TURBO PASCAL FOR THE MAC

PROGRAMMING WITH BUSINESS APPLICATIONS

DR. LEON A. WORTMAN
CONTENTS

Acknowledgment vii

Introduction . . . Get Ready! viii

1 . . . Get Set! 1
Introduction to Pascal—Turbo and otherwise.

2 . . . And Go! 5
Warm up with a series of 14 short programs.

3 HISTOGRAM: A Bar Chart Maker for Graphics Presentations 23
Create histograms; bar charts for your data presentations.

4 CONTRIBUTION: Determining the Contribution to Profit 27
How much does a specific product or service contribute to your profit (or loss)?

5 DAYS CREDIT: Measuring Cash in Float 33
Cash flow and accounts receivable.

6 CHECKBOOK: Excellent Code Practice 41
Keep your checking account records up-to-date.

7 DEPRECIATION: Depreciating Your Capital Assets 53
Three methods for depreciation.
COUNTWORDS, COUNTSTROKES & COUNTCHARACTERS: Counting Programs
Tidy counting programs for text files; continued learning about programming.

INVESTVAL: Evaluating Investments
By what percent of the original value has the current market value of your cash investment changed?

LOANPAYMENTS: Mortgage and Loan Tables
Create data tables; calculate, display, and print periodic payments, interest and principal amounts.

NETWORTH: Calculating Your Net Worth
Calculate the difference between what you own and what you owe; string arrays in Turbo Pascal.

AUTOREPORT & TRIPREPORT: Generate Detailed Reports of Trip and Automobile Expenses
Two programs to generate reports of expenses incurred during trips.

SETPRINTER: Send Control Codes to a Printer
A menu set of options for sending configuration codes to the Apple Imagewriter or another printer.

TIMETREND: Use Time-Trend Analysis to Smooth the Rough Spots
Prepare a moving averages table to track business directions.

PLOTTER: A Graph with Equidistant Data Points
Here’s an application for any user who has to plot a series of equidistant data points on a graph.

METRICSENGLISH: Use of the CASE Statement in a Program for English/Metric Conversions
Choose from a menu of 24 metric-to-English and English-to-metric conversions; precision formulas are built in.
<table>
<thead>
<tr>
<th>Number</th>
<th>Program Description</th>
</tr>
</thead>
</table>
| 17     | SORTER: Alphanumeric Sorting . . .  
Garbage in, Neatness out  
Sort text and alphanumerics according to the ASCII collating sequence. |
| 18     | APPRAISAL: A Personnel Appraisal  
Program Provides a Good Session  
in Writing Code  
Appraise the merits of a manager, supervisor, or employee |
| 19     | BREAKEVEN: When Revenues and  
Costs Come Together  
Enter costs and prices; program instantly computes the break-even point and generates a comprehensive data table. |
| 20     | REPCOMMISSION: A Program for  
Sales Managers  
Generate and print a master record of sales revenues and commissions earned, paid out, and still owed to sales reps. |
| 21     | SALESREPORT: Analyzing Sales Performance  
Compute and print sales data, including individual performances and deviations from the averages. |
| 22     | MINICALC: A Four-Function Calculator  
Learn math programming with this four-function calculator, customizable to your specific needs. |
| 23     | PRINTERTEST1 & PRINTERTEST2:  
Programs That Will Test and Benchmark  
Your Printer's Printing Quality  
Two versions of a program that will put your printer through its paces. |
| 24     | COPYIT: Copying and Renaming ASCII Files  
Copy a file to another disk (or to the same disk) using the same or another "legal" file name; make a hard copy with your printer. |
| 25     | PRINTIT1 & PRINTIT2: Utility Programs to  
Make Hard Copy Printouts of Your ASCII Files  
Two programs for listing ASCII files; includes error-handling examples for Turbo Pascal. |
26 SHOWIT: Display a Text File
Interactive program to name and display text files.

27 TYPEIT: Send Your Typing
Directly to the Printer
Use your inexpensive Macintosh like an expensive typewriter; another lesson in programming.

28 EDITIT1, EDITIT2 & EDITIT3:
Three Text Editor Programs
Line-oriented text editors that become progressively more powerful.

29 START, UPDATE & LISTALL:
Essential Elements of a Database Program
Three programs that, together, form a study of the structure of a database program.

30 DATABASE1: Database Program
with Enhanced I/O Error Handling
START, UPDATE & LISTALL are converted to procedures to form an integrated database program.

31 DATABASE2: Enhancements to DATABASE1
A database program with wider applications.

APPENDICES

A Turbo Pascal Expressions and Rules
B Turbo Pascal and Standard Pascal
C Identifiers, Reserved Words, Constants, Variables, and Expressions
D Predefined Procedures and Functions
E I/O Error Handling

Index
Acknowledgment

I am especially grateful to Philippe Kahn and his talented, high-spirited team at Borland International, who continue to amaze us with products that concentrate on speed and economy. Under Kahn’s direction, Borland has broken the pattern that was appearing to become the software industry’s mold: high prices, doubtful quality, incomprehensible documentation, and nonexistent or grudgingly given customer support. I certainly hope Borland will be able to maintain this “attack” on the software “institution.”

Borland’s business methods have proven that if all else fails, one should try integrity. Borland doesn’t even appear to be faced with “failure,” only with continual success.
Introduction

... Get Ready!

In November 1983 a small and unknown company in Scotts Valley, California, started shipping a new compiler for the relatively "old" and well-known Pascal programming language. The company called its product "Turbo Pascal." The first release was for microcomputers that used the CP/M operating system. This was followed shortly by a version for the operating systems used by the IBM PCs. In December 1986, shipments began for Version 1.0 of Turbo Pascal for the Macintosh.

Of course, the "company" is Borland International. Still located in Scotts Valley, it is big by any measure. It is publicly traded, and markets its products worldwide.

Pascal, the programming language, was originally created as a high-level language suited to teaching computer programming in school classrooms. Pascal may never have been intended for use in writing complete applications programs. If that was so, however, many of us weren't aware of it and began to extend the use of Pascal outside the classroom and into applications and utilities of all sorts—simple and complex.

During the past several years, a number of variations of what is generally referred to as the standard or generic Pascal have been created. Each of the variations may follow the general structure of the standard or generic Pascal; however, each invariably incorporates significant modifications in syntax. Perhaps in search of individuality, each adds to or subtracts from its library of predefined procedures and functions, which usually prevents a compiler from completely processing an "alien" construction. The predefinitions will be recognized only by the applicable compiler, which may be equally nonstandard or nongeneric. One can understand, if not appreciate, the confusion, discussions, and conflicts that are caused by personal preferences or dislikes for specific compilers.
With rare exceptions, Pascal compilers had or have common failings (which may have contributed to the Pascal’s virtual disappearance from the programming scene and replacement by other programming languages). They are slow in doing their jobs of converting the Pascal source code to executable code. Their error-trapping methods have been considered inadequate, and error messages may be baffling, ambiguous, or vague. But, slow as it may be, the compiling process is a necessity. To a creative and busy programmer eager to see the compiler process his source code and make it executable, the waiting period is boring, irritating, time-consuming, and frustrating.

Thus, when Borland’s Turbo Pascal compiler Version 1.0 for the IBM PC arrived on the scene on November 15, 1983, many who had been working with other Pascal compilers eagerly risked the relatively small purchase price of $49.95 to sample and test the hard-to-believe claims for speed made for Turbo Pascal.

To say the least, the product was startling! Minutes of compiling time were reduced to seconds. The self-contained editor, error-trapping, and error messages were uniquely helpful. They dramatically conserved the programmer’s time and temper. Locating errors in source code and editing were so wonderfully facilitated by Borland International’s Turbo Pascal, that the very word “Turbo” has become part of the programmer’s and marketer’s jargon, equated with speed, simplicity, and reliability.

Turbo Pascal Version 1.0 was soon updated to Version 2.0. At the same time, Turbo-87 with support for the 8087 math coprocessor was introduced. Users marveled at Turbo Pascal’s interactive editor, powerful library of built-in functions, and ease of operation.

