, INPUT#s each stored item, PRINTs each item, and when the end of data is found, CLOSEs the file.

Module 5/19 operates exactly like Module 5/18 except that it prints only the items found in a single chosen account. Initially, the module program asks for an account number (using the account listing of Module 5/14 as a subroutine). Then the program proceeds to operate just like Module 5/18. It OPENs the file and INPUT#s each item, but only if the item account number matches the chosen account will the item be PRINTed. The program continues to INPUT#, check against the chosen account, print or not print, depending on the match, and when the data is exhausted, the program CLOSEs the file. In the

event that more items are to be printed than there is space on the screen, the program will jump to the Module 5/20 subroutine to provide manual scrolling.

Finally, Module 5/20 is the manual scroll subroutine used with previous modules. It PEEKs into the computer's display file to determine whether or not the next line printed will be on the bottom line of the screen. If so, Module 5/20 pops a manual scroll input line on the bottom line of the screen. If the RETURN is entered to scroll, the input line is erased, the data line printed, the screen is scrolled up one line, and a new manual scroll input line is printed on the bottom of the screen. This module can be very useful and can be used in any program where a manual scroll is desired.

CONCLUSION

There are many more programs that would be useful to business. Here we have attempted to list a diverse, yet useful set of modules and at the same time provide a few complete programs.

It should be noted that many of the modules in this chapter can have many other uses, even though they are designed to be used in a specific program. The primary caution is to make certain, when using a module in another program, that all GOTOs and GOSUBs are necessary in the new program. If they are, make certain that the GOTO or GOSUB line exists in the new program.

Do not be afraid to throw together a group of modules that seem to do what you want. Usually, after the modules are assembled, a quick look to avoid obvious errors will take care of 70 percent of the problems of combining modules. Problems such as using different variable names

when they should be the same, or sending the program to a GOTO line that is not there, can usually be avoided with a little effort. After this initial look, a little trial and error will usually get us to a usable program.

The next chapter combines new modules and modules from Chap. 4 with modules from Chap. 5 to produce payroll and other business computations and records.

CHAPTER FIVE

MODULES

MODULE 5/1A Calculates depreciation by the straight-line method. The salvage value of the item can be any value between zero and the cost value. The year to be calculated must be no greater than the life expectancy of the item.

```
400 REM ** 5/1A-STRAIGHT-LINE DEPREC **
405 INPUT"[clear][3x dwn]
    ENTER ITEM COST";C:INPUT"ENTER ITEM
    SALVAGE VALUE";S
406 IF S>C THEN GOTO 405
407 INPUT"ENTER LIFE EXPECTANCY";L:INPUT
    "ENTER YEAR OF LIFE TO DEPRECIATE";Y
408 IF Y>L THEN GOTO 405
410 DS=(C-S)/L
420 B=C-(DS*Y)
490 PRINT"DEPRECIATION FOR YEAR = ";DS:
    PRINT"BOOK VALUE AT END OF YEAR = ";B
```

MODULE 5/1B Calculates depreciation using the sum-of-the-years-digits' method. Input and output parameters are the same as for Module 5/1A.

```
500 REM ** 5/1B-SUM-OF-YEARS DEPREC **
505 INPUT"[clear][3x dwn]
    ENTER ITEM COST";C:INPUT"ENTER ITEM
    SALVAGE VALUE";S
506 IF S>C THEN GOTO 405
```

```
507 INPUT"ENTER LIFE EXPECTANCY";L:INPUT
    "ENTER YEAR OF LIFE TO DEPRECIATE";Y
508 IF Y>L THEN GOTO 405
510 X=0:B=C
520 FOR T=1 TO L
530 X=X+T:NEXT T
540 FOR T=1 TO Y
550 DY=(C-S)*(L-T+1)/X
560 B=B-DY
570 NEXT T
590 PRINT"DEPRECIATION FOR YEAR = ";DY:
    PRINT"BOOK VALUE AT END OF YEAR = ";B
```

MODULE 5/1C Calculates depreciation using the double-declining-balance method. Input and output parameters are the same as for Module 5/1A.

```
600 REM ** 5/1C-DBL DECLINE BALANCE **
605 INPUT"[clear][3x dwn]
    ENTER ITEM COST";C:INPUT"ENTER ITEM
    SALVAGE VALUE";S
606 IF S>C THEN GOTO 405
607 INPUT"ENTER LIFE EXPECTANCY";L:INPUT
    "ENTER YEAR OF LIFE TO DEPRECIATE";Y
608 IF Y>L THEN GOTO 405
610 R=1/L:B=C
620 FOR T=1 TO Y
630 DD=B*R:IF (B-DD)<=S THEN 660
640 B=B-DD
650 NEXT T:GOTO 690
```

```
660 DD=B-S:B=S
670 GOTO 650
690 PRINT"DEPRECIATION FOR YEAR = ";DD:
    PRINT"BOOK VALUE AT END OF YEAR = ";B
```

MODULE 5/2 Calculates a moving-average trend. Requires that the data be in an array [D(X)] and the number of items in the array be given (N). Outputs an array of the moving averages, the number of averages in the array, and the maximum value in the array. Lines 760 through 790 print a graph of the trend averages.

```
700 REM **    5/2-TRENDS     **
705 INPUT"[clear][3x dwn]ENTER
    THE SIZE OF THE AVERAGE GROUP";G
710 MAX=0:P=N-G+1:DIM C(P)
720 FOR Q=1 TO P:B=0
725 FOR R=1 TO G
730 B=B+D(R+Q-1):NEXT R
735 C(Q)=B/G
740 IF C(Q) > MAX THEN MAX=C(Q)
750 NEXT Q:PRINT"[clear]"
760 FOR X=1 TO P:FOR Y=1 TO MAX
770 IF C(X)<Y THEN 790
780 PRINT"X";:NEXT Y
790 PRINT:NEXT X:STOP
```

MODULE 5/3A The initial module for the checkbook program, which dimensions the string arrays and goes to line 1000 (Menu Module 3/14).

```
10 REM ** 5/3A-CHECKBOOK INITIAL **
20 DIM D$(100)
25 DIM R$(100)
30 SCR=1064:FLAG=0
40 INPUT"[clear][3x dwn]
   ENTER TODAY'S DATE";TD$
90 GOTO 1000
```

MODULE 5/3B A subroutine that adds a bottom line to the printed list of checks. This line provides four options for adding to, deleting from, or exiting the list.

```
2200 REM ** 5/3B-BOTTOM LINE **
2210 IF FLAG=2 THEN PRINT:RETURN
2220 IF I<20 THEN FOR W=1 TO 20-I:
     PRINT"[dwn]";:NEXT W
2230 PRINT"ENTER:1=CHEK,2=DEPOSIT,
     3=DELETE,4=STOP"
2240 INPUT Z:IF Z<1 OR Z>4 OR Z=VAL(
     CHR$(13)) THEN PRINT"[up]";:
     GOTO 2240
2250 PRINT"[2x up][39x spc]":
     PRINT"[15x spc]"
2260 ON Z GOTO 200,300,500,2500
2270 GOTO 2240
```

MODULE 5/4 A subroutine that loads the "Ledger" file from the disk (or tape) and returns to the main program.

```
1500 REM **        5/4-LOAD DATA        **
1501 REM **  DR=8 (DISK);DR=1(TAPE) **
```

```
1505 DR=8
1510 OPEN2,DR,2,"0:LEDGER,S,R"
1520 FOR I=1 TO 100
1530 INPUT#2,D$(I)
1540 IF D$(I)="0" THEN 1570
1560 NEXT I
1570 CLOSE 2:FLAG=1:RETURN
```

MODULE 5/5A A subroutine to input checkbook withdrawals into the ledger format. The main body of this program is also used by the deposit module.

```
200 REM ** 5/5A-INPUT WITHDRAWALS **
202 N=1
205 NUM=I+1
210 FOR W=1 TO 23-NUM:PRINT"[up]";
    NEXT W
215 IF I>20 THEN PRINT"[up]";
220 PRINT D$(I);:PRINT"[cmd/+]
    [lft]";
225 FOR J=N TO 30
230 GET A$:IF A$="" THEN 230
235 IF J=24 AND A$="0" THEN A$="[spc]"
237 IF ((J=25 OR J=26) AND
    RIGHT$(D$(I),1)="[spc]") AND A$="0"
    THEN A$="[spc]"
240 PRINT A$;:IF J<31 THEN
    PRINT"[cmd/+][lft]";
245 D$(I)=D$(I)+A$
```

```
250 IF J=4 OR J=10 THEN GOSUB 295:
    GOTO 240
255 IF J=22 AND N=1 THEN A$="-":J=J+1:
    GOTO 240
260 IF J=22 AND N=6 THEN A$="[spc]":
    J=J+1:GOTO 240
265 IF J=7 THEN A$="/":J=J+1:GOTO 240
270 IF J=11 AND I<10 THEN PRINT
    "0[lft]";I;"[cmd/+][lft]";:
    D$(I)=D$(I)+"[3x spc]":J=13:GOTO 285
275 IF J=11 AND I>9 THEN PRINT
    [lft]";I;"[cmd/+][lft]";:
    D$(I)=D$(I)+"[3x spc]":J=13:GOTO 285
280 IF J=27 THEN A$=".":J=J+1:GOTO 240
285 NEXT J
290 GOTO 2030
295 J=J+1:A$="[spc]":RETURN
```

MODULE 5/5B A routine for the input of checkbook deposits. This routine uses much of Module 5/5A to accomplish its purpose.

```
300 REM ** 5/5B-INPUT DEPOSITS **
310 N=6
320 D$(I)=">DEP"
330 GOTO 205:REM ** GOTO MOD 5/5A **
```

MODULE 5/5C The delete subroutine, which deletes an entire line of the checkbook ledger and moves succeeding lines up to fill the space.

```
500 REM ** 5/5C-DELETE LINE **
510 INPUT"[up] ENTER LINE TO
    BE DELETED";D%
520 IF D$(D%)="" THEN 550
530 FOR F=D% TO 99:D$(F)=D(F+1):NEXT F
540 PRINT"[up]"
550 Z$="11":GOTO 2005
```

MODULE 5/6A Prints the checkbook ledger page. If the menu choice was 1 (Z$ = "1"), each item is scrolled onto the page manually. If Z$ = "11," the page automatically scrolls to the end of the data. To function properly, this module requires Module 5/6B.

```
2000 REM ** 5/6A-PRINT CHECK REGISTER **
2003 PRINT"[clear]":IF FLAG=0 THEN
     GOSUB 1500:REM TO 5/4 **
2005 IF COPY=1 THEN CMD3:REM *WITH 5/13
2010 SUM=0:GOSUB 2080
2015 FOR I=1 TO 100:IF D$(I)="" THEN
     2200:REM * TO 5/3B *
2020 PRINT LEFT$(D$(I),11);:IF I>9
     THEN PRINT"[lft]";
2025 PRINT I;"[lft]";:IF
     MID$(D$(I),13,1)="*" THEN
     PRINT"[2x lft]*";
2030 PRINT RIGHT$(D$(I),18);
2035 SUM=SUM+VAL(MID$(D$(I),24,8)):
     SUM=(INT(SUM*100+0.5))/100
2040 GOSUB 2300:REM * TO 5/6B *
```

```
2045 IF LEFT$(SM$,1)="-" THEN PRINT
     "[rvs]";SM$;"[off]":GOTO 2055
2050 PRINT SM$
2055 IF Z$="11" THEN 2075
2060 S$=""
2065 IF I>19 THEN INPUT"ENTER RETURN TO
     SCROLL";S$:IF S$="" THEN PRINT
     "[up]";:GOTO 2075
2070 IF I>19 AND S$<>"" THEN 1000
2075 NEXT I:STOP
2080 REM ** FORMAT SUBROUTINE **
2085 PRINT"[clear] CKNO [2x spc]
     DATE [2x spc] N [3x spc] PAYEE
     [4x spc] AMOUNT [2x spc] BALANCE";
2090 PRINT"[40x cmd/P]":PRINT
     "[up]";
2095 RETURN
```

MODULE 5/6B A subroutine that converts SUM to the string variable SM$. It also adds any zeros needed following the decimal point.

```
2300 REM ** 5/6B-SUM TO SM$ CONVERT **
2310 IF SUM=0 THEN SM$="[4x spc] 0.00":
     RETURN
2320 SM$=STR$(SUM)
2330 IF LEN(SM$) < 8 THEN
     SM$=LEFT$(SM$,1)+"[spc]"+
     RIGHT$(SM$,LEN(SM$)-1):GOTO 2330
2340 IF INT(SUM)=SUM THEN SM$=LEFT$
     (SM$,1)+MID$(SM$,5,4)+".00":RETURN
```

```
2350 IF INT(10*SUM+.05)/10=SUM THEN
     SM$=LEFT$(SM$,1)+MID$(SM$,3,6)+"0"
2360 IF LEN(SM$)<>8 THEN PRINT"ERROR
     IN SM$ LEN":STOP
2370 RETURN
```

MODULE 5/7 The same as Module 3/9 except that some of the variables in Module 3/9 have been made constants in Module 5/7.

```
2500 REM ** 5/7-SAVE DATA **
2510 OPEN 2,8,2,"@0:LEDGER,S,W"
2520 FOR I=1 TO 100
2530 IF D$(I)="" THEN CLOSE 2:GOTO 1000
2540 PRINT#2,D$(I)
2550 NEXT I
2560 CLOSE 2:PRINT"DATA FILLED":STOP
```

MODULE 5/8A A rather large module that asks for a check number (or other token) and searches the check register for that item. It displays the amount, and if an "*" is entered, it asks for another check number. A deposit is recognized by the token ">DEP" and a withdrawal made from a card machine is recognized by the token "CCRD." The CCRD must be used as a check number when the withdrawal is entered into the check register.

```
5000 REM ** 5/8A-RECONCILE TO BANK **
5010 PRINT"[clear]":GOSUB 1500
5015 PRINT"[clear][rev] ENTER
     CHECK NUMBER[off]
```

```
5020 PRINT,,,"* = STOP":C$=""
5025 FOR I=1 TO 4
5030 GET A$:IF A$="" THEN 5030
5035 IF A$="*" THEN GOTO 4300:REM 5/8B
5040 PRINT A$;:C$=C$+A$:NEXT I:
     PRINT"[spc]=[spc]";
5045 FOR I=1 TO 100
5050 IF LEFT$(D$(I),4)=C$ AND
     (C$=">DEP" OR C$="CCRD") AND
     MID$(D$(I),13,1)="*" THEN 5060
5055 IF LEFT$(D$(I),4)=C$ THEN
     PRINT MID$(D$(I),13,1);
     RIGHT(D$(I),8):GOTO 5065
5060 NEXT I
5065 PRINT"[5x rht][rvs] IF
     CORRECT ENTER * [off]"
5070 GET Z$:IF Z$="" THEN 5070
5075 IF Z$="*" THEN D$(I)=LEFT$(D$(I),
     12)+"*"+RIGHT$(D(I),18):GOTO 5015
5080 Z$="Y":INPUT"ERROR ? ([rvs]
     Y [off]/N)";Z$:IF Z$="N"
     THEN 5015
5085 INPUT"ENTER CORRECT VALUE
     (0000.00)";VA$
5090 IF LEFT$(VA$,3)="000" THEN
     VA$="[3x spc]"+RIGHT$(VA$,4)
5093 IF LEFT$(VA$,2)="00" THEN
     VA$="[2x spc]"+RIGHT$(VA$,5)
5095 IF LEFT$(VA$,1)="0" THEN
     VA$="[spc]"+RIGHT$(VA$,6)
```

```
5100 VA$="-"+VA$
5110 IF LEN(VA$) <> 8 THEN 5085
5120 D$(I)=(LEFT$(D$(I),12)+"[spc]"+
     MID$(D$(I),14,9)+VA$
5130 PRINT:PRINT C$ "[spc]=[spc]";:
     GOTO 5045
```

MODULE 5/8B After the canceled checks have been marked with an asterisk in Module 5/8A, Module 5/8B can give a trial balance forward. The value given in line 4360 as F$ should match the balance given on the bank statement.

```
4300 REM ** 5/8B-TRIAL BALANCE FWD **
4310 X=1:SUM=0
4315 FOR I=2 TO 100:IF D$(I)=""
     THEN 4335
4320 IF MID$(D$(I),13,1,) <> "*" THEN
     R$(X)=D$(I):X=X+1
4325 IF MID$(D$(I),13,1,)="*" THEN
     SUM=SUM+VAL(RIGHT$(D$(I),8))
4330 NEXT I
4335 FWD=VAL(MID$(D$(1),24,8)):
     FWD=FWD+SUM
4340 FWD=INT(100*FWD+0.5)/100
4345 F$=STR$(FWD)
4350 IF LEN(F$) < 8 THEN F$=LEFT$(F$,1)
     +"[spc]"+RIGHT$(F$,LEN(F$)-1)
4355 R$(1)=">FWD [spc] 00/00 [4x spc]
     BAL. FWD."+F$
```

```
4360 PRINT "[clear][3x dwn]"
     R$(1)"[2x dwn]":GOTO 4500
```

MODULE 5/9 After the trial balance forward has been run, then if the trial is approved, the actual reconciliation is done by this module.

```
4500 REM **    5/9-NEW BALANCE FWD     **
4501 REM ** & SAVE CANCELED CHECKS  **
4510 Z$="Y":INPUT"READY TO RECONCILE ?
     ([rvs] Y [off] /N)";Z$:IF
     Z$="N" THEN 4700
4520 INPUT"ENTER CHECK FILE DATE";DT$
4530 X=2:SUM=0
4540 OPEN 5,8,5,"0:"+DT$+"CHECKS,S,W"
4550 FOR I=2 TO 100:IF D$(I)=""
     THEN 4600
4560 IF MID$(D$(I),13,1)<>"*" THEN
     R$(X)=D$(I):X=X+1
4570 IF MID$(D$(I),13,1)="*" THEN
     PRINT#5,D$(I)
4580 IF MID$(D$(I),13,1)="*" THEN
     SUM=SUM+VAL(RIGHT$(D$(I),8))
4585 D$(I)=""
4590 NEXT I
4600 FWD=VAL(MID$(D$(1),24,8)):
     FWD=FWD+SUM
4610 FWD=INT(100*FWD+0.5)/100
4620 F$=STR$(FWD)
```

```
4630 IF LEN(F$) < 8 THEN F$=LEFT$(F$,1)
     +"[spc]"+RIGHT$(F$,LEN(F$)-1)
4640 D$(1)=">FWD [spc] 00/00 [4x spc]
     BAL. FWD."+F$
4650 FOR I=2 to 100:D$(I)=R$(I):
     NEXT I:CLOSE 5
4660 GOTO 2500:REM 5/7
4700 INPUT"[dwn] VOID RECONCILE ?
     (Y/N)";Z$
4710 IF Z$="N" THEN 2005:REM 5/6A
4720 IF Z$="Y" THEN RUN
4730 GOTO 4700
```

MODULE 5/10 Loads filed canceled checks into the computer for review. The checks have been filed by dates given to them in Module 5/9.

```
4000 REM ** 5/10-RETR X-ED CHECKS **
4010 INPUT"[clear] ENTER DATE OF
     FILED CHECKS";DT$
4020 OPEN 5,8,5,"0:"+DT$+"CHECKS,S,R"
4030 FOR I=1 TO 100
4040 INPUT #5,D$(I)
4050 IF D$(I)="0" THEN CLOSE 5:GOTO 4070
4060 NEXT I
4070 FLAG=2:Z$="1":GOSUB 2005
4080 INPUT"[dwn][3x rht]
     [rvs] RETURN [off] TO
     STOP,'R' TO REPEAT";Z$:IF
     Z$="R" THEN 4070
4090 FLAG=0:GOTO 1000
```

MODULE 5/11 The complete system for dividing the checks into different accounts by check number. The DATA line at the end of the module must be changed to include the check number limits. The first two numbers in the DATA line must be the minimum and maximum check numbers of the account having the lowest check numbers. If desired, the minimum can be 0. The next two numbers in the DATA line are the maximum and minimum of the account with the middle-sized numbers. Finally, the last two positions in the DATA line are for the "master" account. This set of check numbers should be the largest numbers of the three accounts and will be the only account that can accept nonnumeric check numbers. These nonnumeric numbers include CCRD for card withdrawals and any other mnemonic that you may desire for nonstandard withdrawals such as INS1 or LOAN for automatic check withdrawals. The maximum number for this third account should remain as shown (ZZZZ).

```
3100 REM **          5/11-ACCOUNTS          **
3101 REM **  DATA LINE NEEDS SETTING **
3110 FOR I=1 TO 3:READ L$(I):READ H$(I):
     NEXT I:RESTORE
3115 IF FLAG=1 THEN PRINT
     "[clear]":GOTO 3125
3120 GOSUB 1500:PRINT"[clear]"
3125 LO$=L$(Z):HI$=H$(Z)
3130 IF Z=5 THEN 1000
3135 IF COPY=1 THEN CMD 3
3140 PRINT"[7x rht][rvs][spc]
     ACCOUNT NUMBER";Z;"[lft]
     [off]":SUM=0
```

```
3145 FOR I=1 TO 100:IF D$(I)="" AND
     COPY=1 THEN 3210
3150 IF D$(I)="" THEN 3200
3155 IF Z=3 AND LEFT$(D$(I),1)=">"
     THEN 3190
3160 IF Z=4 AND LEFT$(D$(I),1) <>">"
     THEN 3190
3165 IF Z=4 AND LEFT$(D$(I),1)=">"
     THEN PRINT D$(I);:GOTO 3180
3170 IF LEFT$(D$(I),4) > LO$ AND
     LEFT$(D$(I),4) < HI$ THEN
     PRINT D$(I);
3175 IF LEFT$(D$(I),4) < LO$ OR
     LEFT$(D$(I),4) > HI$ THEN 3190
3180 LET SUM=SUM+VAL(MID$(D$(I),24,8)):
     SUM=(INT(SUM*100+0.5))/100
3185 GOSUB 2300:PRINT SM$:IF COPY=1
     THEN 3195
3190 IF PEEK(209)+255*PEEK(210) > 1945
     THEN INPUT"RETURN TO SCROLL";Q$:
     PRINT"[up]";
3195 NEXT I
3200 Z$="":INPUT"[3x spc] INPUT [rvs]
     RETURN[off] TO CONTINUE";Z$:
     IF Z$ <> "" THEN GOTO 1000
3210 Z=Z+1:PRINT"[up]";:GOTO 3125
3220 DATA "0000";"001","1000","1001","2000",
     "ZZZZ"
```