In 1985, Borland began shipments of Turbo Pascal Version 3.0, which compiled twice as fast as Version 2.0. Also in 1985, Borland introduced TurboBCD, with special numeric and string field-formatting features for business applications. Thus, in less than two years after the release of Version 1.0, there were three varieties of Borland’s Turbo Pascal compilers.

At first, the mistrusting ones among us may have thought of Borland’s Turbo Pascal as a tinkerer’s toy, a cheap way to hack at or play around with Pascal. However, Turbo Pascal has earned a rightful place among the important, indispensable tools for the development of serious applications programs.

All the previously mentioned versions were specifically for use with CP/M systems, and with the series of IBM PC and compatible microcomputers. The rest is history—an enviable model for an entrepreneurial success story.

Borland International’s reputation, physical size, and dollar volume have gone well beyond the original expectations of Philippe Kahn, its founder. Borland’s product pricing and value have irrevocably changed marketing practices among software developers.

There’re more chapters to this never-ending success story. In November 1986, Borland introduced its Turbo Pascal software for the Macintosh computer, which gave birth to this book—a companion volume to Turbo Pascal Programming with Business Applications, TAB BOOKS, 1986, which is primarily for users of IBM PC and compatibles. Borland is writing new history by bringing the power and facility of the Turbo Pascal—a combination of a programming language and a compiler—to the Macintosh family of microcomputers.

This book intends to provide some “ready-made” and some not completely ready-made Turbo Pascal code specifically for the Macintosh. You can, at your leisure,
customize and enjoy the code. At the same time, the heavily commented code provides you with a tutorial—a learning vehicle for some of the features that make Turbo Pascal exciting to work with.

This book is not designed to take you from “zero to 60” as a Pascal programmer. No one book seems able to do that; however, it should provide a good starting point for new users, and additional knowledge for users who may already have some experience with Pascal, or even with Turbo Pascal.

You have to personally experience the phenomenon of Borland’s Turbo Pascal: its power and speed are among the indescribable and satisfying things one encounters in working with a high-level programming language.

With this introduction, I’d like to extend a cordial invitation to use and profit from this book to all Macintosh users, especially those who have already discovered the remarkable capabilities of Turbo Pascal and want more source code to work with—and to those who are still using other Pascal compilers (and, therefore, may still be waiting for their compilers to finish the requisite multiple number of passes).

The opening chapters deal with the unique characteristics of the Pascal programming language, and the Turbo Pascal compiler, as implemented by Borland. They get you writing several fundamental programs that show you how to use the Macintosh version of the Turbo Pascal editor and compiler and how to run the programs. These chapters are filled with assorted examples of source code that can be keyed into your Macintosh computer, compiled in seconds, and then executed. Then this book provides you with a liberal number of chapters that are dedicated to specific programs. Each chapter describes its program’s application, lists the commented source code and, where relevant, shows actual printouts generated by runs of the program made with a standard Macintosh Plus and an Imagewriter II. Several appendices at the back of this book provide reference information concerning the Macintosh version of the Turbo Pascal language and syntax.

I make two simple assumptions: you already know how to use the Macintosh, and, you know how to use the Turbo Pascal program’s editor and compiler. With these assumptions, I can save your time by not repeating the operator-instructions in the documents that are supplied with the computer and the Turbo Pascal program.

As far as Borland’s Turbo Pascal goes, you won’t believe your eyes the very first (and second, and third . . .) time you process source code with the Turbo Pascal compiler. Also, you’ll be amazed, if you have some doubts, at the speed at which your Macintosh can operate.

I believe you are about to have a great learning experience and a very good time!
Chapter 1

. . . Get Set!

Here we go, off into the land of the mystical or mythical "Frank Borland," the red-suspendered hybrid mountain man and leprechaun. Is he a continental elf or an American hillbilly? Is he real or imaginary? Few of us care. We just enjoy his harmless sense of humor and mustachioed visage peeping out at us as though he were thinking, "I know a bug that you don’t know!"

It’s reasonable to assume there are "bugs" somewhere in the Turbo Pascal compiler. However, if there are, you won’t be bothered by any of them as you work and play your way through this book of programs and tutorials. The programs in each of the chapters have been compiled with Turbo Pascal for the Macintosh, Version 1.0; then they were repeatedly run and tested with a Macintosh Plus. The tutorials were checked out by language experts, programmers, and business people. There’s no pain left; you can only gain.

The intention of this book is to help you learn to use the Pascal programming language, enhance your pleasure in working with your Macintosh computer, and enjoy the dash and speed of the Turbo Pascal editor and compiler. I make several assumptions about the readers of this book. You have some knowledge and experience, however minor, with entering source code through an editor and in using a compiler to process the code to a form that can be executed.

I assume you may have already had some experience with the so-called American National Standards Institute (ANSI) or standard Pascal or with one of the near-standard Pascal compilers; or with Turbo Pascal. This book is intended entirely for the Turbo Pascal compiler, which has some important differences (as well as similarities) in syntax as far as standard Pascal is concerned. It also has differences in libraries when compared
with other Pascal compilers. I will not spend time and space identifying these similarities or differences. They are not important at this time. The appendices at the end of this book deal with many of the critical dissimilarities.

What is important right now? You should have the documentation for Turbo Pascal for the Macintosh, Version 1.0, the Turbo Pascal disk itself and, of course, a Macintosh computer and a compatible printer. This book's contents, too, are important to you, and you are urged to use it in close conjunction with the documentation supplied with the Turbo Pascal disks.

The Turbo Pascal documentation contains the syntax and rules of procedure. The documentation and the disks are packaged together; so there is no point in wasting your reading time by duplicating that material here. However, there's always room for a book to interpret the documentation and add to your programming fluency and skills, and to your collections of useful utilities, programs, and procedures. This book can do those things for you.

Each subsequent chapter in this book contains the complete source code for a program written in Turbo Pascal. The code is at the end of each chapter and is preceded by a threefold tutorial: (1) what the application of the program is, (2) what the program does or generates after it has been compiled and executed, and (3) notes concerning features of the code that are especially interesting or important to learn so they can more easily be applied to other programs that you may want to write. The notes on the features of the code are also given in the source code itself in the form of comments.

With care and attention to the programs' details, you will be able to extract clauses, statements, expressions and arguments, procedures, and functions that can very well become the foundation for your own library of coding utilities (which are timesaving programmer's tools that, with very little revising, can be used over and over again in program after program).

Although each chapter and its program can stand alone, I am sure you recognize that many of the programs are less comprehensive and powerful than a full-scale commercially released program usually is. Some of the chapter's programs are designed to trap human errors. Others are not as well-protected against human error. Because of this, they offer you an opportunity to practice and to learn how to add to, revise, and enhance source code. You are encouraged to "borrow" any program's functions or procedures and to insert them appropriately into another program to enhance its operation.

The "omissions" you may detect in the source code given in this book are deliberate. Some of the programs may even crash because of a human error, such as entering a letter of the alphabet when a numeral was expected. If you do make a program crash because this kind of error was not trapped, consider it an opportunity to learn how to add an I/O-error trap. Examples are used in some of the programs and in an appendix in this book. They are also described in the Turbo Pascal documentation.

Rest assured that error traps and other such niceties are known to us. Some have been used quite deliberately in several of the programs. Omitting them from a commercial program not intended for user-modification or for learning, may deserve your wrath and your demand for a "bug-free" update. However, the "bugs" (if that's what you choose to call them) you may find in the programs in this book are there for you to eliminate with any one or more of your favorite Turbo Pascal constructions.
The next chapter provides tutorial text and a substantial number of exercises in the form of short programs written in Turbo Pascal. They can be compiled and executed, serving as warm-ups for the more-complex code you will be working with in later chapters.

At any rate, before we get too involved with tutoring, here are a few words of guidance on creating and compiling programs with Turbo Pascal. They will help you get up to speed very rapidly.