MODULE 5/12 Prints the checkbook ledger page on a Commodore printer. If the accounts modules are not be-

ing used in the program, use only lines 1200, 1220, 1230, and 1260.

```
1200 REM ** 5/12-PRINTER ROUTINE **
1210 INPUT"[clear][4x dwn]
     [4x rht] LIST OR ACCOUNTS (L/A)"
     ;R$:IF R$="A" THEN 1250
1220 COPY=1
1230 OPEN 3,4:PRINT#3,DT$:Z$="11":
     GOSUB 2000:PRINT#3:CLOSE 3:COPY=0
1240 INPUT"[clear][4x dwn]
     [4x rht] ACCOUNTS (Y/N)";R$:
     IF R$="N" THEN 1260
1250 OPEN 3,4:PRINT#3,DT$:Z=1:COPY=1
     GOSUB 3100:PRINT#3:CLOSE 3:COPY=0
1260 GOTO 1000
```

MODULE 5/13 Sets up the initial conditions for the budget program and asks the initial questions.

```
01 REM **  BUDGET PROGRAM  **
02 REM **  5/13-INITIAL MODULE  **
10 DIM D$(100,10):S$="[39x spc]"
50 Z$="A":INPUT"[clear] ASSIGN TO
   ACCOUNTS OR PRINT? ([rev] A
   [off]/P)";Z$:IF Z$="P" THEN 70
55 IF Z$="A" THEN Z$="":GOSUB 500:
   REM 5/4 MODIFIED:GOTO 1000:REM 5/14
60 GOTO 50
70 Z$="I":INPUT" [clear] PRINT
   ITEMS OR PRINT ACCOUNTS? ([rev]
   I [off]/A";Z$
```

```
72 IF F$="" THEN F$="NOT AVAILABLE"
73 PRINT"[3x dwn] CURRENT FILE
   NAME IS "; F$
75 INPUT"[dwn] ENTER FILE NAME";F$
80 IF Z$="I" THEN 2000:REM 5/18
85 IF Z$="A" THEN 3000:REM 5/19
90 GOTO 50
```

MODULE 5/14 Lists all the options that may be used for budget accounts. The account names may be changed, but "DIVIDE" must remain number 15 and "QUIT" and "ERROR" may not be changed.

```
1000 REM ** 5/14-ASSIGN CATEGORY **
1010 PRINT"[clear][3x rht]
     [rvs] ASSIGN TO ONE OF THE
     FOLLOWING [off]"
1020 PRINT"[dwn][rht]
     1=PAYROLL [4x rht] 6=MEDICAL
     [2x rht] 11=FURNITURE"
1030 PRINT"[dwn][rht]
     2=INSURANCE [2x rht] 7=RENT
     [5x rht] 12=PETTY CASH"
1040 PRINT"[dwn][rht]
     3=UTILITIES [2x rht] 8=OFFICE
     [3x rht] 13=POSTAGE"
1050 PRINT"[dwn][rht]
     4=ENTERTAIN [2x rht] 9=TAXES
     [4x rht] 14=OTHER"
1060 PRINT"[dwn][rht]
     5=INTEREST [3x rht] 10=AUTO
     [4x rht] 15=DIVIDE"
```

```
1065 PRINT"[6x rht] ****
     [2x rht][rvs] DEPOSITS
     [off][2x rht]****"
1070 PRINT"[2x rht] I=INVOICES
     [2x rht] C=CASH [2x rht]
     O=OTHER"
1080 IF Z$="A" THEN RETURN
1090 PRINT"[dwn][8x rht] Q=QUIT
     [2x rht] E=ERROR[dwn]"
1100 FOR X=1 TO 100
1110 IF D$(X,1)="" THEN 1200:REM TO 5/16
1120 PRINT"[rvs]";S$,"[4x spc]";
     D$(X,1);"[4x spc]",S$;"[off]"
1130 INPUT"[dwn]CATEGORY";A$:
     IF A$="Q" THEN 1200:REM TO 5/16
1133 IF A$="E" THEN 1180
1135 IF LEN(A$)=1 THEN A$="0"+A$
1137 IF LEN(A$)>2 THEN A$=LEFT$(A$,2)
1140 D$(X,1)=A$+"[2x spc]"+
     RIGHT$(D$(X,1),18)
1150 IF A$="15" THEN GOSUB 200:REM 5/15A
1160 PRINT"[up]";S$:
     PRINT"[5x up]";:A$=""
1170 NEXT X
1180 D$(X-1,1)="[13x spc]"+
     RIGHT$(D$(X-1,1),18):X=X-2:
     GOTO 1160
```

MODULE 5/15A Handles the division of a check into more than one account. Provides up to eight accounts into which the check may be divided.

```
200 REM ** 5/15A-DIVIDE ACCOUNTS **
205 AD=0:FOR Y=2 TO 10:Z$=""
210 PRINT"[4x up][rvs]";S$:PRINT
    PRINT:PRINTS$:PRINT"[6x up]"
215 V=VAL(RIGHT$(D$(X,1),8))+AD:
    GOSUB 400:REM 5/15B
220 PRINT"[rvs] INPUT AMOUNT TO BE
    ASSIGNED";V$;"[off][dwn]"
230 INPUT"[2x dwn] AMOUNT";V:
    AD=AD+V:GOSUB 400:REM 4/15B
240 PRINT"[5x up][rvs]";S$:
    PRINTS$:PRINT:PRINT:PRINTS$;
    "[5x up]"
250 PRINT"[rvs][2x spc] INPUT
    CATEGORY FOR THAT AMOUNT [7x spc]":
    PRINT"[rvs][4x spc]"V$;
    LEFT$(S$,27)
260 INPUT"[2x dwn] CATEGORY";A$:
    IF A$="Q" THEN RETURN
265 PRINT"[5x up][rvs]";S$:
    PRINT S$:PRINT:PRINT:
    PRINT S$;"[up]"
270 IF LEN(A$)=1 THEN A$="0"+A$
275 D$(X,Y)=A$+"[2x spc]"+
    MID$(D$(X,1),5,10)+V$:PRINT
280 IF INT(100*VAL(RIGHT$(D$(X,1),8))
    +.5)/100 <= -INT(100*AD+.5)/100
    THEN 290
285 NEXT Y:RETURN
290 D$(X,Y+1)="0":RETURN
```

MODULE 5/15B Similar to Module 5/6B but designed for a more specific purpose. This module converts a number V into a string V$ whose value = V [VAL(V$) = V].

```
400 REM ** 5/15B-CONVERT V TO V$ **
410 V$=STR$(INT(100*V+.5)/100):
    V$=RIGHT$(V$,LEN(V$)-1)
420 IF INT(V+.5)=V THEN V$=V$+".00":
    GOTO 440
430 IF INT(10*V+.5)/10=V THEN V$=V$+"0"
440 IF LEN(V$) > 7  THEN
    V$=STR$(INT(100*VAL(V$)+.5)100):
    PRINT V$"-440":STOP
450 IF LEN(V$)=7 THEN V$="-"+V$:RETURN
460 V$="[spc]"+V$:GOTO 450
```

MODULE 5/16 Lists the items in a two-dimensional string array. This module is designed for the budget program but can be modified for more general use. It uses Module 5/20 to control the scroll if there are more than 24 items.

```
1200 REM ** 5/16-LIST AN ARRAY **
1210 INPUT"[clear][5x dwn]
     [3x rht] LIST ITEMS ([rvs] Y [off]
     /N)";Z$:IF Z$="N" THEN 50
1220 FOR X=1 TO 100:Y=1
1230 IF LEFT$(D$(X,1),2)="15" THEN 1270
1240 PRINT D$(X,1):GOSUB 2200:REM 5/20
```

```
1250 IF D$(X,1)="" THEN 1500:REM 5/17
1260 NEXT X:GOTO 1500:REM 5/17
1270 FOR Y=2 TO 10:IF D$(X,Y)="0" THEN 1250
1280 PRINT D$(X,Y):GOSUB 2200:REM 5/20:NEXT
     Y:GOTO NEXT Y:GOTO 1250
```

MODULE 5/17 Saves the entire set of items in the two-dimensional array. Again, this module is modified for the budget program but is derived from Module 3/7.

```
1500 REM ** 5/17-WRITE DATA TO STORAGE**
1505 Z$="":INPUT"[dwn][5x rht] SAVE
     ITEMS ([rvs] Y [off] /N)";Z$:
     IF Z$="N" THEN 50:REM 5/13
1510 INPUT"[3x dwn] ENTER FILE
     NAME";F$:F$="0:"+F$
1515 OPEN 15,8,15
1520 OPEN 3,8,3,F$+",S,W":INPUT#15,A,B$
     :IF A=0 THEN 1535
1525 Z$="":IF A=63 THEN CLOSE 3:
     PRINT B$;", OVERWRITE ? ([rvs]
     Y [off] /N)";
1526 INPUT Z$:IF Z$="N" THEN 1505
1530 F$="@"+F$:GOTO 1520
1535 FOR X=1 TO 100:Y=1:
     IF D$(X,1)="" THEN CLOSE 3:CLOSE 15:
     GOTO 2000:REM 5/18
1540 PRINT#3,D$(X,1):IF
     LEFT$(D$(X,1),2)="15" THEN 1570
1550 PRINT D$(X,Y)
```

```
1560 NEXT X:GOTO 70:REM 5/13
1570 FOR Y=2 TO 10:PRINT#3,D$(X,Y):
     IFD$(X,Y)="0" THEN 1560
1580 PRINT D$(X,Y):NEXT Y:GOTO 1560
```

MODULE 5/18 Lists the items straight from storage. Uses Manual Scroll Module 5/20.

```
2000 REM ** 5/18-LIST FROM STORAGE **
2010 OPEN 3,8,3,"0:"+F$+",S,R"
2020 FOR X=1 TO 100:Y=1
2030 GOSUB 2120
2040 IF D$(X,1)="" THEN 2100
2050 IF LEFT$(D$(X,1),2)="15" THEN 2070
2060 NEXT X:GOTO 2100
2070 FOR Y=2 TO 10:GOSUB 2120
2080 IF D$(X,Y)="0" THEN 2060
2090 NEXT Y:GOTO 2060
2100 CLOSE 3:INPUT"ENTER [rvs]
     RETURN [off] TO CONTINUE";Z$:
     GOTO 50:REM 5/13
2120 INPUT#3,D$(X,Y):GOSUB 2200
2130 PRINT D$(X,Y):IF ST=64 THEN 2100
2140 RETURN
```

MODULE 5/19 Selects from storage only the data that matches the chosen account number, and prints it.

```
3000 REM ** 5/19- PRINT ACCOUNT **
3010 GOSUB 1000:SUM=0:REM 5/14
```

```
3020 INPUT"ENTER CATEGORY TOKEN";CK$:
     IF LEN(CK$)=1 THEN CK$="0"+CK$
3030 OPEN 3,8,3,"0:"+F$+",S,R":PRINT"[clear]"
3040 IF LEFT$(D$(X,1),2)="15" THEN 3100
3050 FOR X=1 TO 100:Y=1
3060 GOSUB 3150
3070 NEXT X
3080 PRINT:PRINT TAB(7)"TOTAL= $";SUM:CLOSE
     3:CK$=""
3090 Z$="":INPUT"ENTER [rvs]
     RETURN [off] TO CONTINUE";Z$:
     IF Z$<>"" THEN STOP
3095 GOTO 50:REM 5/13
3100 FOR Y=2 TO 10:IF D$(X,Y)="0"
     THEN 3070
3110 GOSUB 3150:NEXT Y: GOTO 3070
3150 INPUT#3,D$(X,Y):IF ST=64 THEN 3080
3155 IF LEFT$(D$(X,Y),2)=CK$ THEN
     PRINT D$(X,Y):
     SUM=SUM+VAL(RIGHT$(D$(X,Y),7))
3160 RETURN
```

MODULE 5/20 Can be used in any program to control the scroll. It allows scrolling of single lines, controlled by the RETURN key. For use in the budget program, the 39 spaces in line 2250 can be replaced by the variable S$. S$ is specified in Module 5/13.

```
2200 REM ** 5/20-MANUAL SCROLL **
2210 Z$="":SC=PEEK(209)+256*PEEK(210)
```

```
2220 IF SC>1983 THEN INPUT"[rvs]
     RETURN [off] TO SCROLL";Z$:
     IF Z$="" THEN 2250
2230 IF Z$="" THEN RETURN
2240 STOP
2250 PRINT"[up][39x spc][up]":
     RETURN
```

Chapter SIX

MORE BUSINESS

Most businesses will have a use for the checkbook and budget programs from Chap. 5. Perhaps even more important, the modules in these programs can be used in many ways to construct customized programs to meet your exact needs. In this chapter we will construct a payroll program, using many of the modules from previous chapters together with new ones designed for use in this program. The modules in this payroll program will be useful in any program that lists employees, summarizes data, or calculates costs per item or employee.

PRODUCING A PAYROLL

To produce a payroll, we must first have an employee listing. In this list we must have the information necessary to calculate the wages, taxes, deductions, and net pay of each employee listed. Because any of the employee's statistics may change, it is necessary that provisions for

change be included in the system. Once we have the employees and their statistics listed, we are ready to start calculating the payroll.

Assuming that our business is normal, we want to enter the hours that each employee worked. We then want to calculate the gross pay, deductions, and net pay for each employee. Calculating gross pay is easy, since this calculation is just total hours worked times the employee's hourly rate of pay. Figuring deductions is more difficult but not insurmountable. Gross pay less deductions yields net pay.

Calculating the net pay for each employee and keeping track of the deductions is what payroll is all about. We must have a list of the total amounts owed to each deduction account. While doing all of these calculations it will be very easy to calculate how much the whole process is costing the employer. This total employer cost and a payroll summary are easily obtainable as by-products of the payroll computations.

INITIAL MODULES

There are actually two initial modules for this program. The first (Module 6/1) provides data constants that are used in the program. The second is actually the initial card module from Chap. 4, Module 4/1.

Module 6/1 loads a DATA line into constants DD(1) through DD(8). These constants are the values used to generate the amounts of the deductions. The corresponding names of the deductions are stored in string constants DD$(1) through DD$(8).

Notice that both the names and the amounts of the deductions can be changed to fit your specific needs. Only the fifth and sixth items [DD(5) and DD(6)—deduct per

dependent and health insurance (employer)] are not percentages of gross pay. The constant DD(5) is the amount allowed by the Internal Revenue Service (IRS) for each dependent claimed by the employee. The value of this constant should be changed to meet the changing rules of the IRS, but the use of DD(5) and DD$(5) should not be changed. The constant DD(6) is used in the program as a whole number and not as a percentage of gross pay. Therefore, DD(6) and DD$(6) should be used only for fixed-value deductions. In the module it is shown to be the amount paid by the employer for each employee's health insurance.

In Module 6/1 a set of arrays are DIMensioned to be used with the listing of weekly hours worked. There is also a constant P$ which represents the pay period. This constant is initially set to equal "unknown" and is changed to equal the current payroll period when the hours are entered.

The other initial module is the same as Module 4/1 except that the line numbers must be changed to begin with line 20 and end with line 90. The other changes in this module are

```
LI=14
WI=28
DIM EM$(100,16) [instead of DIM C$(100,LI)]
GOSUB 1500 [instead of GOSUB 400 AND GOSUB
1000]
```

Making these changes causes Module 4/1 to look like this:

```
          20 REM** 4/1-CARD INITIAL  **
Change    30 LI=14:WI=28:FOR I=LI TO 20:ST$=
             ST$+"[dwn]":NEXT I:
```

```
        ST$="[home]"+ST$
    40  FOR I=1 TO WI:
        D$=D$+CHR$(32):CT$=CT$+CHR$(138):
        CB$=CB$+CHR$(175):NEXT I
    50  CT$=CHR$(207)+CT$+CHR$(208):
        CB$=CHR$(204)+CB$+CHR$(186)
    60  CS$=CHR$(180)+D$+CHR$(170)
Change  70  DIM EM$(100,16)
Change  80  GOSUB 1500
```

MENU AND LISTING EMPLOYEES

This is as good a place as any to discuss the menu for this payroll program. Module 6/2 is the suggested menu. Using this menu we can choose to:

List employees (together with their statistics)

Enter hours worked (five weeks are provided)

See payroll list (calculates payroll and prints it)

Change employee records (change any of the five employee statistics)

List employer costs (lists costs per employee and sum of costs)

Add employee (allows input of new employee)

See payroll summary (a summary of all payroll parameters)

Exit program (saves the payroll data before erasing the program)

Print selected function (prints any of the noninput functions)

Change payroll variables (changes the DATA line variables)

Now that we have a menu, we can move on to listing employees. There are three modules in this program related to listing employees. First, we must be able to retrieve the payroll data from the storage medium (we will talk later about saving the data); second, we must be able to enter new employee data; and third, we must be able to list the data.

The loading function we listed first is one that we are certainly familiar with by now. It involves loading a two-dimensional array. Our first examples of modules for loading data were in Chap. 3. The module we are going to use in this program is Module 4/11 from Chap. 4. The only changes needed are to change the line numbers to start at 1500 rather than 400, print ENTER PAYROLL FILE NAME instead of ENTER CARD FILE NAME, insert a new line between the old lines 430 and 440 which reads INPUT#2,P$, and substitute EM$(X,1)="" for RIGHT$(C$(X,Y),1)=CHR$(92). The changed module looks like this:

```
1500 REM ** 4/11-LOAD EMPLOYEE LIST **
1510 PRINT"[clear][3x dwn][3x rht]ENTER
     PAYROLL FILE
     NAME":INPUT"[4x rht]";FL$:FL$="0:"+FL$
1520 OPEN 15,8,15
1530 OPEN 2,8,2,FL$+",S,R":GOSUB 1600:IF
     A$="62" THEN RETURN
```

```
1535 INPUT#2,P$
1540 FOR X=1 TO 100: FOR Y=1 TO 16
1550 IF X>1 AND ST<>0 THEN 1590
1560 INPUT#2,EM$(X,Y)
1565 IF EM$(X,1)="" THEN 1590
1570 NEXT Y
1580 NEXT X
1590 CLOSE 2:CLOSE 15: RETURN
```

Of course, this loading module also requires that we use the module that reads the error channel, Module 4/9. Module 4/9 is used exactly as it is shown in Chap. 4, except that the line numbers must be changed to start at line 1600.

Entering new employee data is handled by Module 6/4. This module is a modified version of Module 4/5B. It asks for the new employee name, hourly rate, marital status, tax exemptions, and employee status. The marital status code is 1 for single, 2 for married, and 3 for other (widowed, etc.), but 3 is still classified as single. The employee status code is 1 for an employee paying full FICA (social security) deductions, 2 when an employee's annual pay has exceeded the FICA maximum limit, and 0 for the employee not being paid in the pay period (terminated, sick, retired, etc.).

Listing the employees is done by cycling through the employee statistics one employee at a time. This type of data is often kept in a card file, so that is how we will display the data in this program. We will use the card file format (Module 4/2A) from Chap. 4. Module 4/2A is used exactly as it was in Chap. 4 except that the line numbers must be changed to begin at line 600. This module

prints the outline of a small card (28 characters × 14 lines) and prints the outline of the tops of the three following cards. Using this card format allows us to flip through the cards when searching for a certain employee.

With the card format in place, we can now enter the module, which will list the employees on the card. Module 6/3 does exactly that. It GOSUBs to the card format, erases the card, and then prints on the card the employee's name, hourly rate, marital status, tax exemptions, and employee status. It also prints on the outlines of the tops of the following cards the names of the next four employees in the file.

List Employee Module 6/3 uses the control module that we have used in so many programs, which is Module 4/7. This bottom line and erase card module is used exactly as shown in Chap. 4.

With all these modules in place, we have the core of the program working. We can now retrieve data from storage, input a new employee, and list all the employees in the file. Our next step should be to figure out how to change the employee records and save the employee data.

CHANGING EMPLOYEE RECORDS AND SAVING DATA

Being able to change employee records is a very important function in the payroll program. Every item in the employee statistics can change, including the employee name. Module 6/7A allows us to select a single employee (by using Module 6/7B to find the employee data), change that employee's statistics, or change the hours previously entered for the employee.

Module 6/7A can also be used to view the employee records without making any changes. When the employee record is displayed, entering N, R, M, T, E, or H will allow changes in the name, rate, marital status, tax exemptions, employee status, or hours. Entering only RETURN will make no changes and return us to the menu.

If H is selected and the hours worked are to be changed, all the hours for all five weeks must be entered (whether changed or not). If E is selected, and the change is from a status of 0 or 1 to a status of 2, the program will jump to Module 6/7C. Module 6/7C asks for the amount of the employee's wages subject to FICA in this payroll period. When these questions are answered, the program will return to the change module and ask if there are other changes to be made.

Saving the employee data is done when we quit the program. A list of employee data without other payroll information, such as hours, can be saved by exiting the program after adding new employees. This list can be saved with a file name such as EMPLOYEE. The program can then be reentered, and the normal payroll information can be added to run the payroll. The payroll can then be saved under a name, such as PAYROLL FEB '85 (if the payroll is for February 1985). When there are no new employees to be added, the EMPLOYEE file is loaded from storage, payroll data is added, and the payroll is run and saved.

The exit program module is another module taken from Chap. 4. Module 4/10 requires few modifications to become our exit program module. Most of the changes to Module 4/10 are made to identify the file as a payroll file rather than as a card file. With the needed changes, the new module looks like this:

```
1700 REM ** 4/10 (MOD)-EXIT PROGRAM **
1710 PRINT"[clear][2x dwn][3x rht]CURRENT
     PAYROLL FILE NAME IS":Z$="":
     PRINT"[9x rht]"FL$
1720 INPUT"[2x dwn][2x rht]IF NEW PAYROLL FILE
     NAME ENTER NOW[home][8x dwn][3x
     rht]";FL$
1725 IF LEFT$(FL$,3)="@0:" THEN
     FL$=RIGHT$(FL$,LEN(FL$)-1)
1727 IF LEFT$(FL$,2)<>"0:" THEN FL$="0:"
     +FL$:GOTO 1725
1730 PRINT"IF FILE IS READY TO SAVE ":Z$="":
     INPUT"THEN ENTER RETURN";Z$
1735 IF Z$<>"" THEN RETURN
1750 OPEN15,8,15:Z$="":A$=""
1755 OPEN 2,8,2,FL$+",S,W":GOSUB 1600:
     IFA$="63" THEN 1755
1757 PRINT#2,P$
1760 FOR J=1 TO 100:IF EM$(J,1)="" THEN 1790
1765 FOR Y=1 TO 16
1770 PRINT#2,EM$(J,Y)
1780 NEXT Y: NEXT J
1790 CLOSE 2:CLOSE 15:PRINT"[dwn][2x rht]
     SAVE COMPLETED":
     GOSUB 500:IF LEFT$(Z$,1)="*" THEN NEW
1795 GOTO 100
```

Here we have a program which can save and load employee records, enter and list employee statistics, search and find a single employee's records, and change those

records. What we need now is a module that can calculate the payroll.

INPUT HOURS AND CALCULATE THE PAYROLL

Before we can actually calculate the payroll, we must be able to enter the number of hours worked. This program is designed to take weekly hours for up to five weeks and calculate the payroll for the month. The payroll, as shown, is not a weekly or biweekly payroll. It is a monthly payroll for hourly employees. Later, we will discuss how to change this hourly payroll to other types of payroll.

Module 6/5 allows us to enter the hours worked by each employee. The module begins with the warning that if we enter hours into an employee's record, we will delete any hours which are currently in that record. This means if we have entered and stored weeks 1 and 2, and now want to add week 3, we must reenter weeks 1 and 2 before entering week 3. It is assumed that we will enter all of an employee's hours for the month at one time. If this is impractical for your payroll, the module can be changed to input only one week at a time.

After the module warns us about how entering hours will effect our data, it asks us if only one employee is wanted. This question allows us to work with a single employee's records, if desired. If the answer to the question is no, the module begins with employee 1 and lists, in turn, each eligible employee on the payroll until told to "stop" or until the payroll list is exhausted. Employees whose employee status is 0 are not listed by this module.

If the hours for more than one employee are being entered, the module asks for a payroll period name. This

period name can be whatever we choose, but the period should be either a month such as "January" or a date period such as "1/5/86–2/2/86."

Finally, the module asks for us to input the hours worked in weeks 1 through 5. The module then sums and prints the total number of hours for the month and asks whether or not to continue to the next employee.

At long last we are in a position to be ready to calculate a payroll. The actual calculation is done by Modules 6/6A, 6/6B, and 6/6C. Module 6/6A does the majority of the work and prints the results, Module 6/6B contains the income tax table for single employees (marital status = 1 or 3), and Module 6/6C contains the income tax tables for married employees (marital status = 2).

Examination of Module 6/6A will show that each element of employee wages and deductions are calculated here. The employee arrays hold all the data related to each employee. The array is arranged like this:

```
EM$(N,1)  = employee name
EM$(N,2)  = hourly rate of pay
EM$(N,3)  = marital status
EM$(N,4)  = tax exemptions
EM$(N,5)  = employment status
EM$(N,6)  = total hours worked in the period
EM$(N,7)  = gross pay for the period
EM$(N,8)  = gross pay less exemption deductions
EM$(N,9)  = amount of DD$(1) (state disability)
EM$(N,10) = amount of DD$(3) (FICA)
EM$(N,11) = amount of federal withholding income
            tax
```

EM$(N,12) = amount of DD$(7) (employee retirement)
EM$(N,13) = used to calculate FICA for employee status 2
EM$(N,14) = amount of state withholding income tax
EM$(N,15) = employee's net pay
EM$(N,16) = "£"

The module calculates each deduction in turn. When it arrives at income tax, it goes to either Module 6/6B or 6/6C, depending on the employee's marital status. Modules 6/6B and 6/6C contain the 1984 tables for calculating a monthly payroll using the percentage method. This table is given in IRS Publication 15. Both modules will need to be changed each time the IRS tax table changes. Both modules begin by setting the gross pay less the deductions for exemptions equal to the variable NP [NP = VAL(EM$(N,8)].

Each of the modules also contains the state income tax deduction. This method and amount will vary from state to state and must be adjusted to fit your state's laws. As shown on line 2060 of Module 6/6B, the single tax is 5 percent of any net monthly pay (NP) over $267. In Module 6/6C, line 2560 calculates the married tax using the same form but uses 3.5 percent tax rather than 5 percent. You will have to insert your own state's limits and percentages into each module.

If your state income tax is graduated like the federal tax, you can use as a model the part of the module that calculates the federal tax. It is not difficult. A typical line reads

```
2025 IF NP<1408 AND NP>742 THEN X=94 +
     .20*(NP-742)
```

This line simply says that if NP (the gross pay less the exemption deductions) is more than $742 and less than $1408, the tax (X) is $94 plus 20 percent of everything over $742. This is not too hard to follow and is about as difficult as it gets. With a little care you can cause the module to calculate your state tax.

EMPLOYER COSTS AND PAYROLL SUMMARY

Modules 6/8 and 6/9 are summaries for the payroll data. Module 6/8 is a summary of employer costs. When this module is selected, the cost records of all employees eligible for payroll are displayed one at a time. These cost records include only those items that the employer pays, such as gross pay and industrial accident insurance premium, but not items that the employee pays, such as withholding tax or FICA. These employer cost items include the employer's payment toward the employee's retirement but would not include the employee's payment toward that same retirement.

After displaying the costs of each individual employee, the module summarizes and displays all the sources of employer costs, the total cost of the payroll to the employer, and this month's deposit which is owed to the IRS. This summary is contained on a single page, which can be viewed on the screen or printed out as hard copy.

Payroll Summary Module 6/9, summarizes all the items relating to the payment of the payroll (see Fig. 6-1).

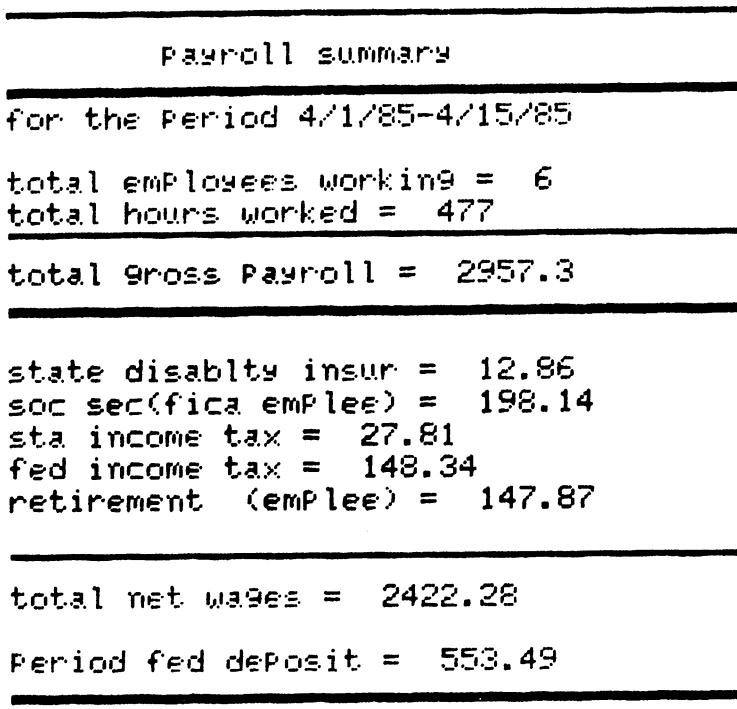

Figure 6-1 Printout of payroll summary

It lists the total number of employees working, the total number of hours worked, and the total gross payroll. It then lists the total of DD$(1) (state disability insurance), DD$(3) (employee FICA), state income tax, federal income tax, DD$(8) (retirement paid by the employee), and total net wages. This list could be considered the summary of employee costs. Finally, the summary prints the amount of federal deposit due for the payroll period listed.

PRINT AND CHANGE PAYROLL VARIABLES

All the functions of the payroll program, which do not expect input, can be printed. Module 6/10 provides a selection of the printable functions. We can print an employee list which includes the employee's statistics. We can print the payroll list of all eligible employees, including the hours worked, gross pay, all deductions, and net pay for each employee. We can choose to list the employer costs, which prints an itemized account of the employer costs for each employee and then prints a summary of those costs. We can print the payroll summary, which is a single-page listing of the total number of workers, total number of hours worked, and totals of gross pay, deductions, and net pay, together with the total amount of federal deposit required for this period.

The print module even allows us to print the listing of the payroll variables. This payroll variable listing prints all the names and values of DD$(1) and DD(1) through DD$(8) and DD(8). These are the values used to calculate the payroll deductions and employer costs. As discussed earlier, these values can be changed to meet your specific needs and to meet the changes in local and federal law.

The last module in this program is Module 6/11, payroll variables. This module lists the variables that we have just been discussing and allows us to change the values of those variables, if desired.

If an item is to be changed while viewing this list, the item number is entered. The screen is cleared and the current value of the chosen variable is displayed together with a request for the new value. This new value must be

six characters in length. For example, if the new value is 0.2, it should be entered as 000.20 or 0.2000 or a similar value which provides six characters without changing the value of the desired variable. If the new value is more than six characters in length, it must be rounded to six characters.

When the new value has been entered, the module asks whether or not this change is to be permanent. If the answer is no, the new value is used in the current calculations, but no changes are made in the payroll program. This feature can be useful for making temporary changes or for estimating the costs for proposed changes in payroll variables.

If, on the other hand, the changes are to be permanent, the module looks through the payroll program for a DATA line and changes the data to incorporate the new variable. The module then saves the entire program, including the new data line. You may wish to insert a warning line in this module to be sure that the right disk is in the drive, since the module will save the program on whatever disk is available. This warning might be something like this:

```
5205 PRINT"SAVING PROGRAM, CORRECT
     DISK?":GOSUB 500:IF
     Z$="*" THEN 5000
```

This line allows us to return to the listing of the changed variables without saving the program. This condition is different from that which we would have if we had selected a nonpermanent change, because in this case we *have* changed the data line.

We now have three possible conditions that we can

use. The first is the nonpermanent condition, which changes only the value held in the computer variable. This value will be lost if the program is RUN or if the program is LOADed from the disk (or tape). The second condition is the one that we have created with line 2505. Here we have changed the DATA line but have not saved the changed program. In this case the changes will remain intact if the program is RUN but not if the program is LOADed. The third condition makes permanent changes in the DATA line and saves the changed program. This condition allows us to RUN or LOAD without losing the new variables.

THE PAYROLL PROGRAM

We now have a complete payroll program that will provide most reports that we would need or want. The modules and their order in the program are:

6/1 Payroll initial
4/1 Card initial
6/2 Payroll menu
6/3 List employees
6/5 Enter hours worked
6/6A Calculate payroll
4/7 Bottom line/erase card
4/2A Print card
6/4 Employee card layout (Module 4/5B modified)
6/7A Change employee records
6/7C Employee status = 2

6/8 Employer costs
6/9 Payroll summary
6/7B Search subroutine
4/11 Load employee list (modified)
4/9 Read error channel
4/10 Exit program (modified)
6/10 Print routine
6/6B Income tax—1 or 3
6/6C Income tax—2
6/11 Change payroll variables

If you are so inclined and have the skills, it would not be too difficult to write a module that would convert the payroll payment data into the form for checkbook data. Writing such a module would provide a checkwriting guide and record. It would also allow the budget program to use the same payroll data and assign account names to the different employee groups, if desired. There are many possible uses.

CHANGING TO A DIFFERENT PAYROLL PERIOD

As we said earlier, this payroll is an hourly payroll, recorded weekly and paid monthly. How can we change it to another type of payroll period? Basically, there are two changes that must be made.

First, if we are changing to a pay period shorter than a month, we must change Module 6/5 to expect only the number of weeks in the period. This is done by removing lines 330 through 350 from the INPUT lines not needed.

Then line 360 can be changed to read "TOTAL HOURS FOR PERIOD".

The other change that must be made is to adjust cost and deduction variables to match the shorter pay period. Of course, the income tax modules (6/6B and 6/6C) must be changed.

To change to a salaried payroll, the easiest change would be to establish an appropriate combination of hourly rate and monthly hours to be used. This combination could be a rate of $1200 per hour for one hour per month if the gross pay for that employee is $1200 per month.

OTHER USES FOR THE MODULES

There are a number of unique modules in this program which can have other uses. Module 6/11 (change payroll variables) can, with minor modifications, be used in other programs. The basic idea of being able to change a program data line is powerful, since we are using a program to change itself. You will find uses for this module in other programs.

Exit Module 6/10 can also be useful in other programs. The basic module concept is usable as shown. It is necessary only to make adjustments that accommodate differences between the data to be saved in the new program and the data being saved in this payroll program. Modules 6/7A and 6/7B can be used in any data listing program where search and change are helpful. Simply making changes in the item names will adapt these modules to other programs. Finally, Module 6/3 can be used for short lists of data, whether name and address or inventory items and costs.

Many of the concepts in this payroll program can be adapted to other programs, such as inventory, sales data, and so on. Some of the modules in this chapter, coupled with modules from the card file program, checkbook program, and others, will provide unusual customized programs to fit your needs.

CHAPTER SIX

MODULES

MODULE 6/1 Initial module for the payroll program, which loads a DATA line into an array DD(X) which corresponds to a name list array DD$(X).

```
001 REM ** 6/1-PAYROLL INITIAL **
005 TP$="[39x cmd/Y]"
006 BT$="[39x cmd/P]"
007 FOR X=0 TO 8:READ DD(X):NEXT X
008 DATA 0,0.0270,0.0818,0.0670,0.0700,
    083.33,080.00,0.0700,0.0500
009 RESTORE
010 P$="UNKNOWN"
011 DD$(1)="STATE DISABILITY INSUR"
012 DD$(2)="INDUSTRIAL ACCIDENT"
013 DD$(3)="SOC SEC(FICA EMPLEE)"
014 DD$(4)="SOC SEC(FICA EMPLER)"
015 DD$(5)="DEDUCT PER DEPENDENT"
016 DD$(6)="HEALTH INSUR (EMPLER)"
017 DD$(7)="RETIREMENT (EMPLER)"
018 DD$(8)="RETIREMENT (EMPLEE)"
019 N=100:DIM W1(N),W2(N),W3(N),W4(N),
    W5(N),HR(N),Q(15)
```

MODULE 6/2 The payroll menu, which selects among the functions available in the payroll program. There are 10 possible choices.

```
100 REM ** 6/2-PAYROLL MENU **
110 PRINT"[clear][2x dwn][2x spc][rvs]
    SELECT ONE OF THE FOLLOWING:[off]"
120 PRINT"[dwn][7x spc]1 LIST EMPLOYEES"
130 PRINT"[7x spc]2 ENTER HOURS WORKED":
    PRINT"[7x spc]3 SEE PAYROLL LIST"
140 PRINT"[7x spc]4 CHANGE EMPLOYEE
    RECORDS":PRINT"[7x spc]5 EMPLOYER
    COSTS"
150 PRINT"[7x spc]6 ADD EMPLOYEE":
    PRINT"[7x spc]7 SEE PAYROLL SUMMARY"
160 PRINT"[7x spc]8 EXIT PROGRAM"
170 PRINT"[7x spc]9 PRINT SELECTED
    FUNCTION"
175 PRINT"[6x spc]10 CHANGE PAYROLL
    VARIABLES"
180 INPUT"[2x dwn][2x spc]ENTER
    YOUR CHOICE";Z$
185 ON VAL(Z$) GOSUB 200,300,400,900,
    1100,800,1200,1700,1900,5000
190 GOTO 100
```

MODULE 6/3 Lists each employee in the file, together with the employee's number, rate, and statistics. A marital status of 1 denotes a single employee, 2 a married employee, and 3 any other classification treated as single. An employee status of 1 denotes a normal employee, 2 is an employee whose wages for the year have exceeded the

FICA maximum, and 0 denotes an employee who is ineligible for pay in the period being computed.

```
200 REM ** 6/3-LIST EMPLOYEES **
210 GOSUB 600:REM ** PRINT CARD SUBR **
215 FOR N=1 TO 100:IF EM$(N,1)=""
    THEN RETURN
220 IF FLAG=1 THEN PRINT:GOTO 230
225 GOSUB 550:PRINT"[home]":FOR B=1
    TO 4:PRINT TAB(12-2*B);EM$(N+5-B,1);
    "[dwn]":NEXT B
230 PRINT TAB(4)"EMPLOYEE NUMBER "N:
    PRINT
235 FOR L=1 TO 5:PRINT"[2x spc]";
240 IF L=1 THEN PRINT"NAME = ";
245 IF L=2 THEN PRINT"[4x spc]HOURLY
    RATE =$";
250 IF L=3 THEN PRINT"[spc]MARITAL
    STATUS = ";
255 IF L=4 THEN PRINT"[4x spc]TAX
    EXEMPTS = ";
260 IF L=5 THEN PRINT"EMPLOYEE STATUS=";
265 PRINT EM$(N,L):NEXT L:PRINT:GOSUB
    500:IF LEFT$(Z$,1)="*" THEN RETURN
270 NEXT N:PRINT"[3x spc][rev]PAYROLL
    END[off]":GOSUB 500:RETURN
```

MODULE 6/4 A modified version of Module 4/5B. It prints the name of an employee statistic and then accepts input for that statistic. Lines 865 through 870 provide for delete and backspace.

```
800 REM ** 6/4-EMPLOYEE CARD **
805 PRINT"[clear]":FOR N=1 TO 100:
    IF EM$(N,1)="" THEN 820
810 NEXT N:PRINT"[3x spc][rvs]PAYROLL
    FULL[off]":STOP
820 PRINT TP$:PRINT TAB(9)"NEW EMPLOYEE
    NUMBER "N:PRINT BT$:PRINT"[2x dwn]"
825 FOR L=1 TO 15:PRINT"[2x rht]";
830 IF L=1 THEN PRINT"[2x spc]EMPLOYEE
    NAME = ";
831 IF L=2 THEN PRINT"[4x spc]HOURLY
    RATE =$";
832 IF L=3 THEN PRINT"[spc]MARITAL
    STATUS = ";
833 IF L=4 THEN PRINT"[4x spc]TAX
    EXEMPTS = ";
834 IF L=5 THEN PRINT"EMPLOYEE STATUS=";
835 IF L<6 THEN GOSUB 850:NEXT L
837 EM$(N,L)="0":NEXT L:PRINT:EM$(N,16)
    ="£":GOSUB 500:IF Z$;"*" THEN RETURN
840 GOTO 800
850 GET A$:IF A$="" THEN 850
855 IF A$=CHR$(13) THEN PRINT:RETURN
860 IF A$<>CHR$(20) THEN 875
865 IF LEN(EM$(N,L)) >0 THEN 870
866 L=L-1:PRINT"[up]":FOR K=20
    TO 39:PRINT"[lft][spc][lft]";:
    NEXT K:EM$(N,L)="":GOTO 850
870 EM$(N,L)=LEFT$(EM$(N,L),LEN
    (EM$(N,L))-1):PRINT"[lft][spc]
    [lft]";:GOTO 850
```

```
875 PRINT A$;:EM$(N,L)=EM$(N,L)+A$
880 IF LEN(EM$(N,L))<WI THEN 850
885 PRINT RETURN
```

MODULE 6/5 Enters hours worked. Will accept a single employee or will cycle through the entire eligible employee list and enter five weeks of hours. Calculates total hours for the month.

```
300 REM ** 6/5-ENTER HOURS WORKED **
302 PRINT"[clear]ENTERING HOURS WILL
    DELETE ALL PREVIOUS HOURS IN
    THAT EMPLOYEE'S ";
303 PRINT"RECORD.":GOSUB 500:IF Z$="*"
    THEN RETURN
305 PRINT:Z$="":INPUT"ENTER ONLY ONE
    EMPLOYEE'S HOURS? (Y/N)";Z$:
    IF Z$="N" THEN 309
306 IF Z$<>"Y" THEN 305
307 GOSUB 1400:X=N:REM ** TO 6/07B **
308 FOR N=X TO X:GOTO 320
309 INPUT"[clear][2x dwn][spc]INPUT
    THE PERIOD";P$
310 FOR N=1 TO 100:IF EM$(N,1)=""
    THEN RETURN
315 IF EM$(N,5)="0" THEN 375
320 PRINT"[clear]"TP$:PRINT TAB(10)
    "WAGE ENTRIES":PRINT BT$:
    PRINT"PERIOD: "P$:PRINT
325 PRINT"[dwn][spc]EMPLOYEE NUMBER-";
    N"[3x spc]"EM$(N,1)
```

```
330 INPUT"[dwn][3x spc]HOURS FOR WEEK
    ONE";W1(N)
335 INPUT"[dwn][3x spc]HOURS FOR WEEK
    TWO";W2(N)
340 INPUT"[dwn][spc]HOURS FOR WEEK
    THREE";W3(N)
345 INPUT"[dwn][2x spc]HOURS FOR WEEK
    FOUR";W4(N)
350 INPUT"[dwn][2x spc]HOURS FOR WEEK
    FIVE";W5(N)
355 HR(N)=W1(N)+W2(N)+W3(N)+W4(N)+W5(N)
360 PRINT"[dwn][spc]TOTAL HOURS FOR
    MONTH=";HR(N)
365 EM$(N,6)=STR$(HR(N))
370 GOSUB 500:IF Z$="*" THEN RETURN
375 NEXT N:RETURN
```

MODULE 6/6A Together with Modules 6/6B and 6/6C, calculates the payroll. The module cycles through the list of employees in sequence, calculating gross pay, all of the deductions, and net pay. Modules 6/6B and 6/6C contain the income tax tables for both single and married employees.

```
400 REM ** 6/6A-CALCULATE PAYROLL **
403 GOSUB 600:REM *TO 4/2A CARD FORMAT*
405 FOR N=1 TO 100:IF EM$(N,1)=""
    THEN 495
407 IF EM$(N,5)="0" THEN 490
410 IF FLAG=1 THEN B=5:PRINT:GOTO 413
412 GOSUB 550:PRINT"[home]":FOR B=1
```

```
    TO 4:PRINT TAB(12-2*B);EM$(N+5-B,1);
    "[dwn]":NEXT B
413 PRINT TAB(12-2*B);EM$(N,1);
415 HR=VAL(EM$(N,6)):GP=HR*VAL
    (EM$(N,2)):GP=INT(100*GP+.5)/100:
    EM$(N,7)=STR$(GP)
417 EM$(N,7)=STR$(INT(100*VAL
    (EM$(N,7))+.5)/100)
420 EM$(N,9)=STR$(INT(100*HR*
    DD(1)+0.5)/100)
425 EM$(N,10)=STR$(INT(100*GP*
    DD(3)+0.5)/100)
430 IF EM$(N,5)="2" THEN EM$(N,10)=
    EM$(N,13)
435 EM$(N,12)=STR$(INT(100*GP*
    DD(8)+0.5)/100)
437 EM$(N,8)=STR$(GP-DD(5)*VAL
    (EM$(N,4)))
440 IF EM$(N,3)="1" OR EM$(N,3)="3"
    THEN GOSUB 2000:GOTO 445
441 IF EM$(N,3)="2" THEN GOSUB 2500:
    GOTO 445
442 PRINT"ERROR IN MARITAL STATUS":
    GOSUB 500:GOTO 900:REM * TO 6/7A **
445 EM$(N,15)=STR$(GP)
446 FOR X=9 TO 14:EM$(N,14)=STR$
    (VAL(EM$(N,15)-VAL(EM$(N,X))):NEXT X
447 EM$(N,15)=STR$(INT(100*VAL
    (EM$(N,15))+.5)/100)
450 PRINT"[2x spc]EMPL #"N:PRINT
    TAB(2)"[spc]RATE="EM$(N,2)"[2x spc]
    HOURS="EM$(N,6):PRINT
```

```
455 PRINT TAB(2)"[5x spc]GROSS PAY =$"
    EM$(N,7):PRINT
460 PRINT DD$(1)"[spc]="EM$(N,9)
465 PRINT DD$(8)"[spc]="EM$(N,12)
470 PRINT DD$(3)"[spc]="EM$(N,10)
475 PRINT TAB(2)"FEDERAL INCOME TAX ="
    EM$(N,11)
480 PRINT TAB(2)"[2x spc]STATE INCOME
    TAX ="EM$(N,14)
485 PRINT TAB(2)"[6x spc]NET PAY ="
    EM$(N,15)
486 IF FLAG=1 THEN PRINT TP$:GOTO 490
487 GOSUB 500:IF Z$="*" THEN RETURN
490 NEXT N:IF FLAG=1 THEN RETURN
495 PRINT"[2x spc]END OF PAYROLL":
    GOSUB 500:RETURN
```