Turbo Pascal makes it easy for you to create a program on the Macintosh. Turbo Pascal will set up a window for you and handle it as a plain video console. You can then write to the screen, prompt for and receive input, move the cursor around, scroll the screen's images, and so on. The marvelous thing is that you don't have to know anything at all about the Macintosh to start writing Pascal programs on it. This chapter isn't intended to be a comprehensive tutorial on Turbo Pascal, or on Pascal in general. Its sole purpose is to briefly augment and simplify the instructions provided in the documentation supplied with the Turbo Pascal disks. Let's begin by reviewing how to get a new program typed in and running.

**CREATING A PROGRAM**

To write a program, first get into Turbo Pascal by double-clicking on Turbo, an icon represented by a race car. Turbo will bring up all its menus, including an *untitled* program window.

Type in your program's source code. Use the keyboard, mouse, and the pulldown-menu commands at the top of the window. The Turbo Pascal documentation will give you comprehensive instructions on the features and capabilities of the built-in editor for writing source code.

**COMPILING A PROGRAM**

Pull down the COMPILE menu and select the RUN command, or simultaneously press the `<cmd>` and the R keys. If an error is found by the compiler, Turbo Pascal will put you back into the editor at the location of the error. Correct the error and again select Run.

Once you've corrected all errors, save your program out to the disk, using the Save or Save As commands in the File menu. That's when Turbo asks you for the name of the program, and that's when you choose a name. To avoid confusion, use the standard practice in Pascal programming: give the program's source code the same name as the one you give to it in the first line of your code; add the suffix PAS separated from the name proper by a period.

Your program should run now... unless, of course, you have any run-time errors. If so, the Macintosh's System-Error dialog box will be displayed. In that case, click the mouse on the Resume button and you'll find yourself back in the Turbo Pascal editor with the cursor placed at the location where the error occurred. Correct the error and again select Run.

If you should happen to crash the system, which does not usually occur, double-click on your program's icon to get back into Turbo Pascal, and then edit your program. (That's a good reason for setting the Autosave command, which automatically saves your program's source code to disk before you enter the Run command.)
Once you've corrected all your run-time errors, save your program out to disk again. Now, select the To Disk command from the pull-down Compile menu, or simultaneously press the <cmd> key and the K key. Your program is now an executable file.

It appears as a standard Macintosh application icon of a hand that is writing on a piece of paper. When that's done, exit Turbo Pascal by selecting the Quit command from the pulldown File menu, or by simultaneously pressing the <cmd> and the Q keys. You can run your compiled program anytime by double-clicking on the application icon.
Let's start with a few short tips and several programs that illustrate and explain a few of the programming techniques you will be using throughout this book. For example, Turbo Pascal for the Macintosh has a built-in function to clear the screen. Thus, when you see this line of code:

```
ClearScreen ;
```

you will recognize instantly that it is there to clear the screen and make it ready for the next instruction. (Don't forget the semicolon that comes after ClearScreen with no other code or characters, other than a space, separating the two.) Note that the space is not essential. It's a matter of personal choice. Some programmers always put one space between the end of a statement and the terminating semicolon character; others do not. Both are technically correct. Neither one performs better than the other because the compiler ignores empty spaces. I prefer to include the space before the semicolon in the code listings (as you will see in this book), but this is purely a matter of personal choice.

**COMMENTS AND DELIMITERS**

There are two kinds of *delimiters* used to incorporate comments in the source code: (1) surround the comment with left and right curly brackets; e.g., `{a comment must be placed inside these brackets}`; or, (2) surround the comment with a matching set of left and right parentheses with an asterisk following the opening or left parenthesis and an asterisk directly preceding the right or closing parenthesis; e.g., `(* a comment`
must be placed inside these delimiters *). It is absolutely essential to make certain the delimiters are used as matched pairs; for every left curly bracket or left parenthesis-asterisk combination there must be a right-going set; and vice versa.

The compiler watches for a matched set of delimiters and signals a compile-time error if they are not matched left and right. The compiler will not complete the compilation until the situation is corrected. Either type of delimiter, but not both types, can be used to enclose a single comment. Which should you use? Whichever looks best to you at the time. Both can be used at will within the same program, as long as they are not mismatched. For example: \{ this is a comment *) or (* this is a comment } will be declared as errors and will stop the compiler's action.

YOUR FIRST PROGRAM

Now's the time to start writing some source code. We begin with simple and minimal code to illustrate the bare-essential ingredients of a compilable and executable program, as simply as the following lines in Fig. 2-1.

To aid this explanation (I emphasize only to aid this explanation) I've numbered the lines below as one through five. Line one starts with the mandatory word PROGRAM. You'll see it in every program in this and other books that list Pascal source code. The name of the program and a semicolon are next.

We leave the second line blank purely for visual aesthetics and to give prominence to lines one and three. In this example, line three must start with the key word BEGIN. Line four contains the Pascal word WriteLn, which is an instruction that the text contained within the set of adjoining parentheses and surrounded by single quotation marks or apostrophes is to be written to a device. Because there is no name given for a specific device, the compiler accepts the default device, which is the console or video display. The fifth line consists of the solitary word END followed, appropriately, by a period to terminate the program.

When the MYFIRST program is compiled and executed, it displays this text at the flush left column of the Macintosh's screen:

Hello world, universe, and whatever is out there!

If you've taken a break to enter and compile the program you noticed two important things. One, the video screen automatically cleared itself before the program was executed. This is a characteristic of Turbo Pascal, Version 1.0. You don’t have to enter the word ClearScreen at the start of a program; only when you deliberately choose to clear the screen of its display. Also, your message didn't really stay on the screen

![Fig. 2-1. The first program displays a one-line statement.](image)
PROGRAM MyFirstPlus;
BEGIN
  WriteLn('Hello world, universe, and whatever is out there!');
  ReadLn;
END.

Fig. 2-2. This program adds an important line of code to the first.

very long; in fact, it raced away. If we want to keep the image on display until we are ready to deliberately send it into the domain of vanished screen images, we must enter another line of code following the WriteLn expression; then we have to use the ReadLn expression, as demonstrated in the next program.

ENHANCING YOUR FIRST PROGRAM

When a program encounters the WriteLn statement, it sends out information to the user via the Macintosh's video screen. Conversely, when a program meets ReadLn it waits to receive information from the user, via the keyboard. Those points are illustrated in a variation of MYFIRST program, MYFIRSTPLUS, as seen in Fig. 2-2.

The essential difference between MYFIRST and MYFIRSTPLUS is that the string of text following the WriteLn expression and enclosed by apostrophes does not show itself for a short moment and then vanish. It continues to be displayed on the Macintosh video screen until the Return key is pressed. When the Return key is pressed, the screen clears and the Macintosh is returned to either the desktop or the Turbo Pascal compiler, depending on where you made the call to MYFIRSTPLUS.

YOUR SECOND PROGRAM

The next program, MYSECOND, uses more of Turbo Pascal's capabilities. The source code is given in Fig. 2-3. Examine it while we explain the added features.

VAR, an identifier, appears below the program's name. It declares the variables that are to be used in the code for MYSECOND. In this case, we have only one variable that will store information. We give the name of the variable, MyName, which will store information, followed by a colon, and then a syntactical definition of the type of variable. We want to store text, letters of the alphabet; so, we identify it as a string. The left and right square brackets enclose the maximum number of characters that the variable string is expected to store. We are going to store a name; 25 characters should be adequate (although we could store as many as 128 characters).

The program has its required BEGIN. The expression GotoXY instructs the cursor to start the next line or send the cursor to a position declared by the X and Y parameters, which are enclosed by the parentheses that follow GotoXY. The X parameter refers to the horizontal location; if X is assigned the value of one, as we have done, the cursor is instructed to move to column one. The Y parameter is the vertical position to which the cursor is to be sent; if Y is assigned the value of five, as we have done, the cursor is commanded to row five of the video screen. In this way the need to insert an empty
PROGRAM MySecond;

VAR
  MyName : String[25];

BEGIN
  GotoXY(1,5);
  WriteLn('This is my second program.');

  GotoXY(1,7);
  WriteLn('It does more than greetings.');

  GotoXY(1,9);
  Write('For example, tell me, what’s your name: '); ReadLn(MyName);

  GotoXY(1,11);
  WriteLn('Your name is ',MyName,'; (or so you say!)');

  GotoXY(1,13);
  Write('How did I get it right?'); Write(' Press Return and I’ll tell you how: '); ReadLn;

  ClearScreen;

  GotoXY(18,10);
  WriteLn('Simple...I’m a smart computer!');

  GotoXY(10,14);
  WriteLn;
  Write('Now, if you will press the Return key, we can ’); Write('both go back to the desktop... ’); ReadLn;
END.