MODULE 6/6B Contains the 1984 federal income tax table for single employees who are paid monthly. The table must be updated each time the tax rate is changed. This module also contains a state income tax calculation which must be changed to meet your state's tax rate and method. This module is designed to be a subroutine of Module 6/6A.

```
2000 REM ** 6/6B-INCOME TAX 1 or 3 **
2010 NP=VAL(EM$(N,8)):IF NP < 117
     THEN TX=0
2015 IF NP<267 AND NP>117 THEN
     TX=0.12*(NP-117)
```

```
2020 IF NP<742 AND NP>267 THEN
     TX=18+0.16*(NP-267)
2025 IF NP<1042 and NP>742 THEN
     TX=94+0.20*(NP-742)
2030 IF NP<1408 AND NP>1042 THEN
     TX=154+0.24*(NP-1042)
2035 IF NP<1875 AND NP>1408 THEN
     TX=241.84+0.30*(NP-1408)
2040 IF NP<2317 AND NP>1875 THEN
     TX=381.94+0.34*(NP-1875)
2045 IF NP>2317 THEN
     TX=532.22+0.37*(NP-2317)
2050 EM$(N,11)=STR$(INT(100*TX+.5)/100):
     REM ** FED INCOME TAX **
2060 IF NP>267 THEN EM$(N,14)=STR$
     (INT(100*(.05*(NP-267))+.5)/100):
     REM ** STATE TAX **
2070 RETURN
```

MODULE 6/6C The tax table for married employees who are paid monthly. This module is the same as Module 6/6B except that this module is for married employees.

```
2000 REM ** 6/6C-INCOME TAX - 2 **
2010 NP=VAL(EM$(N,8)):IF NP < 200
     THEN TX=0
2015 IF NP<506 AND NP>200 THEN
     TX=0.12*(NP-200)
2020 IF NP<998 AND NP>506 THEN
     TX=36.72+0.16*(NP-506)
```

```
2025 IF NP<1545 AND NP>998 THEN
     TX=115.44+0.19*(NP-998)
2030 IF NP<1967 AND NP>1545 THEN
     TX=219.37+0.24*(NP-1545)
2035 IF NP<2408 AND NP>1967 THEN
     TX=320.65+0.27*(NP-1967)
2040 IF NP<2850 AND NP>2408 THEN
     TX=439.72+0.32*(NP-2408)
2045 IF NP>2850 THEN
     TX=581.16+0.37*(NP-2850)
2050 EM$(N,11)=STR$(INT(100*TX+.5)/100):
     REM ** FED INCOME TAX **
2060 IF NP>267 THEN EM$(N,14)=STR$
     (INT(100*(.035*(NP-267))+.5)/100):
     REM ** STATE TAX **
2070 RETURN
```

MODULE 6/7A Part of a set of three modules (6/7A, 6/7B, and 6/7C). These three modules find an employee's records, accept changes to those records, and if the employee's status is changed to 2, asks for additional information. In this module (6/7A) all employee records, including name and hours worked, can be changed.

```
900 REM ** 6/7A-CHANGE EMPLEE RECORDS**
910 GOSUB 1400:REM *TO 6/7B SEARCH **
920 PRINT"[clear][dwn]"TP$:
    PRINT TAB(9)"EMPLOYEE NUMBER [rvs]
    "N"[off]":PRINT BT$:PRINT"[dwn]"
930 PRINT"[4x spc]EMPLOYEE NAME = ";
    EM$(N,1)
932 PRINT"[6x spc]HOURLY RATE =$";
    EM$(N,2)
```

```
934 PRINT"[3x spc]MARITAL STATUS = ";
    EM$(N,3)
936 PRINT"[6x spc]TAX EXEMPTS = ";
    EM$(N,4)
938 PRINT"[2x spc]EMPLOYEE STATUS = ";
    EM$(N,5)
940 PRINT"[2x spc]HOURS FOR WEEK
    ONE = ";W1(N)
942 PRINT"[2x spc]HOURS FOR WEEK
    TWO = ";W2(N)
944 PRINT"HOURS FOR WEEK
    THREE = ";W3(N)
946 PRINT"[spc]HOURS FOR WEEK
    FOUR = ";W4(N)
948 PRINT"[spc]HOURS FOR WEEK
    FIVE = ";W5(N)
950 PRINT"TOTAL HOURS FOR MONTH = ";
    EM$(N,6)
955 Z$="":INPUT"[dwn]ENTER N, R, M, T, E, OR
    H TO CHANGE";Z$:IF Z$="" THEN RETURN
960 ZZ$="NRMTEH":FOR X=1 TO 6:
    IF Z$=MID$(ZZ$,X,1) THEN 970
965 NEXT X:RETURN
970 IF X=6 THEN X=N:GOSUB 307:N=N-1:
    GOTO 920:REM *GOSUB 6/5 ENTER HOURS*
975 PRINT"[home][4x dwn]":FOR I=1 TO X:
    PRINT"[dwn]";:NEXT I
980 INPUT"[20x rht][16x spc]
    [16x lft]";Y$:EM$(N,X)=Y$
985 IF X=5 AND Y$="2" THEN GOSUB 1000:
    REM ** TO 6/7C STATUS=2 **
```

```
990 PRINT"[10x dwn]":GOSUB 500:
    IF Z$="*" THEN RETURN
995 GOTO 920
```

MODULE 6/7B Searches for an employee's records either by name or by employee number. If the input string has a length of less than three, the program assumes that the string is a number. If the length is three or more, the program tries to match the input string to an employee name.

```
1400 REM ** 6/7B-SEARCH ROUTINE **
1410 INPUT"[clear][2x dwn][rht]ENTER
     EMPLOYEE'S NAME OR NUMBER[home]
     [3x dwn][3x rht]";F$
1420 IF LEN(F$)<3 AND VAL(F$)>0
     THEN N=VAL(F$):RETURN
1430 FOR X=1 TO 100:IF LEFT$(EM$(X,1),
     LEN(F$))=F$ THEN N=X:RETURN
1440 NEXT X:PRINT"[clear][3x dwn]
     [2x rht]NOT FOUND, IS IT SPELLED
     RIGHT?":GOSUB 500:IF Z$="*" THEN 100
1450 GOTO 1410
```

MODULE 6/7C Asks for the amount of wages which must have FICA deductions made. When an employee's wages for the year have reached the FICA maximum, the FICA deductions are stopped. If FICA deductions are stopped in the current period, deductions must be made for the portion of the current wages that does not exceed the maximum.

```
1000 REM ** 6/7C-EMPLOYEE STATUS=2 **
1010 PRINT"[clear][4x dwn][2x rht]
     EMPLOYEE STATUS 2 INDICATES THAT"
1015 PRINT"[2x rht]EMPLOYEE WAGES
     FOR THIS PERIOD EXCEED":PRINT"
     [2x rht]THE SOCIAL SECURI";
1020 PRINT"TY MAXIMUM.":PRINT:PRINT"
     [2x rht]ENTER THE AMOUNT
     OF WAGE IN THIS"
1025 PRINT"[2x rht]PERIOD SUBJECT
     TO SOCIAL SECURITY,":PRINT"
     [2x rht]FOR ";EM$(N,1)
1030 PRINT TP$:INPUT"[5x rht]$ ";GM$
1040 IF GM$="0" OR GM$=CHR$(13)
     THEN RETURN
1050 EM$(N,13)=STR$(INT(100*VAL(GM$)
     *DD(3)+.5)/100):RETURN
```

MODULE 6/8 Summarizes and displays all the payroll elements that are paid by the employer. First, each employee's costs to the employer are displayed, and then the total costs for all employees are displayed. The FLAG in line 1115 is a signal used when the data is being printed. It avoids printing the card form, so only data is printed.

```
1100 REM ** 6/8-EMPLOYER COSTS **
1105 GOSUB 600:LL=0:FD=0
1110 FOR N=1 TO 100:L=0:IF EM$(N,1)=""
     THEN 1195
1115 IF FLAG=1 THEN 1125
```

```
1120 GOSUB 550:PRINT"[home]":
     FOR B=1 TO 4:PRINT TAB(12-2*B);
     EM$(N+5-B,1);"[dwn]":NEXT B
1125 GP=VAL(EM$(N,7)):IF EM$(N,5)="0"
     THEN 1190
1130 PRINT"[2x spc]"EM$(N,1)
1133 PRINT"[2x spc]EMPL #"N
     "[2x spc]RATE="EM$(N,2):PRINT
     "[11x spc]HOURS ="EM$(N,6):PRINT
1135 PRINT"[11x spc]GROSS PAY =$"
     EM$(N,7)
1140 PRINT DD$(2)" = "INT
     (100*(VAL(EM$(N,6))*DD(2))+.5)/100
1145 PRINT DD$(7)" = "
     INT(100*GP*DD(7)+.5)/100
1150 PRINT DD$(6)" = "DD(6)
1155 PRINT DD$(4)" = "
     INT(100*GP*DD(4)+.5)/100
1160 L=GP*(1+DD(4))+VAL(EM$(N,6))*DD(2)
     +DD(6)+GP*DD(7):IF GP=0 THEN L=0
1165 L=INT(L*100+.5)/100:LL=LL+L
1170 PRINT:PRINT"[3x spc]TOTAL COST =$"
     L:PRINT:GOSUB 500:IF Z$="*"
     THEN RETURN
1180 FD=FD+(DD(3)+DD(4))*GP+
     VAL(EM$(N,11))
1190 NEXT N
1195 PRINT"[clear][2x dwn]":PRINT TP$:
     PRINT TAB(8)"[2x spc]FED DEPOSIT
     =$"INT(100*FD+.5)/100:PRINT
```

```
1197 PRINT TAB(10)"TOTAL COSTS = $"LL:
     PRINT BT$:GOSUB 500:RETURN
```

MODULE 6/9 Summary of the costs paid by the employer. The number of employees to be paid are counted and the total hours are summed. The gross pay, deductions, and net pay are summed. These figures, together with the amount of the federal deposit due for the period, are displayed on a single page.

```
1200 REM ** 6/9-PAYROLL SUMMARY **
1210 PRINT"[clear][dwn]"TP$:PRINT
     TAB(8)"PAYROLL SUMMARY":PRINT BT$
1215 PRINT"FOR THE PERIOD ";P$:PRINT:
     FOR I=1 TO 15:Q(I)=0:NEXT I:B=0:FD=0
1220 FOR N=1 TO 100:IF EM$(N,1)=""
     THEN 1240
1225 B=B+1:IF EM$(N,5)="0" THEN B=B-1:
     GOTO 1235
1230 FOR W=6 TO 15:Q(W)=Q(W)+
     VAL(EM$(N,W)):NEXT W
1233 FD=DF+(DD(3)+DD(4))*Q(7)+Q(11)
1235 NEXT N
1240 PRINT"TOTAL EMPLOYEES WORKING = "B:
     PRINT"[3x spc]TOTAL HOURS
     WORKED = "Q(6)
1245 PRINT TP$:PRINT"[2x spc]TOTAL
     GROSS PAYROLL = "Q(7):PRINT BT$
1250 PRINT:PRINT DD$(1)" = "Q(9):
     PRINT DD$(3)" = "Q(10)
```

```
1253 PRINT"[4x spc]STATE INCOME
     TAX = "Q(14)
1255 PRINT"[2x spc]FEDERAL INCOME
     TAX = "Q(11):PRINT DD$(8)" = "Q(12)
1260 PRINT:PRINT TP$:PRINT"[3x spc]
     TOTAL NEW WAGES = "Q(15):PRINT
1270 PRINT"PERIOD FED DEPOSIT = "
     INT(FD*100+.5)/100:PRINT BT$:
     GOSUB 500:RETURN
```

Module 6/10 The print routine, which gives a menu of lists and reports that can be printed. This routine can be used with other programs by changing the names of the items and the GOSUB targets.

```
1900 REM ** 6/10-PRINT ROUTINE **
1910 PRINT"[clear][dwn]"TP$:PRINT
     TAB(8)"PRINT ROUTINE":PRINT BT$
1920 PRINT"[dwn][7x spc]1 LIST
     EMPLOYEES"
1930 PRINT"[7x spc]2 PAYROLL LIST"
1940 PRINT"[7x spc]3 LIST
     EMPLOYER COSTS"
1950 PRINT"[7x spc]4 PAYROLL SUMMARY"
1960 PRINT"[7x spc]5 PAYROLL VARIABLES"
1965 PRINT"[7x spc]6 RETURN TO MENU"
1970 INPUT"[2x dwn][2x spc]ENTER
     ITEM TO BE PRINTED";Z$
1973 IF VAL(Z$) <1 OR VAL(Z$) >5
     THEN FLAG=0:RETURN
1975 IF Z$="5" THEN POKE 198,2:
     POKE 631,57:POKE 632,13
```

```
1980 OPEN 2,4:CMD 2:FLAG=1
1981 IF Z$="1" THEN PRINT TP$:PRINT
     TAB (10)"EMPLOYEE LIST":PRINT BT$
1982 IF Z$="2" THEN PRINT TP$:PRINT
     TAB (10)"PAYROLL LIST":PRINT BT$
1983 IF Z$="3" THEN PRINT TP$:PRINT
     TAB (10)"EMPLOYER COSTS":PRINT BT$
1985 ON VAL(Z$) GOSUB
     210,405,1110,1200,5000
1990 PRINT#2:CLOSE 2:RETURN
```

MODULE 6/11 The payroll variable routine, which lists the names and values of the variables that are used to calculate the payroll. The variables can be changed, either temporarily or permanently, by this module. The values for the payroll variables are stored in a DATA line in the initial module of this program.

```
5000 REM ** 6/11-PAYROLL VARIABLES **
5010 PRINT"[clear][dwn]"TP$:PRINT
     TAB(12)"PAYROLL VARIABLES":PRINT BT$
5020 FOR X=1 TO 8:PRINT"[dwn][spc]";X;
     "[3x lft] ( [spc] ) [spc]"DD$(X)
     " = "DD(X):NEXT X
5030 PRINT"[dwn][spc] (9) [zx rht][rev] FOR
     NO CHANGE":Z$=""
5040 INPUT"[dwn][rev]TO CHANGE ITEM
     AMOUNT, ENTER NUMBER[rv off]";Z$
5050 PRINT"[clear]":IF Z$="9"
     THEN RETURN
5060 Z=VAL(Z$)
```

```
5070 PRINT"[3x dwn]"DD$(VAL(Z$))
     " = "DD(VAL(Z$))
5080 INPUT"[2x dwn][2x rht] NEW VALUE
     (6 CHARS) = ";VA$:DD(Z)=VAL(VA$)
5090 IF LEN(VA$)<6 THEN VA$="0"+VA$:
     GOTO 5090
5100 IF LEN(VA$) >6 THEN PRINT"[rev]
     MUST BE ONLY SIX(6) CHARACTERS
     LONG[off]":GOTO 5080
5110 Y$="":INPUT"[2x dwn]ENTER AS
     A PERMANENT CHANGE (Y/N)";Y$:
     IF Y$="N" THEN RETURN
5120 IF Y$<>"Y" THEN PRINT""[2x up]
     "S$"[2x up]":GOTO 5110
5130 PRINT"[dwn][5x rht][rev]WAIT,
     CHANGING DATA LINE[off]"
5140 FLAG=1
5150 D=PEEK(65)+256*PEEK(66):I=0
5160 IF PEEK(D)=131 THEN 5180
5170 D=D+1:GOTO 5160
5180 IF PEEK(D)=44 THEN I=I+1:
     IF I=Z THEN 5200
5185 D=D+1:GOTO 5180
5190 STOP
5200 REM **
5210 FOR X=1 TO 6:
     POKE(D+X),ASC(MID$(VA$,X,1)):NEXT X
5220 SAVE"@0:PAYROLL",8:
     REM ** SAVE CHANGES **
5230 GOTO 5000
```

Chapter SEVEN

STATISTICS AND THE CLASSROOM

Statistics have so many uses that it is difficult to decide which chapter should include the statistics modules. Placing them in the chapter with classroom use is a purely arbitrary decision and should not detract from their usefulness in other disciplines.

In this chapter we have included a number of statistic functions which are relatively simple and which can be generally useful. Although we will refer to the individual data points as "scores," you may interpret scores to mean sales, or number of interviews, or any other set of values that will fit your parameters.

The major programs included in this chapter are a grade book and a program that will construct objective tests for you. Both programs are most useful and contain modules that can be used in other programs of your own design.

To use the statistics modules, we must have some understanding of the ways data is usually grouped. In most of

the statistics modules in this book the data is assumed to be in an array which is some form of frequency distribution.

FREQUENCY DISTRIBUTIONS AND GROUPED DATA

When we talk about the *frequency distribution* of data, we are simply talking about how many times (the frequency) a value appears in our data. For example, suppose that we have a set of data like that shown in Table 7-1. The array of data shown in the table does not have any data whose value is 1 or 6, has one each of 0, 2, 4, and 9, has two each of 3 and 8, has three 7s, and four 5s. This counting of how many there are of each value is the frequency distribution of our data. In a table it looks something like Table 7-2.

Tables 7-1 and 7-2 show the same data in different forms. Table 7-2 shows that for the score of 0, there is a frequency of one (there is one zero). For the score of 1, there is a frequency of zero (there are no 1s in the data); and so on.

Module 7/1A will take an array of data, such as the one in Table 7-1, and convert it to a frequency distribution array, such as the one shown in Table 7-2. This conversion

Table 7-1 **Array of Data**

A(1) = 8	A(6) = 5	A(11) = 3
A(2) = 0	A(7) = 4	A(12) = 9
A(3) = 5	A(8) = 5	A(13) = 7
A(4) = 7	A(9) = 2	A(14) = 7
A(5) = 8	A(10) = 5	A(15) = 3

Table 7-2 **Frequency Distribution of Data**

B(x,1) (score)	B(x,2) (frequency)
0	1
1	0
2	1
3	2
4	1
5	4
6	0
7	3
8	2
9	1

requires that we provide N (the number of scores in the array), MN (the minimum score in the array), and MX (the maximum score in the array). The conversion also DIMensions an array, B(R,2). The value of R is determined by the range (maximum value minus minimum value) of scores in the data. The value of R is not critical as long as it is larger than the range of the score values. If we expect to use this module in a program where the module will be used more than once, the array should be dimensioned at the beginning of the program and should be dimensioned large enough to accommodate the largest range of data used in the program.

The module sets B(1,1) equal to MN. It then runs through the data, and every time a score is found equal to that of B(1,1) a one is added to B(1,2). B(2,1) is then set equal to MN+1, and the data is again run through and a one is added to B(2,2) every time the value of B(2,1) is

equaled. This process goes on until B(R,1) is set equal to MX, and B(R,2) is incremented for each score matching B(R,1). B(x,1) is the list of the score values and B(x,2) is the corresponding frequency of the scores. It should be noted that all scores are assumed to be integer values, since decimals will usually provide too many points between the integers. Decimal fraction scores can be grouped using Module 7/1B.

A normal frequency distribution has an interval of one. That is, each frequency measurement covers only one number. The frequency of 5s in the data tells how many values between 4.5 and 5.5 are found in the data. The distance between 4.5 and 5.5 is one; thus the interval for this normal frequency distribution is one. Grouped frequency distributions have intervals larger than one. If we group our data using an interval of three, all the values from −1.5 to +1.5 are grouped as zero. All the values from 1.5 to 4.5 are grouped as three, all the values from 4.5 to 7.5 are grouped as six, and so on. A table of our example values in a grouped frequency distribution, having an interval of three, would look as shown in Table 7-3.

Module 7/1B will take an array of data such as that shown in Table 7-1, and produce a grouped frequency

Table 7-3 **Grouped Frequency Distribution**

B(x,1) (upper real limit)	B(x,2) (frequency)
10.5	3
7.5	8
4.5	2
1.5	2

distribution of the data such as that shown in Table 7-3. A grouped frequency distribution is often a convenient way to list data since it is a condensed form of the data.

AVERAGES AND RANKING

As we know, "average" can mean different things in different circumstances. It can mean the score most of us make, in which case it is called the *mode*. It can mean the score halfway between the lowest and the highest score, called the *median*. Or it may be the sum of the scores divided by the number of scores, the *mean*. These three functions (mode, median, and mean) are said to be indicators of the *central tendency* of a frequency distribution of scores, and indicate the average value of the scores.

Module 7/2A will take a group of scores in an array A(x) and the number of scores in the array, and will calculate the mode, median, and mean for the array. The calculation of the median may be slightly in error if there is more than one score in the array equal to the median score. The next module (7/2B) calculates a better value of median in all cases.

Module 7/2B calculates the median for a grouped frequency distribution. An input array, different from that for Module 7/2A, is required. Module 7/2B requires an input array such as the one provided by Module 7/1B. The median of an ungrouped frequency distribution can be obtained by using an interval of one with Module 7/1B and then running the resulting array through Module 7/2B.

Modules 7/3A and 7/3B calculate the percentile rank of a score in an ungrouped and in a grouped frequency distribution. Of course, the percentile rank is the percent

of scores smaller than the score being ranked. A score with a percentile rank of 37 percent means that 63 percent of the scores are higher than the ranked score.

STANDARD DEVIATION, Z SCORES, AND REGRESSION

Standard deviation is the square root of a value called *variance.* Both standard deviation and variance are measures of the variability of a set of scores. They are values which, when compared between distributions, can indicate which distribution is the more variable. The larger the standard deviation (or variance), the wider the variability of the scores in the distribution. Module 7/4 calculates the mean, variance, and standard deviation.

Z scores are sometimes called *curved scores* since they convert the raw scores into a standardized form. A Z score is obtained by subtracting the average score (mean) from the raw score and then dividing by the standard deviation of the scores. This gives a score in terms of its deviation above or below the mean of the group of scores—a useful value. It should be noted that Module 7/4 also dimensions an array. The DIM statement may need to be placed at the beginning of the program and cleared before each use. This clearing can be accomplished by replacing line 510 with

```
510 FOR X=1 TO N:Z(X)=0:NEXT X
```

As indicated later in the program listing, Module 7/5 uses Module 7/4 as a subroutine. We can, however, insert Module 7/4 in place of line 520 in Module 7/5.