Fig. 2-3. The second program demonstrates interaction with the user.

WriteLn instruction for each line you want to skip on the screen is omitted, and you can also declare the cursor’s horizontal position.

When you run the MYSECOND program, it introduces itself and then asks for your name. The first ReadLn is followed by the variable MyName enclosed in parentheses. The program waits for you to type a name or other text (no longer than 25 characters including spaces), and then press the Return key. The string you typed is stored in the variable MyName. Line 11, column one, displays exactly what you typed by the WriteLn call to MyName. Note the punctuation used to insert the values stored in
MyName in the middle of a sentence. The remainder of the program's operation has been covered previously and should be quite clear to you.

YOUR THIRD PROGRAM

The program MYTHIRD is a variation of MYSECOND. It demonstrates the use of PROCEDURES. First, look at the code shown in Fig. 2-4. Pay special attention to the very last use of BEGIN.

```
PROGRAM MyThird;

VAR
  MyName: String[25];

PROCEDURE SignOn;

  BEGIN
    GotoXY(1,5);
    WriteLn('This is my third program.');
  END;  (of SignOn)

PROCEDURE Explain;

  BEGIN
    GotoXY(1,7);
    WriteLn('It demonstrates PROCEDURES.');
  END;  (of Explain)

PROCEDURE Query1;

  BEGIN
    GotoXY(1,9);
    Write('For example, tell me, what''s your name: ');
    ReadLn(MyName);
  END;  (of Query1)

PROCEDURE Reply1;

  BEGIN
    GotoXY(1,11);
    WriteLn('Your name is ',MyName,'; (or so you say!)');
  END;  (of Reply1)
```

Fig. 2-4. A variation of Fig. 2-3, this program elaborately demonstrates the use of procedures.
PROCEDURE Query2;

BEGIN
  GotoXY(1,13);
  Write('How did I get it right?') ;
  Write(' Press Return and I’ll tell you how: ') ;
  ReadLn ;
  ClearScreen ;
END ; { of Query2 }

PROCEDURE Reply2;

BEGIN
  GotoXY(18,10);
  WriteLn('Simple...I’m a smart computer!') ;
  GotoXY(10,14);
  WriteLn;
  Write('Now, if you will press the Return key, we can ’) ;
  Write('both go back to the desktop... ’) ;
  ReadLn ;
END ; { of Reply2 }

BEGIN { the main part of MyThird }
  Signon;
  Explain;
  Query1;
  Reply1;
  Query2;
  Reply2;
END. { of MyThird }

Fig. 2-4. Continued.

A fast examination shows that each line in the section of code between the very last BEGIN and END, the main part of MyThird, contains a group of labels that are exactly the same as the labels following each of the preceding uses of the word PROCEDURE. The sequence of the program’s action is to go to the main part and start calling the procedures in the order in which they are given in the main part of the program. Thus, we can divide the MYSECOND program into PROCEDURES, as in MYTHIRD, and have the program call them in precisely the desired order.

There is no noticeable difference between the operation of the two programs. Actually, in a program of such simplicity there is no need to use the PROCEDURE construction. It is extremely helpful in a longer or more complex program. By “breaking up” the program into procedures, you are able to edit, modify, or debug a program’s source code much more expeditiously than if the program were written, so to speak, as one piece, a single unit.
A SHORT DEMONSTRATION OF SIMPLE GRAPHICS

The next bit of program, shown in Fig. 2-5, demonstrates the simplest form of computer graphics (if we can take the liberty of calling it "graphics"). Seriously, it is perfectly valid to use this method for creating a special effect, say a few lines of decorative borders, using printable characters of the printer's font. In this example, the program LAW displays the initials LAW.

Of special interest is the use of the REPEAT . . . UNTIL clause. Everything that is in code between those two keywords will be repeated until the UNTIL condition is met. That "condition" requires that letter "Q" must be pressed in response to the instruction to press the Q key to quit. Read(Ch) waits for a "Q." The construction UNTIL (Ch) IN ['Q', 'q'] means that the letter will be accepted in either uppercase or lowercase.

```pascal
PROGRAM LAW;  { LAW will become the name of the compiled program. }
VAR
  Ch : Char;
BEGIN
  REPEAT
    WriteLn;
    WriteLn;
    WriteLn(' * * * * * ');
    WriteLn(' * * * * * ');
    WriteLn(' * * * * * ');
    WriteLn(' * * * * * ');
    WriteLn(' * * * * * ');
    WriteLn;
    WriteLn;
    WriteLn;
    Write('Press the ' 'Q' ' key to Quit: '); 
    Read(Ch);  { Press a letter other than "Q" or "q" and see what happens. }
    UNTIL (Ch) IN ['Q', 'q'] ;
END.
```

Fig. 2-5. This program shows the simplest way to create a graphics display.
PROGRAM MyName;

CONST
   TotalTimes = 18;  { The total number of times to display Name. }

VAR
   Name : String[25];   { Max number of characters for Name. }
   NumberOfTimes : Integer;

BEGIN
   GotoXY(1,3);
   Write('What is your name, please?');
   ReadLn(Name);
   ClearScreen;
   GotoXY(1,3);
   WriteLn('Here is ',Name,' displayed ',NumberOfTimes,' times:');
   GotoXY(1,5);
      FOR NumberOfTimes := 1 TO TotalTimes DO
         BEGIN
            WriteLn('You typed your name as: ',Name);
         END;
   GotoXY(1,24);
   Write('Press Return to go back to the desktop...');
   ReadLn;
END.

Fig. 2-6. MyName combines several features learned and adds a FOR...DO loop.

MOVING RIGHT ALONG...

MYNAME is the name of our next program, Fig. 2-6, that illustrates several new features. CONST is a constant value, in this case declaring that TotalTimes has a value of 18. The program asks the user to enter a name, displays the statement that the "name" is to be displayed a number of times (NumberOfTimes, an integer value, obviously). A FOR...DO loop then displays the sentence "You typed your name as: ..." the number of times (18) assigned to TotalTimes when its CONST value was declared at the front end of the program's code.

VARIABLES

The program VALUE, Fig. 2-7, demonstrates that the type of information that can be stored in a variable can take a form other than text. In this case, the variable HowMuch has been declared to accept integer numbers only; letters are not accepted by the program.

ALPHANUMERIC VARIABLES

WRITESTRING, Fig. 2-8, demonstrates the declaration of a VARIABLE as a string of text of alphanumeric characters.
(* Value demonstrates the storage of a value. *)

PROGRAM Value;

VAR
  Value : Integer;

BEGIN
  Write('Enter an integer value: ');
  ReadLn(HowMuch);
  WriteLn;
  WriteLn('The integer value you typed is: ',HowMuch);
  WriteLn;
  Write('Press the Return key to return to the desktop... ');
  ReadLn;
END.

Fig. 2-7. An example of storing and recalling an integer value.

DEALING WITH ARITHMETIC

SIMPLEINTSUM, Fig. 2-9, performs simple mathematics, the addition of two integers, and gives their sum as an integer.

SIMPLEREALSUM, Fig. 2-10, is a variation of a mathematical computation program that sums real numbers and formats them to four decimal places. The

(* WriteString demonstrates the storage and retrieval of a string of text whose maximum length is 30 characters.*)

PROGRAM WriteString;

VAR
  StringText : String[30]; { The number 30 declares the maximum number of characters that StringText can store. }

BEGIN
  Write('Type up to 30 characters, and press Return: ');
  ReadLn(StringText);
  WriteLn;
  WriteLn('The characters are: ',StringText);
  WriteLn;
  Write('Press the Return key to go back to the desktop... ');
  ReadLn;
END.