Another very useful tool is the regression function. A

regression function will derive a formula from your existing data which will predict where other data will be found. Modules 7/6A, 7/6B, 7/6C, and 7/6D are each different versions of the regression function.

Module 7/6A will take two related sets of data which have a linear relationship (such as height and weight) and calculate the linear equation that most nearly describes the relationship. The linear equation that this module derives is of the form Y = B*X+C, where Y is one set of data and X is the other. Module 7/6A calculates B and C. When B and C are known, a value of Y can be predicted for any known value of X. This means that when using this module we could predict, with some accuracy, a weight for a given height. The value of R2 is an estimate of the accuracy of our prediction.

Module 7/6A is useful only for those sets of data that are linearly related. Because most of the relationships that we will want to predict will be more complex than a straight line (linear), we need more than one regression formula. Module 7/6B is called an *exponential regression.* This module will take your existing data and fit it into a formula of the form Y = C*EXP(B*X). The module will calculate B and C from the data you supply. Module 7/6C does the same thing for a logarithmic regression, using the form model Y = C+B*LN(X). Module 7/6D is the power regression and uses the model Y = C*X ↑ B.

These four modules (7/6A, 7/6B, 7/6C, 7/6D) are quite similar and can be combined into one program. Such a program will allow us to run our data through all four regression formulas. We can then determine which type of regression is the best fit for our data by comparing the R2s. The regression with the largest R2 value will be the best fit.

It should be noted that logarithms and square roots

cannot be used with negative numbers. Thus it might be wise to make some provision for the program to stop if a negative number is encountered.

GRADE BOOK WITH STATISTICAL CALCULATIONS

Although we will call this program a grade book, it has some other obvious uses. It can be used with any listing of names and associated achievements that can be quantized into numbers. It will work with sales figures, percent of goal achievement, points scored per game, calls per day, or any number of other items. Your own list of uses will far exceed any list that we could make here.

Changing this program to fit your own needs is easy. All that is required is that the names in the program prompts be changed. When the program asks for the maximum possible grade, you will want to change it to ask for the week's goal, or the sales quota, or the total points scored in the game, and so on. You can easily make it fit your needs.

To build a good grade book program, we need to decide what we want it to do. It should list the names of the students in the class, each student's grades, and each student's average of the grades to date. Now, that's not too difficult! It should also provide curved grades (Z scores) for each student's grade. We should include a test evaluation. Test evaluation will provide a listing of the students in order of their ranking on the test and the mean and standard deviation of the test grades.

So, that's what a grade book program will do. What else must it do to work? Well, we must be able to add new students. We should also be able to make changes in the

students' records, in case we should happen to make a mistake. (I am sure that you never do, but we need this feature.) In addition, we will need a way to print out a hard copy of the students' records. We also will need to be able to save the records and to load them back into the computer by file name. Now, that should do it!

The modules of this program are numbered in the order in which they will appear in our program. In other words, those modules with the smaller line numbers have the smaller module numbers. We do it this way to make it easier to assemble the modules into the proper sequence. When we discuss the modules, however, we will follow the logical function order rather than the numerical sequence order.

Initial

Our initial module (7/7) is little more than a list of DIMensioning statements and some string variables to be used between the students' printout. The dimensions allow for up to 100 students and up to 50 grades per student. That should be enough for most needs.

The student's records are stored in C$(100,50). The first zero dimension [C$(0,x)], keeps track of how many sets of scores have been entered. The second zero dimension [C$(s,0)] is the student's name (s = the student's number). All of the student's grades are listed in the second dimension, with the student's number in the first dimension. For example, the fourth grade for student 15 will be stored in C$(15,4). Of course, the student's name is stored in C$(15,0).

The array P(50) is used as the numerical storage of the maximum possible grade for each test. These maximum scores are stored in string form in C$(0,50). The array

A$(100) is used in some of the modules for temporary storage.

The second module (7/8) can be considered to be part of the initial module. It loads the data from the disk or tape. If no file name is entered (FL$=""), the computer assumes that we wish to start a new class roll. If a file name is entered, the file is loaded and the values of P(50) are assigned.

Menu

Now that we have the initial values set and the data loaded, we need a menu of functions. Module 7/18 is our old friend Module 3/14. The only difference between Module 7/18 and Module 3/14 is line 1020. If no data file has been entered, this line sends the computer to Module 7/15 to start a new class.

The menu lists nine functions:

1. Add new student
2. Add new grades
3. List all grades
4. List/change a student grade
5. List grade distributions
6. List curved grades
7. Start a new class
8. Print program
9. Quit program

It should be noted that with the exception of function 6 (list curved grades), the line numbers of the function modules correspond to the menu numbers. Function 1 is

listed, starting with line 100, function 2 with line 200, and so on. Both function 5 and function 6 begin with line 500 and print the grade distribution. When the grade distribution is completed, function 5 returns, but function 6 jumps to line 600 to calculate and print Z scores before returning. The difference between listing grade distributions (5) and listing curved grades (6) is that the curved grades lists the Z scores of each grade. There may be times when we wish to print out the grade distribution without printing the Z scores. Function 5 allows us to do this.

The quit program, function 9, should always be used to exit the program. It checks to see if any changes in the file data have been made, and if so, saves the new file data before exiting the program. Making a habit of using the quit program function will avoid many oversight errors.

Of course, the menu names can be changed to fit your needs. For example, to use the program as a record of worker efficiency, the word "student" can be changed to "employee" or "worker"; "grade" can be changed to "score"; and so on.

Starting the Class

Module 7/15 first checks to determine whether or not there is a class currently in memory. If there is a class in memory, it asks whether the existing class needs to be saved before erasing memory to start the new class. Depending on the response, the module will either save the class data and then erase it or will simply erase the class data. Whatever the initial condition, the module will prepare itself to accept a new class. It will then allow us to enter a number of initial names, which are the new class. Starting a new class is the only time that we will be able to enter a LIST of names easily. We can enter new students later but only one at a time.

Now that we have the class listed, we can begin to enter grades. Module 7/10 keeps track of how many grades have been entered and allows us to enter the next grade for each student and the maximum possible grade for each set of entries. The computer asks for the "possible score." This is the maximum score possible. If there were eight questions and we are giving one point for each question, the possible score is eight. If there are eight questions and we are grading on a basis of percent of perfect (100 percent is maximum) the possible score is 100. The student's scores are now entered. If a student misses a test, the student may be given a zero, in which case it will be treated as a valid score and averaged with the student's other scores. Alternatively, if a minus sign is entered as the student's grade, that grade is not counted in the student's average. Either of these grades can be changed later if circumstances require a change.

To list the grades, we call on Module 7/11. This module uses a part of Module 7/12 to list the grades. Each student is listed by name and number, each of the student's grades is listed (together with the grade's percent of maximum), and the student's grade average is shown. After the grades of all the students have been displayed, the average for the class is calculated and displayed.

Making Changes

To view or change a certain student's record, Module 7/12 is called. When using this module, a student must be identified by number. The student's name and grades are displayed. If changes are desired, the number of the score to be changed is entered. The student's name can even be changed, if desired. If no changes are needed, enter the RETURN key. Doing this will take us back to the menu.

Module 7/9 will add a new student to the class. What can we do about grades of tests that the new student has missed? A minus sign is given to the new student for each test for which other students have grades. This minus sign is not counted in the new student's grade average. If one or more of the grades are to be made up or counted as zeros, these changes can be made by using the "LST/CNG A STUDNT GRAD" function from the menu (Module 7/12).

Grade Distributions

Module 7/13 asks which test is to be evaluated, and then, for that test, lists in ascending grade order the student's name, grade, and the grade's percent. At the end of the listing, the module prints the mean (average) and standard deviation of the class scores on this test.

Module 7/14 adds Z scores to the listing of Module 7/13. The grade list and Z score functions are separated into two modules, so that, if desired, the hard-copy printout can list grades and percentages without Z scores. The Z score is calculated using a modified version of Module 7/5.

Module 7/13 uses Module 7/19 to sort the students into ascending grade order. Module 7/19 has many other uses. It is called a *bubble sort* and will sort a list of values into order. Module 7/13 also uses Variance and Standard Deviation Module 7/4 from earlier in this chapter. Because a number of modifications of Module 7/4 are required, the modified version is listed as Module 7/20. The value of the mean and the standard deviation are obtained from this module. If you compare Module 7/20 with Module 7/4, the similarity will be obvious. Such a comparison will help to illustrate how modifications can be made to modules to make them more useful.

Print and Quit Programs

Print Program Module 7/16 is similar to those in other programs in this book. The "CMD" print command is used so that the modules of the program can be used both to "print" to the monitor and to "print" to the printer without writing special programs for the printer. All the modules are listed as subroutines, so they can be used either from the menu or from the print program equally well.

Quit Program Module 7/17 is designed to help us remember to save any changes that we have made in the grade book. It asks whether to save on tape or disk and what file name to use. It then saves the data in the named file, closes the file, and stops the program.

Assembling the Grade Book

The grade book modules should be entered in the following order:

7/7 Initial
7/8 Load data
7/9 Add new student
7/10 Add new grades
7/11 List all grades
7/12 List/change a student grade
7/13 Grade distribution
7/14 Curved grades
7/15 Start new class
7/16 Print program
7/17 Quit program
7/18 Grade book menu

```
7/19 Bubble sort
7/20 Mean and standard deviation
```

There are many variations that could be applied to this program. If desired, the students could be listed on cards using the card file modules from Chap. 4. It would be simple to add a plotting module which could plot the grade distribution or a student's progress. Additional statistical modules could be added to provide research data or to help track the quality of the testing itself. Again, the possibilities are numerous.

QUIZ DESIGNER

In order to have grades to record in a grade book, one must give tests. To help with this process, perhaps a program to develop the quiz is in order.

The following simple program is meant to be used to develop objective tests. It is not divided into modules. It is listed as a complete program. The program asks how many questions will be on the quiz and how many answer choices there will be for each question. It then accepts your input of the questions, the answer choices, and the correct answer for each question. It will then print the test with or without the correct answer being indicated.

Because the program is designed to use both upper- and lowercase letters in its text, the program is written in the same manner. To enter the program correctly, your computer should be in the text or nongraphic mode. This is done by pressing both the Commodore key and the SHIFT key at the same time.

```
010 s$=" [25x cmd/t] "
020 print chr$(14):print" [clear]
    [2x dwn][7x spc] OBJECTIVE TEST
    DESIGNER"
030 print" [6x spc] "s$:print
040 input"Enter the number of questions
    you plan";n
050 input"How many answers per
    question";m
060 dim a$(n,m+2)
100 for x=1 to n:q=1:print"[clear]
    Enter Question Number "x:print
110 get b$:print "[cmd/+][lft]";:
    if b$="" then 110
120 if b$=chr$(13) then print:<=0:goto 200
130 a$(x,1)=a$(x,1)+b$:c=c+1:
    if c=250 then150
135 if c=255 then b$=chr$(13):goto 120
140 print b$;:goto 110
150 print "[rvs]only 10 more characters
    [off]":goto 140
200 for q=2 to m+1
210 print "Enter answer number" q-1
220 input a$(x,q)
230 next q
240 input "Enter the number of the correct
    answer";a$(x,m+2)
250 next x
300 z$="":input "Print the test
    (y/n)";z$
```

```
310 if z$="n" then 400
320 if z$<>"y" then 300
330 z$="":input "Print test or test
    master (t/m)";z$:if not (z$="t" or
    z$="m") then 330
340 flag=0:if z$="m" then flag=1
350 open 2,4:cmd 2:print,chr$(17)
360 for x=1 to n:print s$:print
    " [5x spc] Question Number "x
370 print:print a$(x,1):print
375 for q=2 to m+1
380 print tab(10) q-1 "- " a$(x,q);:
    if flag=1 and q-1=val(a$(x,m+2))
    then print "**";
385 print:next q:print:print s$:next x
390 print#2:close 2
400 stop
```

This program will work just as it is shown. Nevertheless, you may want to add a few items to make it more useful. Probably you will want to add a module that will save the test for later use. This module could be added, beginning at line 400. You will also want to add a module that will load the saved file so that it can be used. There may be other modules which you will want to add. Here is a program on which you can practice your customizing skills.

CHAPTER SEVEN
MODULES

MODULE 7/1A Will take an array of raw scores and condense them into a frequency distribution array of the scores. The raw scores are input in array A(x). You must furnish the number of scores in the array (N), the maximum score in the array (MX), and the minimum score in the array (MN). You may set the maximum and minimum arbitrarily as long as MX is equal to or larger than the maximum score and MN is equal to or smaller than the minimum score. The output is in array B(x,n), where B(x,1) is the score and B(x,2) is the frequency of the score. Notice that B(R,2) is DIMensioned in line 120. If you are using Module 7/1A in a way that would cause a redimensioning error, the array must be dimensioned to an appropriate size in the initial module of your program.

```
100 REM ** 7/1A-FREQ DISTRIBUTION **
110 R=MX-MN+1
120 DIM B(R,2)
130 FOR I=MN TO MX:B(I-MN+1,1)=I
140 FOR J=1 TO N
150 IF A(J)=B(I-MN+1,1) THEN
    B(I-MN+1,2)=B(I-MN+1,2)+1
160 NEXT J:NEXT I
```

MODULE 7/1B Condenses a set of raw scores into a grouped frequency distribution. The module is similar to 7/1A but groups the distributions within score intervals.

The module requires the same input variables as 7/1A, plus the interval width of the grouping (W). The output is also similar, except that B(x,1) is the array of upper real limits of the interval values and R is the number of groups.

```
100 REM ** 7/1B-GROUPED FREQ DISTR **
110 R=INT((MX-MN)/W)+1:T=MX+0.5:B=T-W
120 DIM B(R,2)
130 FOR I=1 TO R:B(I,1)=T
140 FOR J=1 TO N
150 IF A(J)<B OR A(J)>T THEN GOTO 170
160 B(I,2)=B(I,2)+1
170 NEXT J
180 B=B-W:T=T-W
190 NEXT I
```

MODULE 7/2A Accepts an array of scores, A(x), and the number of scores in the array (N). It then outputs the mean (M1), median (M2), the mode (M3), and the number of modes (K). In the case where there is more than one score equal to the median (M2), the value of M2 may be slightly in error.

```
200 REM ** 7/2A-MEAN, MEDIAN, MODE **
210 SUM=0:M3=0:K=0
220 FOR I=1 TO N:SUM=SUM+S(I)
230 IF A(I)=M3 THEN K=K+1
240 IF A(I)>M3 THEN K=1
250 IF A(I)>M3 THEN M3=A(I)
260 NEXT I
```

```
270 IF N/2 <> INT(N/2) THEN
    M2=A(INT(N/2)+1)
280 IF N/2=INT(N/2) THEN M2=A(N/2)
290 M1=INT(100*(SUM/N)+.5)/100
```

MODULE 7/2B Provides the median for a grouped frequency distribution. Requires the array of upper real limits of possible scores, B(x,1); the array of the frequency of scores, B(x,2); and the number of groups (R). The output is M2, the median value.

```
200 REM ** 7/2B-GROUPED MEDIAN **
210 N=0:CF=0:W=B(2,1)-B(1,1)
220 FOR I=1 TO R
230 IF I<=R/2 THEN CF=CF+B(I,2)
240 N=N+B(I,2)
250 NEXT I
260 M2=B(R/2,1)+W*(0.5*N-CF)/B(R/2+1,2)
```

MODULE 7/3A Calculates the percentile rank for a score in a nongrouped frequency distribution. The module requires the array, A(x), the number of scores in the array (N), and the score to be ranked (X). It returns the rank of the score (PR) and the cumulative frequency (CF) of the position of the ranked score.

```
300 REM ** 7/3A-PERCENTILE RANK **
310 R=0:CF=0
320 FOR I=1 TO N
330 IF A(I) < X THEN CF=CF+1
340 IF A(I)=X THEN R=R+1
```

```
350 NEXT I
360 CF=CF+R/2:PR=100*CF/N
```

MODULE 7/3B Calculates the percentile rank for a score in a grouped frequency distribution. It requires the array of grouped scores and upper limit values, B(x,2), the number of groups in the array, R, and the score to be ranked, X. It then returns the rank of the score, PR, and the cumulative frequency, CF, of the position of the ranked score.

```
300 REM ** 7/3B-GROUPED PERCENTILE **
310 W=B(2,1)-B(1,1):CF=0:N=0
320 FOR I=1 TO R:N=N+B(I,2)
330 IF B(I,1) < X THEN CF=CF+B(I,2)
340 IF B(I,1) >= X AND B(I,1)-W < X
    THEN CF=CF+((X-B(I-1,2)+W)/W)*B(I,2)
350 NEXT I:PR=100*CF/N
```

MODULE 7/4 Requires only the array of scores, A(x), and the number of scores in the array, N. It returns the mean of the scores (AVG), the variance of the set of scores (VAR), and the standard deviation of the scores (SD).

```
400 REM ** 7/4-VAR & STD DEVIATION **
410 SUM=0:SQS=0
420 FOR I=1 TO N
430 SUM=SUM+A(I):SQS=SQS+A(I)*A(I)
440 NEXT I
450 AVG=SUM/N
460 VAR=(SQS/N)-AVG*AVG
```

```
470 SD=INT(100*SQR(VAR)+.5)/100
480 AVG=INT(100*AVG+.5)/100
490 VAR=INT(100*VAR+.5)/100
```

MODULE 7/5 The Z-score calculator, which uses Module 7/4 for the variance and standard deviation required. All that is required to make Module 7/4 a subroutine is to add ":RETURN" on the end of line 470. If you do not want to use Module 7/4 as a subroutine, you can insert it into Module 7/5 in place of the GOSUB. Notice that line 510 DIMs the array Z(N). This means that unless this module will be used only once in your program, the DIMensioning of the Z(N) array must be done in the initial program.

```
500 REM ** 7/5-Z-SCORE **
510 DIM Z(N)
520 GOSUB 400:REM ** TO MODULE 7/4 *
530 FOR I=1 TO N
540 Z(I)=(A(I)-AVG)/SD
550 NEXT I
```

MODULE 7/6A Calculates the linear equation that most nearly describes the relationship between two sets of paired values. It also calculates the Pearson correlation coefficient. The required input is the array B(x,2) of paired values and the number (N) of paired values in the array. Note that R2 is an estimate of the percent validity of the prediction line: Y = B*X+C.

```
600 REM ** 7/6A-LINEAR REGRESSION **
610 SX=0:SY=0:XX=0:YY=0:XY=0
620 FOR I=1 TO N
```

```
630 SX=SX+B(I,1):SY=SY+B(I,2)
640 XX=XX+B(I,1)↑2:YY=YY+B(I,2)↑2
650 XY=XY+B(I,1)*B(I,2)
660 NEXT I
670 B=(N*XY-SX*SY)/(N*XX-SX*SX)
680 C=(SY-B*SX)/N
690 R2=((XY-(SX*SY)/N)↑2)/((XX-(SX↑2)/N)
    *(YY-(SY↑2)/N))
```

MODULE 7/6B Uses an exponential model for comparison between a set of paired values. The model is Y = C*EXP(B*X).

```
700 REM ** 7/6B-EXPONENTIAL REGRESS **
710 SX=0:SY=0:XX=0:YY=0:XY=0:DIM Y(N)
720 FOR I=1 TO N:Y(I)=LOG(B(I,2)):NEXT I
725 FOR I=1 TO N
730 SX=SX+B(I,1):SY=SY+Y(I)
740 XX=XX+B(I,1)↑2:YY=YY+Y(I)↑2
750 XY=XY+B(I,1)*Y(I)
760 NEXT I
770 B=(N*XY-SX*SY)/(N*XX-SX*SX)
780 C=EXP((SY-B*SX)/N)
790 R2=((XY-(SX*SY)/N)↑2)/((XX-(SX↑2)/N)
    *(YY-(SY↑2)/N))
```

MODULE 7/6C Uses a logarithmic model of the form Y = C+B*LN(X).

```
800 REM ** 7/6C-LOGARITHMIC REGRESS **
810 SX=0:SY=0:XX=0:YY=0:XY=0:DIM X(N)
820 FOR I=1 TO N:X(I)=LOG(B(I,1)):NEXT I
```

```
825 FOR I=1 TO N
830 SX=SX+X(I):SY=SY+B(I,2)
840 XX=XX+X(I)↑2:YY=YY+B(I,2)↑2
850 XY=XY+X(I)*B(I,2)
860 NEXT I
870 B=(N*XY-SX*SY)/(N*XX-SX*SX)
880 C=(SY-B*SX)/N
890 R2=((XY-(SX*SY)/N)↑2)/((XX-(SX↑2)/N)
    *(YY-(SY↑2)/N))
```

MODULE 7/6D Uses a power model of the form Y = C*X↑B.

```
900 REM ** 7/6D-POWER REGRESSION **
910 SX=0:SY=0:XX=0:YY=0:XY=0:
    DIM X(N),Y(N)
920 FOR I=1 TO N:X(I)=LOG(B(I,1)):
    Y(I)=LOG(B(I,2)):NEXT I
925 FOR I=1 TO N
930 SX=SX+X(I):SY=SY+Y(I)
940 XX=XX+X(I)↑2:YY=YY+Y(I)↑2
950 XY=XY+X(I)*Y(I)
960 NEXT I
970 B=(N*XY-SX*SY)/(N*XX-SX*SX)
980 C=EXP((SY-B*SX)/N)
990 R2=((XY-(SX*SY)/N)↑2)/((XX-(SX↑2)/N)
    *(YY-(SY↑2)/N))
```

MODULE 7/7 The initial module of the grade book program. It DIMensions the needed arrays and provides lines and spaces in string variables.

```
010 REM ** 7/7-INITIAL GRADE BOOK **
020 DIM C$(100,50),P(50),A$(100):
    C$(0,0)="0":C$(0,1)="*":FILL=0
025 S$=" [39x spc] "
030 TB$=" [39x cmd/t] "
035 BT$=" [39x cmd/p] "
```

MODULE 7/8 Asks for a file name, and if one is given, loads it. If no file name is entered (return is entered), the program goes to line 1000 (Menu Module 7/18).

```
040 REM ** 7/8-LOAD GRADE BOOK FILE **
050 INPUT"[clear][2x dwn][2x rht] ENTER
    NAME OF GRADE FILE WANTED [home]
    [3x dwn]";FL$:IF FL$="" THEN 1000
053 OPEN 2,8,2,"0:"+FL$+",S,R"
055 FOR N=0 TO 100:FOR G=0 TO 50
057 IF N>0 AND C$(0,G)="*" THEN 75
060 INPUT#2,C$(N,G):IF C$(N,0)="0"
    THEN 80
065 IF C$(0,G)="*"THEN 75
070 NEXT G
075 NEXT N
080 CLOSE 2:FILL=1
085 FOR G=1 TO 50:IF C$(0,G)="*" THEN 95
090 P(G)=VAL(C$(0,G):NEXT G
095 GOTO 1000
```