Fig. 2-8. An example of storing and recalling a string of characters.
{ SimpleIntSum adds (or sums) integers until a Zero is entered. }

PROGRAM SimpleIntSum;

VAR
    Number : Integer;
    Sum    : Integer;

BEGIN { SimpleIntSum }
    Sum := 0; { Initialize "Sum" at a value of Zero. }
    Write('Enter numbers to be summed together. ');
    WriteLn('Press Return after each entry.');
    WriteLn;
    WriteLn('To see the total, type a '0' (Zero) and press Return. ');
    WriteLn;
    WriteLn('Start now: ');
    WriteLn;
    Repeat
        ReadLn(Number);
        Sum := Sum + Number;
    Until (Number = 0);
    WriteLn;
    WriteLn('THE TOTAL IS: ', Sum);
    ReadLn;
END. { of SimpleIntSum }

Fig. 2-9. Adding integer values and displaying their sum.

formatting is shown for values A, B, and Total in the WriteLn argument that displays all three.

AN APPLICATION OF NUMBER HANDLING

COINCHANGER demonstrates a use of division and, at the same time, provides a bit of amusing diversion from our studies. The program asks you to enter an amount of pennies. When you press the Return key, the program instantly calculates the number of quarters, dimes, nickels, and pennies needed to reach the original amount.

As with so many Turbo Pascal programs, the amount of code needed to perform the task is surprisingly small. This is shown in Fig. 2-11, the listing of the code for COINCHANGER.

DIRECTING INFORMATION TO DEVICES

At this point, it is worth repeating that every BEGIN must have a matching END. And a REPEAT must be matched with UNTIL. Also, the word DO must follow a FOR statement when the argument contains more than one line of compilable code. DO is
{ SimpleRealSum adds two real numbers, as well as integers. }

PROGRAM SimpleRealSum;

VAR
  A, B, Total : Real;  // Note, VARIABLES of the same type
                     // can be separated by commas and
                     // then entered on the same line.
BEGIN  // SimpleRealSum
  GotoXY(1,5);
  WriteLn('ON THE SAME LINE, ENTER TWO NUMBERS TO BE ADDED TOGETHER...');
  WriteLn;
  Read(A, B);
  Total := A + B;
  WriteLn;
  WriteLn('The sum of ', A:0:4, ' and ', B:0:4, ' is: ', Total:0:4);
  WriteLn;
  Write('Press Return to go back to the desktop...');
  ReadLn(Ch);
END.  // of SimpleRealSum

Fig. 2-10. Adding real numbers and displaying their sum.

followed by a BEGIN statement and, of course, the argument must conclude with an
END. For easy reading and checking with the unaided eye, the column indentations
are formatted to line up a BEGIN with its matching END; similarly for REPEAT and
UNTIL. The number of columns or the size used for indentations is not meaningful
to the compiler. Use an indent that you find most comfortable and that makes the code
most readable to your unaided eye. My preference is for either a two-column or a
four-column indent; I use two-columns more often.

In Turbo Pascal, the default path for the output of a Write or WriteLn argument
is to the console, which is the screen display of the Macintosh video monitor. The
following line of code within a program steers the output of the statement to the screen:

  WriteLn('This would be seen on the monitor.');

That line is the same as:

  WriteLn(Output, 'This would be seen on the monitor.');

Generally, you can use uppercase or lowercase, or a mixture, for output to the
console. Choose whichever you prefer, but try to be consistent in your calls to a device
such as printer. I have no personal preference.
Turbo Pascal makes it very easy for you to steer the same statement to the printer instead of the console, as in this example:

```pascal
WriteLn(Printer, 'This would be sent to the printer.');
```

You can use both statements to send almost simultaneously to the screen and to the printer:

```pascal
WriteLn('This would be seen on the monitor.'); WriteLn(Printer, 'This would be sent to the printer.');
```

```
PROGRAM CoinChanger;

VAR
  ch       : char;
  amount   : Integer;
  dimes    : Integer;
  nickels  : Integer;
  quarters : Integer;

BEGIN
  WriteLn;
  Write('Enter the amount--in pennies--for which ');
  Write('you want to make change: ');
  Read(amount);

  { Calculations of change are made and stored here, }
  quarters := amount div 25;
  amount := amount mod 25;
  dimes := amount div 10;
  amount := amount mod 10;
  nickels := amount div 5;
  amount := amount mod 5;

  { The results are displayed here. }
  WriteLn;
  Write('Your change is: ');
  WriteLn(quarters, ' Quarters, ', dimes, ' Dimes, ', nickels, ' Nickels, ', amount, ' Pennies.');

  WriteLn;
  Write('Press the Return key to go back to the desktop... ');
  ReadLn(ch);
END.
```

Fig. 2-11. COINCHANGER combines calculations and entertainment.
PRINTOUTS AND THE USES CLAUSE

You must carefully note that whenever a program’s code makes a call to the printer, the following line, known as a call to a unit, must be part of the code:

```
uses PasPrinter;
```

It necessarily follows the opening instruction, which invariably is the word PROGRAM, followed by the name of the program, followed by a semicolon. If you should omit that call to the PasPrinter unit, the compiler will not recognize the word “Printer” and will flag it as an error.

So, generally speaking, use no argument following the Write or WriteLn expression to send to the display (the Console), and do use Printer to send to the printer. There are conditions where the keyword REWRITE must be used. I’ll deal with those events in other programs and chapters later in this book. Note that the key words INPUT and OUTPUT do not have to be declared. They must be included in standard Pascal. But, when they are omitted in Turbo Pascal, the compiler assumes they exist, unless the code has been written so as to deliberately cause them to be omitted.

With the program PRINTDEMO we demonstrate a method for steering the output of WriteLn arguments to either device, the video screen (Console) of the Macintosh or to the printer (Printer). Note, the use of the IF ... THEN ... ELSE clause, which enables the user to choose between the video display and the printer. You will notice that we have used an underscore character inside the name of the variable `Print_I`. This was done to enhance the visual readability and intelligibility of the word.

Be aware that Turbo Pascal and ANSI Pascal require that names of programs, arrays, procedures and functions be a single word. By separating the words Print and It with an underscore character, we meet this convention; Print_It is, technically, a single word. We sometimes use this method and at other times choose not to use it in the source code of the programs in this book.

A demonstration of direction to the printer device is given in the PRINTDEMO program, Fig. 2-12.

IMPROVING THE IMAGEWRITER PRINTOUT

The ImageWriter defaults to draft or standard print quality for text material. If you find that the visual quality of the printouts could stand some improvement, we suggest you include the following commands in your source code, positioning them at the point in the program where the printer is addressed. The program, when it is run, will then automatically send an appropriate “software” signal to the ImageWriter.

1. To command the ImageWriter to print in its near letter quality mode:

   ```
   Write(Printer,Char(27),Char(97),Char(50)) ;
   ```

2. To command the ImageWriter to print zeros with slashes running through each zero:

   ```
   Write(Printer,Char(27),Char(68),Char(0),Char(1)) ;
   ```

Using the same method, other printing-control codes can be transmitted to the printer. Of course, you are aware that the quality of the printout, good or bad, is not inherent
PrintDemo demonstrates one of the ways to direct information to the Printer or to the Console.