MODULE 7/9 Asks for the name of the new student and adds it to the student list. Any grade positions for the new student will be filled with minus signs so that they will not be counted on the student's record.

```
100 REM ** 7/9-ADD NEW STUDENT **
105 IF C$(1,0)="" THEN INPUT"[clear]
    [2x dwn] NO STUDENTS";Z$:RETURN
110 PRINT"[clear][3x dwn]":FOR N=1
    TO 100:IF C$(N,0)="*" THEN 130
120 NEXT N:PRINT"[3X rht] NO ROOM FOR
    NEW STUDENT":INPUT Z$:RETURN
130 PRINT"ENTER THE NAME OF ":
    PRINT"[3x rht] STUDENT NUMBER "N;:
    INPUT"[rht]";C$(N,0)
140 IF N<100 THEN C$(N+1,0)="*"
150 FOR G=1 TO 50:IF C$(0,G)="*"
    THEN 170
160 C$(N,G)="-":NEXT G
170 RETURN
```

MODULE 7/10 Scans the grade file and stops at the first empty grade space. It places the "possible" score in C$(0,G) (where G is the number that identifies the set of grades) and lists each student's name in order. The module then asks for each student's grade. If RETURN is entered, the grade recorded will be a zero.

```
200 REM ** 7/10-ADD NEW GRADES **
205 IF C$(1,0)="" THEN INPUT"[clear]
    [2x dwn] NO STUDENTS";Z$:RETURN
210 FOR G=1 TO 50:IF C$(0,G)="" THEN 220
215 NEXT G:PRINT"[clear][2x dwn][rht]
    BOOK FULL":STOP
220 C$(0,G)="*":G=G-1
230 PRINT"[clear][2x rht] THIS WILL BE
    GRADE NUMBER "G"[2x dwn]"
```

```
235 INPUT"[2x dwn][2x rht] ENTER THE
    POSSIBLE SCORE";C$(0,G):
    P(G)=VAL(C$(0,G))
240 FOR N=1 TO 100:IF C$(N,0)="*"
    THEN RETURN
250 PRINT N"[spc]"C$(N,0) TAB(25);:
    INPUT C$(N,G)
260 IF C$(N,G)="" THEN C$(N,G)="0"
270 NEXT N:RETURN
```

MODULE 7/11 Sets up the variables needed and then uses Module 7/12 to display each student's grades. It then calculates and displays the class average.

```
300 REM ** 7/11-LIST ALL GRADES **
310 PRINT"[clear]":X=0:TT=0:FOR N=1
    TO 100:IF C$(N,0)="*" THEN 360
320 CH$="":GOSUB 410:REM ** TO 7/12 **
330 IF CH$<>"" THEN RETURN
340 X=X+1:TT=PC+TT
350 NEXT N
360 PRINT TB$:PRINT"[2x dwn] CLASS
    AVERAGE=" INT(100*TT/X+.5)/100"%":
    PRINT BT$
370 IF FLAG=1 THEN RETURN
380 INPUT Z$:RETURN
```

MODULE 7/12 Asks for a student number, searches for that student, and displays the student's name, test grades, and grade average. If the grade on any test is a minus sign ("−"), that test is not included in the grade average. If the grade for any test is 0, that test *is* included in the grade

average. After the grades are listed, the listed data can be changed by entering the number of the score to be changed and then entering the new data. Module 7/11 uses the listing portion of this module as a subroutine to list grades.

```
400 REM ** 7/12-LST/CHG STUDENT GRADE**
405 INPUT"[clear][2x dwn][2x rht]
    ENTER STUDENT NUMBER";N
410 FOR I=1 TO 100:IF C$(I,0)="*"
    THEN 414
412 NEXT I
414 IF N > I-1 THEN RETURN
416 PRINT TB$:PRINT TAB(6)"#" N
    "[2x spc]" C$(N,0):TS=0:PC=0:
    TP=0:PRINT
420 FOR G=1 TO 50:IF C$(0,G)="*"
    THEN PRINT:GOTO 465
425 IF C$(N,G)="-" THEN 460
430 TS=TS+VAL(C$(N,G))
433 IF FLAG=1 THEN PRINT CHR$(16)"06" G
    "-[spc]" C$(N,G) "[spc]/" P(G);
435 IF FLAG=1 THEN PRINT CHR$(16)"23="
    INT(1000*VAL(C$(N,G))/P(G))/10 "%":
    GOTO 450
440 PRINT TAB(6) G "-[spc]" C$(N,G)
    "[spc]/" P(G) TAB(21)"="
    INT(1000*VAL(C$(N,G))/P(G))/10 "%"
450 TP=TP+P(G)
460 NEXT G
465 PRINT"TOTAL =" TS:PRINT"POSSIBLE ="
    TP:PC=INT(1000*TS/TP+.5)/10
```

```
470 PRINT"PERCENT =" PC:
    IF FLAG=1 THEN RETURN
473 PRINT"[2x dwn][2x rht]ENTER RETURN
    TO CONTINUE":IF VAL(Q$)<>4 THEN 485
475 CH$="":INPUT"[dwn][rht]ENTER SCORE
    TO CHANGE (0=NAME)";CH$:
    IF CH$="0" THEN 490
477 IF CH$="" THEN RETURN
480 V$="":INPUT"[spc]ENTER NEW SCORE
    VALUE";V$:C$(N,VAL(CH$))=V$:RETURN
485 INPUT CH$:RETURN
490 V$="":INPUT"[spc]ENTER NEW NAME";V$
    :C$(N,0)=V$:RETURN
```

MODULE 7/13 Displays a student's names and grades for any test in the order of the student's grade rank. Module 7/19 is used to sort the students into rank order, and Module 7/20 is used to obtain the mean and standard deviation for the test.

```
500 REM ** 7/13-GRADE DISTRIBUTION **
510 INPUT"[clear][2x dwn][2x rht]ENTER
    TEST NUMBER TO EVALUATE";Z:
    IF Z<1 THEN RETURN
520 FOR G=1 TO 50:IF C$(0,G)="*"
    THEN 530
525 NEXT G
530 G=G-1:IF Z>G THEN RETURN
535 PRINT TB$:PRINT TAB(6) "TEST #" Z:
    PRINT BT$
540 FOR N=1 TO 100:IF C$(N,0)="*"
    THEN 560
```

```
543 IF C$(N,Z)="-" THEN A$(N)="-":
    GOTO 550
545 A$(N)=STR$(INT(1000*VAL(C$(N,Z))/
    VAL(C$(0,Z))+.5)/10)
550 A$(N)=C$(N,0)+"[3x spc]"+A$(N)
555 NEXT N
560 N=N-1:GOSUB 2000:GOSUB 3000:
    PRINT TAB(20) "SCORES";
563 IF FLAG =1 THEN RETURN
565 IF VAL(Q$)=6 THEN PRINT
    TAB(29) "Z-SCORE";
570 PRINT:PRINT:FOR N=1 TO 100:
    IF C$(N,0)="*" THEN 585
580 PRINT N "[spc]" LEFT$(A$(N),
    LEN(A$(N))-5) TAB(20)
    RIGHT$(A$(N),4) "[spc]%";
581 IF RIGHT$(A$(N),1)="-" THEN 584
582 IF VAL(Q$)=6 THEN PRINT TAB(30)
    INT(100*(VAL(RIGHT$(A$(N),4))-AVG)
    /SD)/100;
584 PRINT:NEXT N
585 PRINT:PRINT "MEAN =" AVG,
    "STD DEV =" SD
590 INPUT"[dwn][2x rht]RETURN TO
    CONTINUE";Z$;RETURN
```

MODULE 7/14 Calculates Z scores using the mean and standard deviation values obtained from Module 7/20. The ranking and listing of the raw scores is done by using the listing part of the Module 7/13 program.

```
600 REM ** 7/14-Z SCORES (CURVE GRADE)**
610 GOSUB 520
620 PRINT CHR$(16) "29Z-SCORE"
630 PRINT:FOR N=1 TO 100:
    IF C$(N,0)="*" THEN 680
640 PRINT N "[spc]" LEFT$(A$(N),
    LEN(A$(N))-5) CHR$(16) "20"
    RIGHT$(A$(N),4) "[spc]%";
650 IF RIGHT$(A$(N),1)="-" THEN 670
660 PRINT CHR$(16) "30" INT(100*(VAL
    (RIGHT$(A$(N),4))-AVG)/SD)/100;
670 PRINT:NEXT N
680 PRINT:PRINT "MEAN =" AVG,
    "STD DEV =" SD:PRINT TB$:RETURN
```

MODULE 7/15 Allows us to enter a number of students' names at one time. This is helpful when starting a new class register. The module keeps us from accidentally erasing a class in memory because it checks to be sure that the memory is clear before allowing us to continue.

```
700 REM ** 7/15-START NEW CLASS **
705 Z$="":PRINT "[clear][3x dwn]
    [3x rht]";:IF FILL=0 THEN 720
710 PRINT "THERE IS A CLASS IN MEMORY":
    INPUT "[5x rht]SAVE IT (Y/N)";Z$
715 IF Z$<>"N" THEN GOSUB 900:FILL=0:
    GOTO 700:REM ** TO SAVE ROUTINE **
720 PRINT "[3x lft]" TB$:PRINT TAB(8)"
    [rev][2x spc]NEW CLASS[2x spc]
    [rev off]":PRINT BT$
```

```
725 PRINT "ENTER THE NAMES OF THE
    STUDENTS":PRINT "[4x rht]
    ENTER * TO RETURN TO MENU"
730 PRINT:PRINT:FILL=1:FOR N=1 TO 100
740 PRINT "STUDENT NUMBER "N;:
    INPUT"[spc]";C$(N,0)
750 IF C$(N,0)="*" THEN RETURN
760 PRINT "[up]" S$ "[up]"
770 NEXT N
780 PRINT "TOTAL OF 100 STUDENTS
    FILLS BOOK":INPUT Z$:RETURN
```

MODULE 7/16 The print program module, which allows us to select from the grade book any of four printing options.

```
800 REM ** 7/16-PRINT PROGRAM **
810 PRINT "[clear][dwn]" TB$:PRINT
    TAB(8) "PRINT ROUTINE":PRINT BT$
820 PRINT "[dwn][7x rht] 1 LIST ALL
    GRADES"
825 PRINT "[7x rht] 2 LIST A STUDENT'S
    GRADES"
830 PRINT "[7x rht] 3 LIST A TEST
    DISTRIBUTION"
835 PRINT "[7x rht] 4 RETURN TO MENU"
840 INPUT "[2x dwn][2x rht]ENTER ITEM
    TO BE PRINTED";Z$
850 IF VAL(Z$)=4 THEN RETURN
860 IF VAL(Z$)=2 THEN INPUT "[clear]
    [2x dwn][2x rht][rev]ENTER
    STUDENT NUMBER[off]";N
```

```
865 IF VAL(Z$)=3 THEN INPUT "[clear]
    [2x dwn][2x rht][rev]ENTER TEST
    NUMBER TO EVALUATE[off]";Z
870 OPEN 2,4:CMD 2:FLAG=1
880 ON VAL(Z$) GOSUB 300,410,600
890 PRINT#2:CLOSE 2:FLAG=0:RETURN
```

MODULE 7/17 The quit program module, which helps us to make certain that any changes to the data are preserved. Different classes can be saved using different file names, such as "Class 1," "Class 2," or "History," "Math '85," and so on. If desired, the command RUN in line 990 can be changed to NEW. This change will erase the program from memory as soon as the data is saved.

```
900 REM ** 7/17-QUIT PROGRAM **
910 Z$="":INPUT "[clear][2x dwn][2x rht]
    SAVE GRADE BOOK ON DISK OR TAPE
    ( [rev] D [off] /T)";Z$
920 D=8:IF Z$="T" THEN D=1
930 INPUT "[dwn][2x rht]ENTER THE FILE
    NAME FOR THE GRADES";FL$:
    FL$="@0:"+FL$
940 OPEN 2,D,2,FL$+",S,W"
950 FOR N=0 TO 100:FOR G=0 TO 50
960 IF N>0 AND C$(0,G)="*" THEN 980
965 PRINT#2,C$(N,G)
970 IF C$(0,G)="*" THEN 980
975 NEXT G
980 IF C$(N,0)="*" THEN 990
985 NEXT N
990 PRINT#2,C$(N,0):CLOSE 2:RUN
```

MODULE 7/18 Our old standby menu from Chap. 3, Module 3/14. The only reason for listing this menu module in this chapter is to make it easier to enter the complete grade book program without referring to other chapters.

```
1000 REM ** 7/18-GRADE BOOK MENU **
1010 DIM M$(20)
1011 M$(1)="ADD NEW STUDENT":M$(2)="ADD
     NEW GRADES"
1012 M$(3)="LIST ALL GRADES":M$(4)=
     "LST/CNG A STUDNT GRAD"
1013 M$(5)="LIST GRADE DISTRIB":M$(6)=
     "LIST CURVED GRADES"
1014 M$(7)="START NEW CLASS":M$(8)=
     "PRINT PROGRAM"
1015 M$(9)="QUIT PROGRAM"
1020 IF FL$="" THEN Z$="7":GOTO 1090
1025 N=9:PRINT"[clear]"
1030 TP$="[4x spc][30x cmd/+]"
1040 PRINT TP$:PRINT"[4x spc][cmd/+]"
     TAB(33) "[cmd/+]":FOR X=1 TO N+2:
     PRINT"[4x spc][cmd/+]";
1050 IF X<N+1 THEN PRINT
     TAB(28-LEN(M$(X))) M$(X) "=" X
     TAB(33) "[cmd/+]":GOTO 1060
1055 PRINT TAB(33) "[cmd/+]"
1060 NEXT X:PRINT TP$
1070 PRINT "[2x up]" TAB(7);:
     INPUT "CHOOSE ONE";Q$
```

```
1080 IF VAL(Q$)<1 OR VAL(Q$)>N THEN
     PRINT "[up]" TAB(18) "[4x spc][dwn]":
     GOTO 1070
1090 ON VAL(Q$) GOSUB 100,200,300,
     500,500,700,800,900:GOTO 1025
```

MODULE 7/19 A sorting routine which has many uses. lines 2015 and 2017 allow the routine to skip grades that are not to be counted.

```
2000 REM ** 7/19-BUBBLE SORT **
2010 FOR I=1 TO N-1:FOR J=I TO N
2015 IF RIGHT$(A$(I),3)="[2x spc]-"
     THEN 2040
2017 IF RIGHT$(A$(J),3)="[2x spc]-"
     THEN 2040
2020 IF VAL(RIGHT$(A$(J),3)) >
     VAL(RIGHT$(A$(I),3)) THEN 2040
2030 X$=A$(I):A$(I)=A$(J):A$(J)=X$
2040 NEXT J:NEXT I
2050 RETURN
```

MODULE 7/20 Calculates the mean (AVG) and standard deviation (SD) of an array of values A$(N). The number of values in the array must be given as N.

```
3000 REM ** 7/20-MEAN & STD DEVIATION**
3010 SUM=0:SQS=0:X=0:SD=0
3020 FOR I=1 TO N:A=0:
     IF RIGHT$(A$(I),1)="-" THEN 3050
3030 X=X+1:A=VAL(RIGHT$(A$(I),4))
```

```
3040 SUM=SUM+A:SQS=A*A
3050 NEXT I
3060 AVG=SUM/X:VAR=(SQS/X)-AVG*AVG:
     IF VAR>0 THEN SD=SQR(VAR)
3070 AVG=INT(100*AVG+.5)/100:SD=
     INT(100*SD+.5)/100
3080 RETURN
```

Chapter EIGHT

SPRITES AND GRAPHICS

This final chapter describes a few interesting modules that will be useful for graphic presentations. The first part provides a rather unique system for designing, assigning, and controlling sprites. Sprites are those graphic elements which, in the Commodore 64, can be designed and controlled by the operator.

The second part of this chapter gives a few modules that will be useful in developing graphs and other visual representations using Commodore BASIC. These modules can be used as part of other programs to add a graphic facility to the calculation ability of the original program.

SPRITE DEVELOPMENT

Sprites are defined in the Commodore 64 User's Manual as "a high-resolution programmable object that can be made into just about any shape—through BASIC commands." Sprites have many more interesting attributes.

They can be moved around the screen, their color can be changed, collisions between two sprites or between a sprite and copy can be detected, and sprites can pass in front of or behind other sprites. In addition, a sprite can be doubled in size vertically or doubled in size horizontally, or both.

The main problem with sprites is that programs to develop sprites and programs to control sprites are not easy to write. The sprite is made up of a group of dots. Each sprite is 24 dots wide and 21 dots high. Each dot is either "on" or "off," depending on the value of a number associated with the dots. Three eight-digit binary numbers are needed to define each horizontal line in the sprite. As there are 21 horizontal lines, this means that 63 numbers are needed to define one sprite. If we have to calculate each of these numbers, developing a sprite can be a real hassle.

Certainly we have not come this far just to decide that sprites are too much trouble. Help is at hand. We have modules that will help us design these small graphics—and very easily! Let's examine the problems and look at how best to overcome them.

The first problem seems to be that sprites must be defined in terms of binary numbers. Although not a big problem, it is not easily done. The standard method is to sketch the sprite on a 24 × 21 grid by blacking in the spaces that make up the sprite figure and then converting each eight spaces of the grid into a binary number. This conversion is done by assigning a 0 for each open space and a 1 for each blacked-in space. These numbers are then POKEd into the computer to make the sprite.

To make the design of the sprite as easy as possible, we can have the computer draw the grid. We can then specify which squares in the grid are to be filled in and have the

computer calculate the numbers that correspond to the filled-in spaces (see Fig. 8-1, p. 222). In addition to the grid method, perhaps we should also make provision for defining a sprite by inputting binary numbers rather than blacked-in squares. Of course, we should provide for retrieving sprite data stored on a disk (or tape). These all sound like good ideas for our program. Let's do it!

Inputting the Sprite Data

Module 8/1 is the initial module of this program. It DIMensions the sprite array, Z(21,3). It then asks whether we intend to input data through a binary number, the grid, or from a disk. Of course, if desired, a tape can be used in place of the disk. If the grid is chosen, a second choice must be made about whether to use X-Y coordinates, cursor-controlled points, or joystick-controlled points on the grid. Finally, except in the case of input from a disk or tape, we are asked whether we want to SAVE the sprite data.

Because the cursor and joystick method of input often leave information in the keyboard buffer queue, it is necessary to empty the queue before asking for other input. Line 80 (GET JUNK$) takes care of emptying the queue so that anything in the buffer queue has no effect on the question on line 85: "SAVE SPRITE?".

If we choose to input the sprite with binary numbers, the program goes to Module 8/8, which is used as a subroutine. Each line of the sprite is entered as one long (24-digit) binary number. The computer then divides the long number into three eight-digit numbers. At any point on a line, while entering a number, the RETURN can be entered and all the remaining digits for that line will be entered as 0s. This means that if we have entered

0011010111 (10 digits) and then enter a RETURN, the computer will add 14 zeros to complete the 24-digit number. This function can be very handy, especially when we want a whole line of 0s.

After the line is entered, the computer calculates the three binary numbers for the line and displays them as three decimal numbers. We are then asked "READY ?". If yes, the next line is set up for input. If no, the line in question is erased and is set up again for input. When all 21 lines have been entered and approved, the computer returns to Module 8/1.

If instead of choosing the binary input of Module 8/1 we choose to retrieve sprite data from a disk, the computer is sent to the subroutine Module 8/7. Module 8/7 loads a set of sprite data into the Z(21,3) array and returns to Module 8/1. Here it is sent to the middle of Module 8/4 (GOSUB 350), and the data numbers are listed on the screen for inspection.

Finally, if the grid method of input has been chosen, the computer goes to the subroutine Module 8/2. Module 8/2 prints a 24 by 21 square grid on the screen.

If X-Y coordinates have been chosen as the input method, the computer goes to Module 8/3 to accept that input. After the input of each pair of coordinates we are asked, "O.K., DELETE, OR STOP?" "O.K." leaves our last entry and sets up for the next pair of coordinates. If delete is chosen, the last set of coordinates is erased from the grid. If stop is chosen, the sprite design is taken as it exists on the grid and the computer is directed to Module 8/4, where the filled-in grid points are converted to numbers.

If the joystick has been chosen as the input method, the computer goes to Module 8/9, and the input comes under joystick control. The joystick cursor moves around

the grid in response to the joystick motion. If the fire button is pressed, the position of the joystick cursor is replaced by a filled-in square. If the fire button is not depressed, the position of the joystick cursor is not filled in. This last mode allows us to erase squares that have been previously filled in. In order to stop, the joystick cursor is maneuvered into the lower right-hand corner of the grid (row 24, line 21) and the fire button is pressed. The design on the grid is accepted as the completed sprite and the computer returns to convert the grid into numbers.

The cursor input works very much like the joystick input. The up-down and left-right cursor control keys are used to maneuver the grid cursor to the positions desired. The asterisk key (*) is used to replace the grid cursor with a filled-in sphere. Thus the asterisk key works with the cursor control like the fire button works with the joystick control. Placing the cursor in the lower right-hand square of the grid (row 24, line 21) and pressing the asterisk key stops the process, just as it does in the joystick program.

Now we have a program that will accept input as binary numbers, from disk (or tape) storage, or can be input onto a grid using X-Y coordinates, joystick control, or cursor control.

Calculate, Save, and View the Sprite

As soon as we stop the design phase of the program, the computer goes to Module 8/4, which calculates the sprite numbers. The computer scans the grid and each filled square is considered to be a 1 in the appropriate binary number. This is done by PEEKing into each memory position where the grid squares are stored, and if a 102 is found (the CHR$ number for a filled square), its position

in the grid is used to fashion the number. Line 320 is where all this action takes place.

Module 8/4 also lists the numbers that define the sprite. Line 360 lists the 63 numbers in decimal base. They are listed three abreast in 21 rows. This part of the module is also used separately from the calculation part of the module.

Module 8/5 saves the 63 numbers stored in the sprite array. This module is written to save the sprite on a disk, but of course, it can easily be changed to accommodate tape by changing the 8 to a 1 in line 420.

Now that we have calculated the numbers and saved them, it is time to have a look at the little graphic. Module 8/6 first allows us to assign a number and a color to the sprite we have created. The sprite identification number allows us to select and control a single sprite from among the eight that may be on the screen at one time. The number also determines which sprite will pass in front and which will pass behind in a sprite to sprite collision. The lower-numbered sprite will pass in front of the higher-numbered sprite. Once we have assigned a number and color, the module POKEs the numbers into the appropriate memory locations and displays the sprite. The sprite is first displayed double size for a few seconds and then is displayed normal size.

Module 8/11 allows us to move the sprite around the screen with a joystick. It will not be difficult to convert this module to use the keyboard controls rather than a joystick. (To do this, compare the first 11 lines of Module 8/9 to Module 8/10.) Module 8/11 selects its function by using an ON–GOSUB command. Module 8/11A accomplishes exactly the same thing that Module 8/11 does, except that it uses the IF–THEN command, and may be slightly faster in its responses.