PROGRAM PrintDemo;
uses PasPrinter; { Call the library unit for the Printer device. }

VAR
  Ch  : Char; { User's responses are stored here. }
  Print_It : Text; { This is the printer. }

BEGIN { PrintDemo }
REPEAT
  Write('Do you want a D)isplay or a P)rintout? (D or P): '); 
  Read(Ch);
  UNTIL (Ch) IN ['D','d','P','p'] ;

IF (Ch) IN ['D','d'] THEN
BEGIN  { If 'D' (Display) was selected, send to the console. }
  WriteLn;
  WriteLn('This is the first line at the screen (Console).');
  WriteLn('This is the second line at the screen (Console).');
  WriteLn('This is the third and last line at the screen (Console).');
  WriteLn;
  Write('*** Press Return to go back to the desktop ***');
  Read(Ch);
END

ELSE  { If 'P' (Printer) was selected, send to the printer. }
BEGIN
  { To redirect the output to the printer, the path to the }
  { Printer must be opened with the directive 'Printer'. The }
  { statement "uses PasPrinter ;" must be included in the }
  { source code immediately after the first line, which }
  { contains the name of the program. To send to the Printer, }
  { specify 'Printer' rather than the default device, which is }
  { the monitor's screen or 'Console'. This is demonstrated }
  { in next WriteLn statements. }
  WriteLn(Printer);  { 'Printer' sends output to the printer. }
  WriteLn(Printer,'This is the first line at the printer.'); 
  WriteLn(Printer,'This is the second line at the printer.');
  WriteLn(Printer,'This is the third and last line at the printer.');
END;  { Note: for every BEGIN there must be a matching END. }
END. { PrintDemo }

Fig. 2-12. Directing the output of a program to the video display or to the printer.
in the program. However, as described above, the program’s code can force the ImageWriter to print in a mode other than its default setup dictates.

**TRAPPING HUMAN ERRORS**

It is not unusual for the person at the keyboard to inadvertently enter an incorrect response to a question displayed on the video screen. If the program is not designed to intercept such typos or bloopers, the program can bomb, requiring a restart and, possibly, the loss of some time to re-enter data or information that was being stored up to the point in the program where the error occurred.

The ERRORTRAP program affords an example of one way to trap a workaday human error, entering a number that is out of range. See the source code listing for ERRORTRAP in Fig. 2-13.

```pascal
{ ErrorTrap demonstrates IF..THEN..ELSE as }
{ used in trapping an out-of-range error. }

PROGRAM ErrorTrap ;

VAR
  Ch  : Char ;  { Identify Ch as a variable of type Char. }
  Number : Real ;  { The program can handle real numbers. }

BEGIN  { ErrorTrap }
  { The user is asked to pick a number in a given range. }
  Write('Select a number in the range of 1 to 12: ') ;
  { The user enters a number. }
  ReadLn(Number) ;

  { The program tests the user’s entry. }
  { Is it less than 1 or greater than 12? }
  IF (Number < 1) OR (Number > 12) THEN
    { If the user enters a number less than 1 or }
    { greater than 12, the trap is sprung and a }
    { message is displayed in the first WriteLn. }

  { The user is asked to pick a number in a given range. }
  WriteLn ;
  { Number:4:2 formats the display of the }
  { real-number entry to two places. }
  WriteLn('Your selection of ',Number:4:2,
    ' is out of the range of 1 to 12.') ;
  WriteLn('Sorry about that!') ;
  WriteLn ;  { Skip a line. }
  Write('You must start again...') ;
  Write('Press Return to go back to the desktop... ') ;
  ReadLn ;
END
```

Fig. 2-13. Use of an error-trapping program to learn about IF . . . THEN . . . ELSE loops.
(On the other hand, if it is in the range of 1 to 12, the program jumps past the first statement to the 'else'.)

```
ELSE
BEGIN
WriteLn;
WriteLn('Your selection of ',Number:4:2,
     ' is in the range of 1 to 12.');
WriteLn;
Write('You can continue with the program...');
Write('Press Return to continue...');
ReadLn;
END;

(Additional PROCEDURES that continue)
(the program could be placed here.)

END. (of ErrorTrap)
```

Fig. 2-13. Continued.

**A PROGRAM THAT DEMANDS A PASSWORD**

Human errors made at the keyboard can be caught and signaled via a screen display through the use of various error traps.

However, here is an illustration of error trapping in conjunction with a password program that can prevent unauthorized persons without absolute knowledge of the very specific password embedded in the program's source code from accessing the private data or security information.

The source code is given in Fig. 2-14 for PASSWORD. It has been abundantly commented to guide you through its actions.

```
PROGRAM Password;

CONST
  StrLength = 20; (StrLength limits the password to 20 characters.)

VAR
  Decoded_Input : String[StrLength]; (Decoded user input.)
  Password     : String[StrLength]; (Put the real password here.)
  User_Says    : String[StrLength]; (Put the user's response here.)
  I,TOP        : Integer;    (Whole numbers are required.)
  Right_On     : Boolean;    (TRUE or FALSE conditions.)
```

Fig. 2-14. A password program adds to our knowledge of writing code for special applications.
BEGIN { the main part of PassWord }
Password := 'PRIVATE';
{ PRIVATE is the password. }
{ If you want another or want to }
{ change the password, replace }
{ PRIVATE with the new password. }

{ To make the password work, you }
{ MUST type the letters in REVERSE }
{ order and in the same case, upper }
{ and lower, as it has been entered. }
{ You can change the password to }
{ another desired one by revising }
{ or changing the word 'private' to }
{ one that you prefer. }
GotoXY(1,5);
Write('----> Enter the password: ');

{ Get the user's entry for the password. }
ReadLn(User_Says);

{ Set Top to equal how long the string is, and }
{ set the length of Decoded_Input equal to Top. }
Top := Length(User_Says);
Decoded_Input[0] := Chr(Top);

{ The program reverses the typed order of the }
{ characters of User_Says by moving each of the }
{ characters to its corresponding position at }
{ the end of the string Decoded_Input. }
FOR I := 1 TO Top DO
  Decoded_Input[Top - I + 1] := User_Says[I];

{ Right_On is TRUE if Decoded_Input and Password }
{ are the same; otherwise Right_On is FALSE. }
Right_On := Decoded_Input = Password;

{ If the user doesn't enter the correct password, }
{ display a message and continue. }
IF ( NOT Right_On) THEN
  BEGIN
  WriteLn; { Skip a line. }
  WriteLn('The password entered is not correct.' );
  WriteLn; { Skip a line. }
  Write('Press the Return key to go back to the desktop... ');
  ReadLn;
  EXIT;
  END;
( If the password is entered correctly, )
( this confirming message is displayed. )
WriteLn;
WriteLn('The password is correct!');
WriteLn;
WriteLn('Therefore, you are now inside the program...');
WriteLn;
WriteLn('(The rest of the program would follow from this point.)');
WriteLn;
Write('Press the Return key to go back to the desktop...');
ReadLn;
END. { of PassWord }

Fig. 2-14. Continued.

Now, proceed to the next chapters, which will take you more deeply into the realm of applications programs.
Chapter 3

HISTOGRAM: A Bar Chart Maker for Graphics Presentations

Histograms, data represented as bars of relative length, are among the most common of all graphics used in the sciences and in business environments. Our HISTOGRAM program will take numeric data points that are keyed in and convert them to strings of symbolic characters. When these characters are printed end to end, according to the size of the numeric entry, they appear to be a bar. The larger the data entry, the longer the bar.

The restraint of an 80-column video screen and printers sets a pragmatic limit on the number of characters that can be displayed or printed end to end. Normally, the limit would be 80-character bars: however, 80 is an inconvenient number to work with. Therefore, I’ve put an algorithm in the code that enables you to display histogram bars representing data values as large as 100 on a single screen by the simple expedient of dividing the values of data entries by two. Therefore, one character is printed for every two units of value. In other words, if you enter a data value of 100, 50 characters will be displayed and printed; a data value of 50 generates 25 end-to-end characters; and so on.

The bars are printed and displayed horizontally. The value you enter for a bar is displayed and printed at the left of that bar with two decimal places to the right of the integer for each bar; however, since the video monitor and the printer cannot display or print fractions of characters, the data value is rounded to two decimal places before being processed as a bar.

Histograms display data in a step-function order. This is especially valuable when you want to gain an overall, visual comparison of relative values. Bar charts accomplish this by placing bars of varying length or height side-by-side against a reference scale.
The length of each bar, if presented horizontally, or the height of each bar with the value displayed at its base, if presented vertically, represents a numeric value. Histograms are widely used in statistical quality control. In a manufacturing or research process, for example, they suggest the outline of the frequency curve that represents the underlying probability law that governs the variation pattern.

This chapter presents a Turbo Pascal version of a program that accepts information entered at the keyboard and converts it into a string or series of asterisks that represents the bar. The choice of the character that simulates a bar is entirely a matter of individual preference. Although the asterisk is very popular for such applications, you may prefer to use a capital X, or a capital O, or even a capital H. You can experiment with various characters and use the one that, in your view, is most acceptable.

As the program is listed in this chapter, each histogram value entered at the keyboard is computed by the program in such a way as to produce a string of asterisks equal to half the value that is entered. The series or strings of asterisks are displayed simultaneously on the screen and, if it is turned on, at the printer. The program has a built-in error trap to restrict to 100 the maximum value of any individual entry of data that will be converted to asterisks.

```
 >>>>>>>>>>>>> HISTOGRAM MAKER <<<<<<<<<<<<<<<<
 ENTER THE LENGTH OF THE HISTOGRAM BAR.  
 (Enter -1 to END the Program):  50

 50.00 ***********************

 ENTER THE LENGTH OF THE HISTOGRAM BAR. 
 (Enter -1 to END the Program):  25.55

 25.55 **************
```

As each bar is displayed at the screen, the value entered and the line of asterisks are duplicated at the printer (if it is turned on) just as they appear at the screen. The printout shows the bars alongside each other. This makes the printout adaptable for inclusion in a document. The value-scale of the bar is deliberately limited to 100 maximum, which prints a total of 50 asterisks. This ensures that the string of asterisks, the simulated bar, will not run off the right edge of the printed page, or be wrapped.

SAMPLE HISTOGRAM PRINTOUTS

Two examples of printouts generated by HISTOGRAM are included here. The first, Fig. 3-1, represents a display of individual quantities that create a typically irregular pattern. The second, Fig. 3-2, is an example of a distribution of data that forms a bell curve, as might be found in a statistical analysis.

The program is surprisingly short for all the work it performs. The code is commented to aid your understanding and to guide you in revising it to meet your special requirements.
Fig. 3-1. A sample run of HISTO.

Fig. 3-2. A sample run of HISTO displays statistical data that takes on the appearance of a bell curve.
CODE LISTING FOR HISTOGRAM

( HistoGram generates and prints histograms based on data entries. )

PROGRAM HistoGram;
uses PasPrinter;  { Invoke the library unit, PasPrinter. }

VAR
    Ch      : Char;
    J,Y     : Integer;
    Number  : Real;

BEGIN { HistoGram }
    GotoXY(20,5);
    Write('!!! TURN ON THE PRINTER, PLEASE !!!');  { It's essential!}
    GotoXY(20,10);
    Write('Press any key when ready to continue.'); { Any key will do. }
    Read(Ch);
    ClearScreen;
    GotoXY(10,5); { Start at column 10, row 5. }
    Write('******************** ***>> ');
    Write('HISTOGRAM') ; { Start the banner display. }
    Write(' <<*********************');
    WriteLn;
    REPEAT { Continue until the UNTIL argument is satisfied. }
        WriteLn;
        WriteLn('Enter the Length of the bar.'); { Maximum value is 100. }
        Write('(Type -1 to END the program): '); 
        Read(Number);
        IF (Number >= 0) AND (Number <= 100.0) THEN
            BEGIN
                WriteLn; { Or go to the next IF...THEN. }
                WriteLn('Enter the Length of the bar.'); { Format 6 digits; 2 decimals. }
                Write( Printer, Number:6:2,' : ');
                Y := Round(Number / 2); { Divide Number by 2 to fit space. }
                FOR J := 1 TO Y DO
                    BEGIN
                        Write( 'X'); { Show the bars: "X" }
                        Write( Printer, 'X'); { on the screen and }
                        WriteLn(Printer); { on the printout. }
                    END;
                WriteLn;
                WriteLn(Printer); { Skip a line at the printer. }
                END;
        IF (Number > 100.0) THEN { A simple error trap. }
            WriteLn(' ***Maximum Length Is 100 ***');
        UNTIL (Number < 0); { Of course, any negative }
                         { value terminates Histo. }
    END. { of HistoGram }
Chapter 4

CONTRIBUTION: Determining the Contribution to Profit

There are many ways to measure the value of a product or service to the company marketing them. In taking such measurements, arithmetic and judgment must be combined to give the manager of a business operation a reasonable appraisal of the contribution a particular product or service makes to the company’s financial or market position. The appraisal is then used in making dynamic decisions concerning actions to be taken that can dramatically affect the life of a specific product or service and, of course, the welfare of the company.

The judgment factors are human considerations and are not left to the computer. On the other hand, the arithmetic can be more rapidly and possibly even more accurately dealt with by a computer operating with appropriate software.

Surprisingly, few companies know the specific contributions that any one product makes to overall profitability. If a product is not meeting the targets for profit, the conclusion appears to be obvious. Get rid of it! Remove it from the catalog! Profit targets, however, are often set arbitrarily, on the basis of intuition and gut feel rather than as the result of a comprehensive study and analysis of the market and the competition. The decision to continue or remove a product offering must take into account the real contributions it makes to the company’s overall operation. One important method for determining this is called contribution analysis.

Contribution is the selling price of the item less the variable costs associated with its production and marketing. Variable costs are those directly affected by the production and sales volumes of the item. They include materials, labor, shipping, royalties paid, commissions paid, and other miscellaneous costs of manufacturing and marketing that can be attributed to or that vary with quantity or volume.
Why is contribution analysis important? Given the limits of resources and fixed costs, all businesses have a finite capacity to produce. For example, a dealer may be able to sell a variety and a varying quantity of video products with no significant change in fixed overhead costs, assuming he does not have to rent additional storage space and thereby increase his fixed costs. A consultant may be able to do several different kinds of studies and reports for his clients without affecting his overhead costs for rent, utilities, and administrative labor. In fact, if he has no work at all, his fixed costs may continue at the same level, unrelated to the amount of work at hand. A product manufacturer is in the same situation with respect to fixed costs, where overhead remains essentially the same for a specific facility, regardless of the level of production.

In these cases, is contribution to profit the major measure of merit? Isn't contribution to paying for overhead also important in measuring the merits of a product? The answer is obvious. Contribution to paying for overhead makes good sense. It is however, too often overlooked. Managers often try to maximize sales for a specific product or service purely on the basis of measured profit dollars.

With two nearly identical items on the shelf, one priced at $40 and one at $50, a manager might be tempted to push the $50 item, deemphasizing the $40 one. This could be an error, if contribution analysis is overlooked. Assume that the $50 item cost $30 to move off the shelf after all variable costs have been paid. The contribution is $20 each. Assume that the $40 item's variable costs are $15, resulting in a contribution of $25. On this basis it would be better to concentrate on moving the $40 item, even to the extent of placing a relatively larger sales commission on it than on the higher-priced item. On the basis of these assumptions, it would be more prudent to concentrate on contribution than to focus on sales dollars.

For various reasons, it is not always possible to discontinue certain items if they are needed to round out the line. It is not always practical to ignore the remainder of the line and concentrate on only those with maximum contribution. Total contribution over a time period must be considered.

If the manager can move more of the $50 items than of the $40 ones during a special sales promotion, if the higher-priced item can sell (turnover) faster than the $40 item, for whatever reason, it is possible the $50 item will make a greater total contribution than the lower-cost item. The point is that maximization of sales dollars should not be used as the only measure of merit for a product's performance in the marketplace or for deciding whether or not to continue a product. By making comparisons for contribution among various products, it becomes relatively easy to resolve questions concerning the life of a product and where to place sales emphasis. The same examination for contribution should be made when considering adding new products or services to the catalog.

Perhaps it's a fundamental mistake to measure the performance of sales and merchandising managers purely on the basis of gross sales. It would seem to make good sense to measure their performance on the basis of contribution to profit.

This program, CONTRIBUTION, provides a means for doing a rapid analysis of the contribution to profit of a single product or service. By running the program several times, once for each item in the line, a comparison table can be developed that helps to resolve the problem. Also, by changing the input data for a single product, the right
mix of variable-cost elements can be developed and analyzed to optimize and attain an appropriate target for a desired level of contribution.