Finally, Module 8/7 will load a saved sprite from a disk. Once loaded, the sprite can be listed, viewed, moved, and if desired, can be resaved. Of course, the module can be modified to load from tape.

Here, then, is the sprite design factory. You may want to design a module that will allow you to add multicolor to your sprite. You will find the modules quite useful in other programs. The idea of POKEing characters into a grid and then scanning the grid to define some other function has many intriguing possibilities. Once the sprite is saved, the load, the view, and the move modules can be used in a program to place or manipulate the sprite or sprites to suit your needs. There are certainly many useful possibilities.

GRAPHIC SKETCHPAD

As with most of the Commodore functions, the possibilities for graphics are almost unlimited. Here we will cover some basic graphic functions that may be useful. We will develop a "sketchpad" which uses basic commands. We will use the joystick or cursor module to create graphics and we will add colors.

Introduction

The points of the display screen have two elements, position and color. When using the normal screen (not high resolution), the screen positions are kept in memory in the space beginning at 1024, and the color of each of those positions are kept in memory beginning at 55296. Both position and color must be specified in order for a character to appear on the screen. This information is not neces-

sary in order to use the modules but may help you to understand the modules.

To avoid writing the addresses of the position and color memory every time we refer to them, we can assign each of them to a variable. In line 20 of Initial Module 8/12, we have assigned the color memory starting place (55296) to variable C, and the position memory start (1024) to—what else—variable P.

The next four lines of Module 8/12 (25, 30, 35, and 40) POKE color bars onto the screen in the top two screen lines. The POKE commands use the C and P variables which we assigned in the first line of the module. The 160 that is being POKEd into the position memory is an inverse space. This means that the character is a solid square. The color being POKEd into the color memory is X, which changes from 0 (black), through all the colors, to 15 (gray 3). The combination of character and color produces 16 color bars along the top of the screen. Later, these bars will be used to select colors for the sketchpad cursor.

The next six lines of Module 8/12 (45 through 70) place 12 symbols down the left side of our screen. These symbols are defined by the numbers found in the DATA lines at the end of the module. To change the symbols, simply change the numbers in the DATA lines to correspond to the characters desired. These symbols will be used to define what the cursor will draw on the sketchpad.

The next two lines (75 and 80) define functions that will be used in Module 8/14. As we can see, they each contain the "40*" element, which is associated with the screen memory. These functions could have been defined in the first lines of the module and used in the POKE commands. Notice how these functions could have been used in lines 30, 35, 55, 60, and 65. Finally, lines 90 and 95 are the DATA lines that we referred to earlier.

The screen now has the color bars and symbols printed on it from which we can select the desired sketch "brush." What we need now is a method for drawing on the pad. The drawing cursor is a small diamond. Its screen display code is 90. There are two methods provided for moving the cursor around on the screen. Module 8/13A uses a joystick input, and Module 8/13B uses the cursor controls on the Commodore keyboard.

Joystick Control

Module 8/13A starts by setting X = 5 and Y =1. These define the starting place for the cursor. If we wanted the cursor to start in the middle of the screen, we could set X = 20 and Y = 13. The remainder of line 105 sets RY = 32 (space), CL = 14 (light blue), and RX = 67 (line).

RY is used to store the screen code of the space into which the cursor is moving. When the joystick or cursor control tells the cursor to move to a new space, the screen code for that space is put into RY before the cursor moves in. Later, when the cursor moves out of the space, the value in RY is POKEd back into the vacated space. The variable CY is the color information for the spaces through which the cursor is moving. This saving and reinserting of space values allows the cursor to move over an existing drawing without erasing it.

The variables RX and CX store the symbol and color which has currently been selected for use. We select color by placing the cursor over the desired color bar and pressing the fire button (if using the joystick) or the asterisk (if using cursor control). Doing either of these will place the color code number in CX. The same process is used to select the symbol code and to place it in RX. This selec-

tion of colors and symbols is done in Module 8/15, which will be discussed later.

The joystick value is determined by PEEKing 56321. Line 115 then separates the direction value of the stick from the condition of the fire button. If the cursor is in the lower right-hand corner and the fire button is pushed (X = 39, Y = 24, and BT = 0), the program stops. If the cursor is not in the lower right-hand corner of the screen with the fire button pushed, the program passes to line 140. Line 140 sends the program to Module 8/15, where the color or symbol can be selected.

Finally, line 150 sends the program to the appropriate subroutine of Module 8/14. Which subroutine is used is determined by the value of the joystick position. On return from Module 8/14, the cursor is moved to the new position by line 160, and the program starts this same sequence over at line 110.

Keyboard Cursor Control

Module 8/13B does essentially the same thing as Module 8/13A, except that 8/13B uses the keyboard, rather than a joystick, to control the cursor. These two modules use the same line numbers, since we have assumed that only one of the modules will be used in a program. If both modules are wanted in one program, it is easy to change the line numbers of one of the modules.

There are differences between the two control modules. The joystick module draws only when the fire button is pressed. The cursor control module, however, uses the asterisk key (*) to turn on the drawing facility and uses the AT Key (@) to turn it off. This means that, once turned on, Module 8/13B will draw until turned off. Another difference between these two modules is that Module

8/13B cannot move in a diagonal direction, since the cursor controls are only up-down and left-right. If we want to move diagonally using keyboard controls, we can choose any four keys and assign directions to them.

Because the brackets and the inequality signs are easily identified and are near the cursor controls, we will choose them for our four diagonal control keys. We will actually use colon, semicolon, comma, and period so that we will not have to use the shift key. Here are the four lines that need to be added to the program if we wish to move diagonally:

```
125 IF A$=":"THEN GOSUB 250
126 IF A$=","THEN GOSUB 260
127 IF A$="."THEN GOSUB 290
128 IF A$=";"THEN GOSUB 300
```

The program works quite well without adding lines 125 through 128. To use only the up-down and left-right controls, however, does require more use of the on-off keys.

Directions and Materials

Module 8/14 might be called the core of the program. It is made up of eight subroutines, each of which has four lines. These eight subroutines correspond to the eight directions (up, down, left, right, up-left, up-right, down-left, and down-right) in which we can move around the screen. Each subroutine begins with a line which checks to be sure that the correct value is being used. The next line stores (in RY and CY) the values of the square being moved into, and decides whether or not to draw in the square. If no drawing is required (BT = 16), the subroutine changes the

values of X and Y and POKEs the values saved in the preceding cycle into the square being vacated.

If drawing is required (BT = 0), the subroutine moves the value of CL into the holding variable CX, and some manipulation is performed to allow different symbols to be used if a line is to be drawn (RX = 67). The subroutine then proceeds to change the X-Y values and to POKE the previous square. It then RETURNs to the joystick or cursor control module.

The final module in this program (Module 8/15) is the select module. Here the color and symbol can be selected. The two halves of the module work in similar fashion. First, the module determines whether or not the cursor is in a position that corresponds to a color. If the answer is no, it determines whether or not the cursor is in a position that corresponds to a symbol. If the answer is again no, the computer RETURNs to the main program. If the cursor were in a color position and BT = 0 (the fire button is pushed or the asterisk is on), the drawing color value (CL) is made the same as the color under the cursor. If the cursor is over a symbol and BT = 0, the symbol value (RX) becomes the same as the symbol value under the cursor.

Summary

The ideas in the modules of this sketchpad program have other uses. There are three sets of parameters that work to describe the characters presented on the screen. The values of R and CL are used to define the immediate symbol and color being used. The variables RX and CX are used to store the current values of symbol and color that have been selected. These values can be changed only in Module 8/15.

RY and CY are used as temporary holding variables. They hold the values of the square into which the cursor is about to move. If the cursor is not drawing (BT = 16), then when the cursor moves out of the square, RY and CY restore the square's original values (R = RY and CL = CY on line 110). If the cursor is drawing (BT = 0), R and CL are changed to the drawing values (R = RX and CL = CX on line 213 and other XX3 lines in Module 8/14).

Although there is no direct correlation, the ideas in this program can be used to develop a high-resolution graphics program. Such a program would allow joystick or keyboard control of a cursor using the 320-horizontal-dot by 200-vertical-dot grid. The high-resolution mode is turned on by

```
POKE 53265, PEEK(53265) OR 32
```

and turned off by

```
POKE 53265, PEEK(53265) AND 223
```

Of course, there are many other changes that must be made, the most obvious of which are those needed in the memory locations used for storage of the screen data. This type of programming is beyond the scope of this book, but it is hoped that some helpful directions have been given.

CONCLUSION

In this book we have provided a number of program modules which can have a wide range of uses. In general, we have tried to present the modules in the context of useful programs. This context has often caused the modules to be

less generic than desired. Although all the modules can be used in other programs, most of them will require some modification to make them compatible with the new program's parameters. Do not hesitate to experiment with adjusting the modules to fit your needs. That is what this book is meant to encourage.

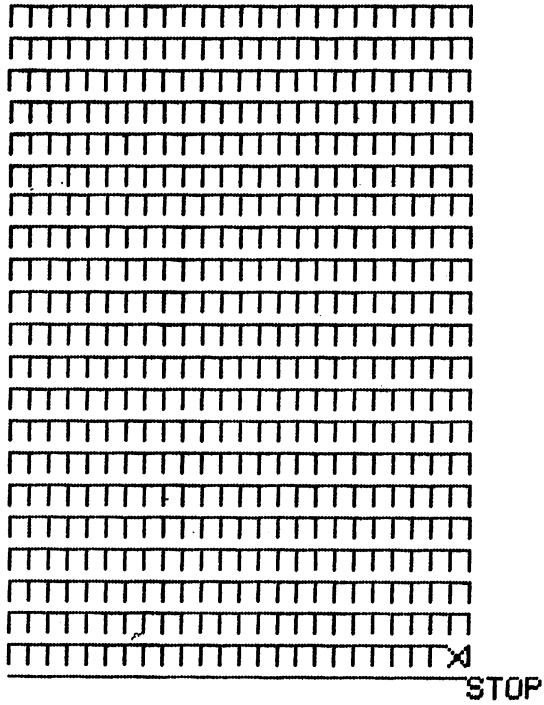

Figure 8-1 Sprite design grid.

CHAPTER EIGHT

MODULES

MODULE 8/1 The initial module for the sprite design program, which DIMensions the sprite array and asks all the questions that determine how the program operates.

```
010 REM ** 8/1-SPRITE INITIAL **
020 DIM Z(21,3):S$="[39x spc]"
030 INPUT"[clear][2x dwn] ENTER
    BINARY,GRID, OR DISK? ([rev]B
    [off],/G/D)";Z$
040 IF Z$="G" THEN INPUT "COORDINATE,
    CURSOR,OR JOYSTICK (C/S/J)";R$
050 IF Z$="G" THEN GOSUB 100:GOSUB 300:
    GOTO 80
060 IF Z$="D" THEN GOSUB 600:GOSUB 350:
    GOTO 500
070 GOSUB 900
080 GET JUNK$:IF JUNK$ <> "" THEN 80
085 Z$="":INPUT"[5x spc] SAVE SPRITE?
    ([rev] Y [off]/N)";Z$:
    IF Z$="N" THEN 500
090 GOTO 400:REM ** GOTO 8/5 **
```

MODULE 8/2 Draws the 24 × 21 square grid and then goes to the module that provides the selected input control.

```
100 REM ** 8/2-SPRITE GRID **
110 SM=1024:CM=55296:PRINT"[clear]"
```

```
120 FOR Y=1 TO 20:PRINT TAB(7);:
    FOR X=1 TO 24
130 PRINT CHR$(207);:NEXT X:
    PRINT CHR$(180):NEXT Y
140 PRINT TAB(7):FOR X=1 TO 23:
    PRINT CHR$(207);:NEXT X:
    PRINT CHR$(118);CHR$(180)
150 PRINT TAB(7);:FOR X=1 TO 24:
    PRINT CHR$(183);:NEXT X:
    PRINT "STOP":PRINT
160 IF R$="J" THEN 1000:REM TO 8/9 **
170 IF R$="S" THEN 1500:REM TO 8/10 **
```

MODULE 8/3 The first input control module (the others are Modules 8/9 and 8/10). This module accepts an X coordinate (column number) and a Y coordinate (row number) and changes in the grid square at that position to CHR$(102).

```
200 REM ** 8/3-X-Y COORDINATES **
210 INPUT"[up][3x rht] ENTER X COORD.
    [6x spc][6x lft]";X
215 INPUT"[up][3x rht] ENTER Y COORD.
    [6x  spc][6x lft]";Y
220 IF (X<1 OR X>24 OR Y<1 OR Y>21)
    THEN 200
230 POKE SM+40*Y+6+X,102:
    POKE CM+40*Y+6+X,13
240 INPUT"[up][3x rht] O.K.,DELETE,
    OR STOP? ([rev] O [off]
    /D/*)";Z$:PRINT"[up]";S$
```

```
250 IF Z$="D" THEN GOSUB 290
260 IF Z$="*" THEN RETURN
270 GOTO 200
290 POKE SM+40*Y+6+X,79:
    POKE CM+40*Y+6+X,14:Z$="":RETURN
```

MODULE 8/4 Reads the grid and calculates the decimal numbers that describe the sprite. The numbers are then listed in sets of three.

```
300 REM ** 8/4-SPRITE NUMBERS **
305 PRINT "[up]" S$ "[up][4x rht]
    [rev] WAIT – CALCULATING SPRITE
    [off]"
310 FOR Y=1 TO 21:FOR N=0 TO 2:
    FOR X=7 TO 0 STEP -1
320 IF PEEK(SM+40*Y+6+N*8+(8-X))=102
    THEN Z(Y,N+1)=Z(Y,N+1)+2↑X
330 NEXT X:NEXT N:NEXT Y
350 REM ** LIST SPRITE **
360 FOR Y=1 TO 21:FOR N=1 TO 3:
    PRINT Z(Y,N) "[spc]" ;:NEXT N:
    PRINT:NEXT Y:RETURN
```

MODULE 8/5 Saves the sprite numbers on disk in a 3 × 7 element array. To save the sprite on tape, change the 8 in line 420 to a 1.

```
400 REM ** 8/5-SAVE SPRITE **
410 INPUT "ENTER SPRITE NAME";NA$
420 OPEN 5,8,5,"0:"+NA$+",S,W"
```

```
430 FOR X=1 TO 21:FOR Y=1 TO 3
440 PRINT#5,Z(X,Y):NEXT Y
450 NEXT X:CLOSE 5:GOTO 500
```

MODULE 8/6 The module to use to see what the sprite actually looks like. It asks for a sprite number and color. These are assigned to the sprite and it is then assembled and displayed on the screen—first double size and then normal size.

```
500 REM ** 8/6-VIEW SPRITE **
510 INPUT "[2x dwn][2x rht] ENTER
    SPRITE NUMBER";S
515 INPUT "[dwn] ENTER SPRITE COLOR
    NUMBER";C
520 PRINT"[clear][2x dwn][2x rht][rev]
    WAIT - POKING SPRITE [off]"
530 V=53248:POKE V+21,0
540 POKE V+21,PEEK(V+21) OR (2↑S):
    FOR X=2040 TO 2047:POKE X,13:NEXT X
550 FOR N=0 TO 62
560 POKE 832+N,Z(INT(N/3+1),N-3*
    INT(N/3)+1):NEXT N:POKE V+39+S,C
570 POKE(V+2↑S),175:POKE(V+1+2↑S),100:
    POKE V+23,4:POKE V+29,4:
    FOR X=1 TO 1000:NEXT X
580 POKE V+2↑S,75:POKE V+23,251:
    POKE V+29,251
590 GOTO 2000
```

MODULE 8/7 Loads sprite numbers from a disk. To load from tape, change the 8 in line 620 to a 1.

```
600 REM ** 8/7-LOAD SPRITE **
610 INPUT "[clear][2x dwn][2x rht]
    ENTER SPRITE NAME";N$
620 OPEN 2,8,2,"0:" +N$+ ",S,R"
630 FOR X=1 TO 21:FOR Y=1 TO 3
640 INPUT#2,Z(X,Y):NEXT Y:NEXT X:
    CLOSE 2:RETURN
```

MODULE 8/8 Accepts a line of twenty-four 0s and 1s as if in a binary number and converts them into three decimal numbers, using eight of the binary digits per decimal number.

```
900 ** 8/8-BINARY INPUT **
910 FOR I=1 to 21:PRINT "[clear][2x dwn]
    [2x rht] *=STOP [2x dwn] ENTER
    BINARY FOR ROW" I
920 PRINT "[2x rht] V [8x rht] 1
    [9x rht] 2 [3x rht] V":INPUT A$:
    IF A$="" THEN A$="000-[16 0's]-000"
930 IF A$="*" THEN STOP
940 IF LEN(A$) < 24 THEN A$=A$+"0":
    GOTO 940
950 IF LEN(A$) > 24 THEN A$=LEFT$(A$,24)
960 FOR X=1 TO 3:B$=LEFT$(A$,8):
    A$=RIGHT$(A$,8*(3-X))
970 FOR Y=0 TO 7:
    Z(I,X)=Z(I,X)+VAL(RIGHT$(B$,1))*2 ↑ Y:
    B$=LEFT$(B$,7-Y):NEXT Y
980 PRINT Z(I,X);"[spc]";:NEXT X:PRINT:
    Z$="":INPUT"[dwn][2x rht] READY ?
    ([rev] Y [off] /N)";Z$
```

```
985 IF Z$="N" THEN I=I-1:FOR X=1 TO 3:
    Z(I+1,X)=0:NEXT X
990 NEXT I:RETURN
```

MODULE 8/9A Starts the cursor at position X = 1, Y = 1. Reads the joystick output and goes to the appropriate subroutine (line 1050). If the fire button is pushed (BT = 0), the module POKEs the character value 102 into the grid space.

```
1000 REM ** 8/9A-JOYSTICK **
1005 X=1:Y=1
1010 JS=PEEK(56321)
1020 BY=JS AND 16
1030 JS=15-(JS AND 15)
1040 IF NOT (X=24 AND Y=21 AND BT=0)
     THEN 1050
1043 GET JUNK$:IF JUNK$ <> "" THEN 1043
1045 POKE CM+40*Y+6+X,0:RETURN
1050 ON JS GOSUB 1100,1150,1010,1200,
     1250,1300,1010,1350.1400,1450
1060 POKE SM+40*Y+6+X,90:
     POKE CM+40*Y+6+X,13
1070 IF BT=0 THEN POKE SM+40*Y+6+X,102:
     POKE CM+40*Y+6+X,13:R=102
1080 GOTO 1010
```

MODULE 8/9B Accomplishes the same thing as Module 8/9A but uses the cursor controls on the Commodore keyboard.

```
1500 REM ** 8/9B-KEYBOARD CURSOR **
1505 X=1:Y=1
1520 BT=16:GET A$:IF A$="" THEN 1520
1521 IF A$="[rht]" THEN GOSUB 1350
1522 IF A$="[lft]" THEN GOSUB 1200
1523 IF A$="[up]" THEN GOSUB 1100
1524 IF A$="[dwn]" THEN GOSUB 1150
1525 IF A$="*" THEN BT=0
1530 POKE SM+40*Y+6+X,90:
     POKE CM+40*Y+6+X,13
1540 IF BT=0 THEN POKE SM+40*Y+6+X,102:
     POKE CM+40*Y+6+X,13:R=102
1545 GET JUNK$:IF JUNK$<>"" THEN 1545
1550 IF X=24 AND Y=21 AND BT=0 THEN
     POKE CM+40*Y+6+X,0:RETURN
1560 GOTO 1520
```

MODULE 8/10 Contains eight subroutines, each corresponding to one of the eight directions of the joystick control.

```
1100 REM ** 8/10-DIRECTION CONTROLS **
1110 IF NOT Y > 1 THEN RETURN
1120 Y=Y-1:POKE SM+40*(Y+1)+6+X,R:
     POKE CM+40*(Y+1)+6+X,14:R=79:RETURN
1150 IF NOT Y < 21 THEN RETURN
1170 Y=Y+1:POKE SM+40*(Y-1)+6+X,R:
     POKE CM+40*(Y-1)+6+X,14:R=79:RETURN
1200 IF NOT X > 1 THEN RETURN
```

```
1220 X=X-1:POKE SM+40*Y+6+(X+1),R:
     POKE CM+40*Y+6+(X+1),14:R=79:RETURN
1250 IF NOT (X >1 AND Y >1) THEN RETURN
1270 X=X-1:Y=Y-1:POKE SM+40*(Y+1)+6+
     (X+1),R:POKE CM+40*(Y+1)+6+
     (X+1),14:R=79
1275 RETURN
1300 IF NOT (X >1 AND Y<21) THEN RETURN
1320 X=X-1:Y=Y+1:POKE SM+40*(Y-1)+6+
     (X+1),R:POKE CM+40*(Y-1)+6+
     (X+1),14:R=79
1325 RETURN
1350 IF NOT X < 24 THEN RETURN
1370 X=X+1:POKE SM+40*Y+6+(X-1),R:
     POKE CM+40*Y+6+(X-1),14:R=79:RETURN
1400 IF NOT (X <24 AND Y>1) THEN RETURN
1420 X=X+1:Y=Y-1:POKE SM+40*(Y+1)+6+
     (X-1),R:POKE CM+40*(Y+1)+6+
     (X-1),R:POKE CM+40*(Y+1)+6+
     (X-1),14:R=79
1425 RETURN
1450 IF NOT (X<24 AND Y<21) THEN RETURN
1470 X=X+1:Y=Y+1:POKE SM+40*(Y-1)+6+
     (X-1),R:POKE CM+40*(Y-1)+6+
     (X-1),14:R=79
1475 RETURN
```

MODULE 8/11A Allows us to move the sprite that we have created. The module is designed to work with a joystick.

```
2000 REM ** 8/11A-MOVE SPRITE **
2005 PRINT "[clear]"
2010 X=0:Y=0
2020 JS=PEEK(56321)
2030 JS=15-(JS AND 15)
2040 ON JS GOSUB 2100,2110,2020,
     2120,2130,2140,2020,2150,2160,2170
2045 IF Y<1 OR X<1 THEN 2020
2050 Z=X:IF Z<256 THEN POKE V+16,0
2060 IFZ>255 THEN POKE V+16,4:Z=Z-255
2070 POKE V+4,Z:POKE V+5,Y:GOTO 2020
2100 Y=Y-1:RETURN
2110 Y=Y+1:RETURN
2120 X=X-1:RETURN
2130 X=X-1:Y=Y-1:RETURN
2140 X=X-1:Y=Y+1:RETURN
2150 X=X+1:RETURN
2160 X=X+1:Y=Y-1:RETURN
2170 X=X+1:Y=Y+1:RETURN
```

MODULE 8/11B Provides another method for moving the sprite. It, too, is designed for the joystick. It operates just a little faster than Module 8/11A.

```
3000 REM ** 8/11B-MOVE SPRITE DIRECT**
3005 PRINT "[clear]"
3010 X=0:Y=0
3020 JS=PEEK(56321)
```

```
3030 JS=15-(JS AND 15)
3041 IF JS=1 THEN Y=Y-1
3042 IF JS=2 THEN Y=Y+1
3044 IF JS=4 THEN X=X-1
3045 IF JS=5 THEN X=X-1:Y=Y-1
3046 IF JS+6 THEN X=X-1:Y=Y+1
3048 IF JS=8 THEN X=X+1
3049 IF JS=9 THEN X=X+1:Y=Y-1
3050 IF JS=10 THEN X=X+1:Y=Y+1
3055 IF Y<1 OR X<1 THEN 3020
3060 Z=X:IF Z<256  THEN POKE V+16,0
3065 IF Z>255 THEN POKE V+16,4:Z=Z-255
3070 POKE V+4,Z:POKE V+5,Y:GOTO 3020
```

MODULE 8/12 The initial module of the graphics sketchpad. It POKEs the color bars onto the screen, reads the DATA lines, POKEs the symbols onto the screen, and DEFines four functions to be used in other modules.

```
10 REM ** 8/12-GRAPHIC INITIAL **
20 C=55296:P=1024
25 PRINT "[clear]":FOR Y=0 TO 1:
   FOR X=0 TO 15
30 POKE P+8+2*X+40*Y,160:
   POKE P+9+2*X+40*Y,160
35 POKE C+8+2*X+40*Y,X:
   POKE C+9+2*X+40*Y,X
40 NEXT X:NEXT Y:PRINT
45 FOR Y=0 TO 24 STEP 2
```

```
50 READ Z
55 POKE P+1+40*Y,Z
60 POKE C+1+40*Y,1
65 POKE P+2+40*Y,89:POKE P+2+40*(Y+1),89
   POKE C+2+40*Y,0:POKE C+2+40*(Y+1),0
70 NEXT Y
75 DEF FN Y(Z)=40*(Y+Z)+X:
   DEF FN X(Z)=40*Y+(X+Z)
80 DEF FN XY(Z)=40*(Y+Z)+(X+Z):
   DEF FN NV(Z)=40*(Y+Z)+(X-Z)
85 POKE P+1+40*Y,Z
90 DATA 67,76,79,80,122,86,91,102
95 DATA 108,123,124,126,160
```

MODULE 8/13A Similar to Module 8/9A. It reads the joystick input and sends the program to the appropriate subroutine.

```
100 REM ** 8/13A-JOYSTICK CONTROL **
105 X=5:Y=1:RY=32:CL=14:RX=67
110 JS=PEEK(56321):BT=16:R=RY:CY=CX
115 BY=JS AND 16:JS=15-(JS AND 15)
120 IF NOT (X=39 AND Y=24 AND BT=0)
    THEN 140
130 STOP
140 GOSUB 400
150 ON JS GOSUB 210,220,230,240,250,
    260,270,280,290,300
160 POKE P+40*Y+X,90:POKE C+40*Y+X,13
180 GOTO 110
```

MODULE 8/13B A module that uses the keyboard cursor controls to move the screen cursor. It can be used in place of Module 8/13A.

```
100 REM ** 8/13B-KEYBOARD CONTROL **
105 X=5:Y=1:RY=32:CL=14:RX=67:BT=16
110 R=RY:CY=CX:GET A$:IF A$=""THEN 110
121 IF A$="[up]" THEN GOSUB 210
122 IF A$="[dwn]" THEN GOSUB 220
123 IF A$="[lft]" THEN GOSUB 240
124 IF A$="[rht]" THEN GOSUB 280
130 IF A$="*" THEN BT=0
135 IF A$="@" THEN BT=16
140 GOSUB 400
150 POKE P+40*Y+X,90:POKE C+40*Y+X,13
160 GET JUNK$:IF JUNK$<>"" THEN 160
170 GOTO 110
```

MODULE 8/14 The eight four-line subroutines that control the screen cursor. This module is used by both the joystick and keyboard controls.

```
200 REM ** 8/14-DIRECTION SUBROUTINES **
210 IF NOT Y>0 THEN RETURN
212 RY=PEEK(P+FNY(-1)):
    CX=PEEK(C+FNY(-1)):IF BT=16 THEN 215
213 CY=CL:R=RX:IF RX=67 THEN R=66
215 Y=Y-1:POKE P+FNY(1),R:
    POKE C+FNY(1),CY:RETURN
220 IF NOT Y<24 THEN RETURN
```

```
222 RY=PEEK(P+FNY(1)):
    CX=PEEK(C+FNY(1)):IF BT=16 THEN 225
223 CY=CL:R=RX:IF RX=67 THEN R=66
225 Y=Y+1:POKE P+FNY(-1),R:
    POKE C+FNY(-1),CY:RETURN
240 IF NOT X>0 THEN RETURN
242 RY=PEEK(P+FNX(-1)):
    CX=PEEK(C+FNX(-)):IF BT=16 THEN 245
243 CY=CL:R=RX:IF RX=67 THEN R=67
245 X=X-1:POKE P+FNX(1),R:
    POKE C+FNX(1),CY:RETURN
250 IF NOT (X>0 AND Y>0) THEN RETURN
252 RY=PEEK(P+FNXY(-1)):
    CX=PEEK(C+FNXY(-1)):IF BT=16 THEN 255
253 CY=CL:R=RX:IF RX=67 THEN R=77
255 Y=Y-1:X=X-1:POKE P+FNXY(1),R:
    POKE C+FNXY(1),CY:RETURN
260 IF NOT (X>0 AND Y<24) THEN RETURN
262 RY=PEEK(P+FNNV(1)):
    CX=PEEK(C+FNNV(1)):IF BT=16 THEN 265
263 CY=CL:R=RX:IF RX=67 THEN R=78
265 Y=Y+1:X=X-1:POKE P+FNNV(-1),R:
    POKE C+FNNV(-1),CY:RETURN
280 IF NOT X<39 THEN RETURN
282 RY=PEEK(P+FNX(1)):
    CX=PEEK(C+FNX(1)):IF BT=16 THEN 285
283 CY=CL:R=RX:IF RX=67 THEN R=67
285 X=X+1:POKE P+FNX(-1),R:
    POKE C+FNX(-1),CY:RETURN
290 IF NOT (X<39 AND Y>0) THEN RETURN
```

```
292 RY=PEEK(P+FNNV(-1)):
    CX=PEEK(C+FNNV(-1)):IF BT=16 THEN 295
293 CY=CL:R=RX:IF RX=67 THEN R=78
295 Y=Y-1:X=X+1:POKE P+FNNV(1),R:
    POKE C+FNNV(1),CY:RETURN
300 IF NOT (X<39 AND Y<24) THEN RETURN
302 RY=PEEK(P+FNXY(1)):
    CX=PEEK(C+FNXY(1)):IF BT=16 THEN 305
303 CY=CL:R=RX:IF RX=67 THEN R=77
305 Y=Y+1:X=X+1:POKE P+FNXY(-1),R:
    POKE C+FNXY(-1),CY:RETURN
```

MODULE 8/15 The final module for the sketchpad, which controls the color and symbol used to draw on the pad.

```
400 REM ** 8/15-SELECT COLORS **
410 IF NOT (( Y=0 OR Y=1) AND X>2 )
    THEN 450
420 IF X>7 AND BT=0 THEN CL=INT
    ((X-8)/2):R=160:BT=16:CY=CL:RETURN
430 RETURN
450 REM ** SELECT SYMBOL **
460 IF NOT (X=0 OR X=1 OR X=2)
    THEN RETURN
470 IF NOT (X=1 AND Y/2=INT(Y/2))
    THEN BT=16:RETURN
480 IF BT=0 THEN RX=RY:BT=16 RETURN
490 RETURN
```

INDEX

A

Accounts:
  divide (modules 5/11, 5/15A), 123, 127
  print (module 5/19), 131
Add:
  new grades (Gradebook) (module 7/10), 198
  new student (Gradebook) (module 7/9), 197
Add on card copy (module 4/13), 84
Address card (write on) (module 4/5B), 74
Amount of payment (module 4/22), 92
Analyzing problems, 8
Array, read by name (module 3/11), 45
Assembling a program from modules, 12
Assign category, budget program (module 5/14), 126
Asterisk adder (module 4/12A), 83
Averages and ranking, 177

B

Backspace and delete (module 4/6), 76
Balance forward:
  new (module 5/9), 121
  trial (module 5/8B), 120
Bank discount (module 4/19), 90
Binary input, sprite (module 8/8), 227
Bottom line (module 5/3B), 113
Bottom line/erase (module 4/7), 77

Bubble sort (module 7/19), 207
Budget program:
  assign category (module 5/14), 126
  convert V to V$ (module 5/15B), 129
  divide accounts (module 5/15A), 127
  initial (module 5/13), 125
  list an array (module 5/16), 129
  list from storage (module 5/18), 131
  manual scroll (module 5/20), 132
  print account (module 5/19), 131
  program, 103
  write data to storage (module 5/17), 130

C

Calculate payroll (module 6/6A), 160
Calculate, save, and view the sprite, 213
Calendar:
  initial (module 4/14), 85
  menu (module 4/16), 87
  reset (module 4/15), 86
Canceled checks, retrieve (module 5/10), 122
Card:
  employee (module 6/4), 157
  initial (module 4/1), 69
  menu (module 4/3A), 71
  title input (module 4/5C), 75
Card file:
  format, 58
  load data (module 4/11), 82
  modules, 57
  other uses, 60
  save (module 4/10), 81
Change employee records (module 6/7A), 164
Checkbook:
  accounts (module 5/11), 123
  bottom line (module 5/3B), 113
  delete line (module 5/5C), 115
  initial (module 5/3A), 112
  input deposits (module 5/5B), 115
  input withdrawals, (module 5/5A), 114
  load data (module 5/4), 113
  new balance forward (module 5/9), 121
  print check register (module 5/6A), 116
  printer (module 5/12), 124
  program, 97
  reconcile to bank (module 5/8A), 118
  retrieve canceled checks (module 5/10), 122
  save canceled checks (module 5/9), 122
  save data (module 5/7), 118
  SUM to SM$ conversion (module 5/6B), 117
  trial balance forward (module 5/8B), 120
Check register print (module 5/6A), 116
CLOSE, 28
Color selection (module 8/15), 236

Common search (module 4/8C), 79
Compound interest (module 4/20), 90
Controls of direction (module 8/10), 229
Convert V to V$ (module 5/15B), 129
Conventions used in this book, 3
Convert SUM to SM$ (module 5/6B), 117
Coordinates, X-Y (module 8/3), 224
Curved grades, Z-Scores (module 7/14), 202

D

Data:
  grouped, 174
  load card (module 4/11), 82
  save (module 5/7), 118
  write to storage (module 5/17), 130
Delete and backspace (module 4/6), 76
Deposits input (module 5/5B), 115
Depreciation, 95
Designing a program using modules, 13
Direction controls (module 8/10), 229
Direction subroutines (module 8/14), 234
Directories using card file modules, 57
Distribution of grades (module 7/13), 201
Divide accounts (module 5/15A), 127
Dollars and cents:
  input 6 digit numeric variable (module 3/16), 49
  input 6 digit string variable (module 3/16A), 49
Double-declining-balance depreciation (module 5/1C), 111

E

Employee:
  card (module 6/4), 157
  list (module 6/3), 156
  records, change (module 6/7A), 164
  status (module 6/7C), 166
Employer costs (module 6/8), 167
Enter hours worked (module 6/5), 159
Erase/bottom line (module 4/7), 77
Error channel, read (module 4/9), 80
Exponential regression (module 7/6B), 195

F

Family records, 61
Flip cards (module 4/1Y), 69
Format:
  small cards (module 4/2A), 70
  single (large) cards (module 4/2B), 71
  checkbook string, 98

Frequency distribution (module 7/1A), 190
Frequency distributions and grouped data, 174
Function symbols, 3
Future value (module 4/24), 92

G

GET#, 28
Gradebook, 180
  add new grades (module 7/10), 198
  add new student (module 7/9), 197
  bubble sort (module 7/19), 207
  grade distribution (module 7/13), 201
  initial module (module 7/7), 196
  list all grades (module 7/11), 199
  list/change student grade (module 7/12), 199
  load file (module 7/8), 197
  mean and standard deviation (module 7/20), 207
  menu (module 7/18), 206
  module list, 186
  print program (module 7/16), 204
  program, 14
  quit program (module 7/17), 205
  start new class (module 7/15), 203
  Z-scores, curved grades (module 7/14), 202
Grade distribution (module 7/13), 201
Graphic:
  direction subroutines (module 8/14), 234
  initial (module 8/12), 232
  joystick control (module 8/13A), 233
  keyboard control (module 8/13B), 234
  select colors (module 8/15), 236
  sketchpad, 215
Grid for sprite (module 8/2), 233
Grocery list menu (module 4/12B), 83
Grocery lists, 60
Grouped data frequency distributions, 174
Grouped frequency distribution (module 7/1B), 191
Grouped median (module 7/2B), 192

H

High resolution mode, 221
Hours worked, enter (module 6/5), 159
Housekeeping, example, 13

I

Income tax:
  1 or 3 (module 6/6B), 162
  2 (module 6/6C), 163
Initial module:
  Budget program (module 5/13), 125

Initial module: (*cont.*):
card file (module 4/1), 69
calendar (module 4/14), 85
checkbook (module 5/3A), 112
gradebook (module 7/7), 196
graphic (module 8/12), 232
payroll (module 6/1), 155
sprite (module 8/1), 223
Input, 34
binary, sprite (module 8/8), 227
card title (module 4/5C), 75
deposits (module 5/5B), 115
name and read a sequential string (module 3/12), 45
one dim numeric array, variable size (module 3/17), 50
one dim string array, variable size (module 3/17A), 50
two dim numeric array, 2 by X (module 3/18), 51
two dim numeric array, 4 by X (module 3/20), 53
two dim string array, 2 by X (module 3/18A), 51
withdrawals (module 5/5A), 114
INPUT#, 28
INPUT# PRINT# and Get$, 36
Inputting the sprite data, 211
Interest:
compound (module 4/20), 90
loans and savings, 64

J

Joystick (module 8/9A), 228
Joystick control, sketchpad, 217
Joystick control module, sketchpad (module 8/13A), 233

K

Keyboard:
control, graphic (module 8/13B), 234
cursor (module 8/9B), 228
cursor control, sketchpad, 218

L

Linear regression (module 7/6A), 194
List:
all grades (module 7/11), 199
an array (two dim, string) in budget program (module 5/16), 129
change student grade (module 7/12), 199
employees (module 6/3), 156
from storage (module 5/18), 131
Load, 26
and run (module 3/03), 39
card data (module 4/11), 82
data (module 5/4), 113
gradebook file (module 7/8), 197
sprite (module 8/7), 226
Loans, savings, and interest, 64
Logarithmic regression (module 7/6C), 195

M

Manual scroll (module 5/20), 132

Maturity value, (module 4/17), 88
Mean, median, mode (module 7/2A), 191
Median for grouped distribution (module 7/2B), 192
Menu, 32
  auto-insertion of items into data line (module 3/15), 47
  calendar (module 4/16), 87
  for card file (module 4/3A), 71
  for message center (module 4/3B), 72
  for payroll (module 6/2), 156
  gradebook (module 7/18), 206
  grocery list (module 4/12B), 83
  with items listed in string variables (module 3/14), 46
Message center menu (module 4/3B), 72
Module list:
  for budget program, 103
  for checkbook program, 99
Monthly payments (module 4/18), 89
Months to pay (module 4/21), 91
Move sprite (module 8/11A), 230
  direct (module 8/11B), 231

N

Name:
  and load (module 3/01), 38
  load and run (module 3/02), 38
  save, verify (module 3/05), 40
New balance forward (module 5/9), 121
Numbers for sprite design (module 8/4), 225
Numeric array:
  (one dim) input variable size (module 3/17), 50
  prints two dim in variable rows and cols (module 3/23), 55
  (two dim) dims = B and C, input (module 3/19A), 52
  (two dim) 4 by X input (module 3/20), 53
  (two dim) input (module 3/18), 51
  (two dim) variable dims input (module 3/19), 52

O

Output:
  20 lines by 4 columns (module 3/22), 55
  string data pages (module 3/21), 54
  variable rows and columns (module 3/23), 55
OPEN, 28

P

Payments:
  amount of (module 4/22), 91

Payments: (*cont.*):
monthly (module 4/18), 89
months to pay (module 4/21), 91
Payroll:
calculate (module 6/6A), 160
changing employee records and saving data, 141
changing to a different period, 152
employee array, 145
employer costs and payroll summary, 137
exit program module, 143
income tax 1 or 3 (module 6/6B), 162
income tax 2 (module 6/6C), 163
initial (module 6/1), 155
initial modules, 136
input hours and calculating the payroll, 144
menu (module 6/2), 156
menu and listing employees, 138
other uses for the modules, 153
print and change payroll variables, 149
print routine (module 6/10), 170
producing a, 135
program modules, 151
summary (module 6/9), 169
variables (module 6/11), 171
Percentile rank (module 7/3A), 192
grouped (module 7/3B), 193
Power regression (module 7/6D), 196
Print:
account (module 5/19), 131
check register (module 5/6A), 116
pages of string data (module 3/21), 54
program, gradebook (module 7/16), 204
routine, payroll (module 6/10), 170
variable rows and columns (module 3/23), 55
PRINT#, 28
Printer routine (module 5/12), 124
Producing a payroll, 135

Q

Quit program, gradebook (module 7/17), 205
Quiz designer program, 187

R

Random files, 31
Read:
array (module 3/11), 45
error channel, (module 4/9), 80
sequential string–input name (module 3/12), 45
sequential string–variable name (module 3/13), 46
sequential using gosub (module 3/10), 44

Reconcile checkbook, 101
Reconcile to bank (module 5/8A), 118
Regression, 178
  exponential (module 7/6B), 195
  linear (module 7/6A), 194
  logarithmic (module 7/6C), 195
  power (module 7/6D), 196
Repeated functions, 3
Retrieve canceled checks (module 5/10), 122
Retrieving sequential data, 30
Run after loading (module 3/03), 39
Run after loading by name (module 3/02), 39

S

Save, 26
  a program, 17
  and verify (module 3/04), 39
  by name and verify (module 3/05), 40
  card file (module 4/10), 81
  data (module 5/7), 118
  sprite (module 8/5), 225
Savings, loans, and interest, 64
Scroll manually (module 5/20), 132
Search for card (module 4/8A), 77
Search routine (module 6/7B), 166
Search security (module 4/8B), 78
Security code, 63
Security for search (module 4/8B), 78
Select colors (module 8/15), 236
Sequential data, 27
  read an array (module 3/11), 45
  read using gosub (module 3/10), 44
  retrieval, 30
  storage, 27
  string read file having a variable name (module 3/13), 46
  string read named file (module 3/12), 45
  write over existing (module 3/08), 42
  write over existing using gosub (module 3/09), 43
  write string (module 3/06), 40
  write using gosub (module 3/07), 41
Single card form (module 4/2B), 71
Sinking fund (module 4/23), 92
Six digit numeric $ and cents (module 3/16), 49
Six digit string $ and cents (module 3/16A), 49
Sketchpad, graphic, 215
Small card form (module 4/2A), 70
Sort, bubble type (module 7/19), 207
Sprite:
  binary input (module 8/8), 227
  calculate, save, and view, 213
  development, 209
  direction controls (module 8/10), 229

Sprite: (*cont.*):
grid (module 8/2), 223
initial (module 8/1), 223
joystick (module 8/9A), 228
keyboard cursor (module 8/9B), 228
move sprite (module 8/11A), 230
move sprite direct (module 8/11B), 231
save (module 8/5), 225
view (module 8/6), 226
X-Y coordinates (module 8/3), 224
Standard deviation and variance (module 7/4), 193
Standard deviation, Z-scores, and regression, 178
Start new class (module 7/15), 203
Statistics, 173
grouped median (module 7/2B), 192
grouped percentile (module 7/3B), 193
mean, median, mode (module 7/2A), 191
percentile rank (module 7/3A), 192
variance and standard deviation (module 7/4), 193
Status of employee (module 6/7C), 166
Storage:
list from (module 5/18), 131
write data to (module 5/17), 130
Storing sequential data, 27
Straight-line depreciation (module 5/1A), 110
String:
array (1 dim) input (module 3/17A), 50
array (2 dim) input (module 3/18A), 51
data (2 dim) print by page and line number (module 3/21), 54
data (2 dim) print 20 lines by 4 columns (module 3/22), 55
Stringing modules together, 12
Subroutines, 21
direction, sketchpad (module 8/14), 234
Sum-of-years depreciation (module 5/1B), 110
SUM to SM$ convert (module 5/6B), 117
Summary, payroll (module 6/9), 169
Symbol conventions, 3

T

Tax:
income 1 or 3 (module 6/6B), 162
income 2 (module 6/6C), 163
Trends, 96
Trends module (module 5/2), 112
Trial balance forward (module 5/8B), 120

U

Using this book, 11

V

Variable element array input (module 3/19), 52
Variables, payroll (module 6/11), 171
Variance and standard deviation (module 7/4), 193
Verify a Save by name (module 3/05), 40
Verify a Save having a variable name (module 3/4), 39
View sprite (module 8/6), 226

W

Withdrawals input (module 5/5A), 114
Write:
  address card (module 4/5B), 74
  data to storage (module 5/17), 130
  on card, (module 4/5A), 73
  over existing sequential (module 3/08), 42
  over existing sequential with gosub (module 3/09), 43
  sequential string alpha/num (module 3/06), 40
  sequential using gosub (module 3/07), 41

X

X-Y coordinates, sprite design (module 8/3), 224

Z

Z-score:
  calculation (module 7/5), 194
  curved grades (module 7/14), 202
  standard deviation and regression, 178

Computers

COMMODORE 64

PROGRAM BUILDING BLOCKS

Ewin and Shirley Gaby

If you are awed by the technical complexity of programming or dissatisfied with pre-packaged, inadequate software, *Commodore 64 Program Building Blocks* offers you the ideal solution. Here in one book is a library of basic modular programs and useful subroutines that can be "linked together" to form your own customized programs.

A basic set of core programs and a group of useful subroutines allow you to design unique, "customized" programs. All the "building blocks" are fully tested and ready to be put to work. Designed to meet Commodore 64 users' specific needs, *Commodore 64 Program Building Blocks* provides practical solutions to a host of application-specific programming problems.

Ewin and Shirley Gaby are the authors of BYTE/McGraw-Hill's *GOSUBS: 100 Program-Building Subroutines in Timex/Sinclair BASIC*. They are the owners of S/E Gaby Associates, Inc., a business consulting firm specializing in communications and training.

ISBN 0-07-022667-9