RUNNING THE CONTRIBUTION PROGRAM

Here is a sample run of CONTRIBUTION, with the screen’s queries shown at the left and the typed-in responses underscored at the right:

WHAT IS THE NAME OR MODEL OF THE UNIT? WIDGET-1A
WHAT IS THE SELLING PRICE PER WIDGET-1A? 59.95
VARIABLE COST DETAIL:
WHAT IS THE COST FOR MATERIALS PER UNIT? 11.23
WHAT IS THE DIRECT LABOR COST PER UNIT? 4.87
WHAT IS THE SHIPPING COST PER UNIT? .92
WHAT COMMISSION DOLLARS ARE PAID PER UNIT? 5.99
WHAT ROYALTIES ARE PAID PER UNIT? 1.79
ENTER ANY MISCELLANEOUS DIRECT COSTS: 0

After making the last entry and pressing the Return key, the screen clears and instantly displays a summary of the data that were entered, plus the calculations that have been programmed:

==============================================================================
WIDGET-GADGET -- CONTRIBUTION ANALYSIS
==============================================================================
SELLING PRICE PER UNIT: $59.95
LESS THE VARIABLE COSTS:
   MATERIALS:    $11.23
   DIRECT LABOR: $4.87
   SHIPPING:     $0.92
   COMMISSIONS:  $5.99
   ROYALTIES:    $1.79
   MISC. COSTS:  $0.00
TOTAL VARIABLE COSTS: $24.80
UNIT CONTRIBUTION TO PROFIT: $35.15

DO ANOTHER CALCULATION? (Y/N):

Thus, quickly, the story is unfolded for WIDGET-GADGET. Each item of WIDGET-GADGET contributes $35.15 to the company’s profitability. One product or service can be compared with another as a means for quantifying the contribution-to-profit each makes. The merits of the product or service can then be more accurately evaluated. Recognize that the explicit value of the contribution-to-profit is only one of the factors that a prudent person uses in a decision-making process concerning whether or not a specific product or service is “doing well,” or is beneficial to company’s overall interests.
The code provided with this chapter is intended to fulfill the arithmetic requirements. The code and the algorithms are quite straightforward and can, at the very least, serve as models upon which more complex, more-powerful programs can be built for determining the dollars-and-cents contribution to profit being generated by a product or a service.

**CODE LISTING FOR CONTRIBUTION**

```pascal
{ Contribution calculates the contribution to profit made by a product or a service. Costs are totalled and then subtracted from the selling price. The amount that is the difference between selling price and total cost is the amount considered to be the "contribution." }

PROGRAM Contribution ;

TYPE
    StringType = STRING[80] ;

VAR
    Ch : Char ;
    Commission : Real ;
    DirLabor : Real ;
    Line : Integer ;
    Materials : Real ;
    Misc : Real ;
    Name : StringType ;
    Royalties : Real ;
    SellPrice : Real ;
    ShipCost : Real ;

PROCEDURE Draw_A_Line ;

BEGIN { Draw_A_Line }
    FOR Line := 1 TO 60 DO { Draw a character 60 times in a row. }
    BEGIN { Here's the character, "-". You can change it. }
        Write('---') ;
    END ;
    WriteLn;
END ; { of Draw_A_Line }

PROCEDURE SignOn ; { Display the startup message. }

BEGIN { SignOn }
    GotoXY(12,5) ;
```

30
PROCEDURE StartingConditions;  { Get the input data. }
BEGIN  { StartingConditions }
ClearScreen;
Write('What is the name or model of the unit?  ') ;
ReadLn(Name);
Write('What is the selling price per ',Name,'?  ') ;
ReadLn(SellPrice);
Write('What is the cost of materials per unit?  ') ;
ReadLn(Materials);
Write('What is the direct labor cost per unit?  ') ;
ReadLn(DirLabor);
Write('What is the shipping cost per unit?  ') ;
ReadLn(ShipCost);
Write('What commission-dollars are paid per unit?  ') ;
ReadLn(Commission);
Write('What royalty-dollars are paid per unit?  ') ;
ReadLn(Royalties);
Write('Enter any miscellaneous direct costs:  ') ;
ReadLn(Misc);
ClearScreen;
END ;  { of StartingConditions }
WriteLn('Selling Price/Unit: ', SellPrice:10:2); 

WriteLn('Less the Variable Costs:');  
( List the cost-categories formatted to )  
( to two decimal places, as above. )  
WriteLn(' Materials: ', Materials:10:2);  
WriteLn(' Direct labor: ', DirLabor:10:2);  
WriteLn(' Shipping: ', ShipCost:10:2);  
WriteLn(' Commission: ', Commission:10:2);  
WriteLn(' Royalties: ', Royalties:10:2);  
WriteLn(' Misc. Costs: ', Misc:10:2); 

( Now add together the above six categories. )  
Sum := Materials + DirLabor + ShipCost +  
Commission + Royalties + Misc; 

( Display the calculated sum of Variable Costs. )  
WriteLn('Total Variable Costs: ', Sum:10:2);  
WriteLn;  

( Contribution to profit is the Selling Price )  
( minus the Variable Costs. )  
WriteLn('Unit contribution to profit: ', SellPrice - Sum:10:2);  
WriteLn
END;  

BEGIN  
SignOn;  
REPEAT  
StartingConditions;  
Show_Results;  
Draw_A_Line;  
WriteLn;  
WriteLn;  
Write('Do another calculation? (Y/N): '); ReadLn(Ch);  
UNTIL (Ch = 'N') OR (Ch = 'n');  
END.
Chapter 5

DAYS CREDIT: Measuring Cash in Float

It's the nature of our economy to do business on credit. And the tendency toward credit purchases, whether through personal credit cards, open or self-funded revolving-credit accounts is on the rise. It may be an oversimplification, but it is realistic to say that because of rising credit-sales more and more business operations are waiting for their money, which means some of their money is tied up and not readily available to the business for its own use. The period of waiting for the money is the collection period. It is important to know how long, on the average, the collection period is.

When interest charges are at high rates (especially high for small businesses, which usually must pay a premium above the prime rates on bank loans), the days that people take to "live" on their credit can become a significant cost factor for doing business.

By periodically measuring the days-of-credit, the manager can keep track of the cash movement—the credit extended by the business to its customer accounts (not the credit that suppliers extend to the business). That's the cost of doing business on credit.

This chapter's program, DAYS CREDIT, is important to the business operation's health and important to the credit manager who must make yes/no decisions on continuing a line of credit to a customer. (It also provides a valuable and easily comprehended exercise in writing code for the Turbo Pascal compiler.)

The DAYS CREDIT program queries the user for the number of days to be considered in doing the calculations. Ninety days, a fiscal quarter, is the usual number. Then, the user is asked to enter the dollar value of sales made during the period. Naturally, because you are dealing with accounts receivable, you consider only sales made on credit. Cash sales are not "accounts receivable." They are closed when paid, which is immediately on delivery of the goods or services to the buyer. The last two questions
are for the dollar value of accounts receivable at the start and at the end of the period.

A business normally selling on terms of net-30 would expect to have the previous 30-days’ sales-dollars outstanding. But if sales that occurred 40 or 50 days earlier are still displayed on the balance sheet, the indication is that some of the customers are experiencing some difficulty in meeting their commitments or may be abusing their credit privileges. At any rate, an effective job of collection is not being done. This affects cash flow.

CLASSIFYING LATE PAYERS

A measure of the effectiveness of a business’s credit terms and collections can be made by aging the accounts, that is, by classifying individual accounts into tabular columns such as 30 days, 45 days, 60 days, and 90 days, each column indicating the number of days past the due date. The rows contain the names of the accounts. Accounts past due, depending on the net terms you allowed, are totaled in dollars under the appropriate column headings. The table of data then pinpoints the accounts that need attention.

ACCOUNTS RECEIVABLE AND LIQUIDITY

If your enterprise sells on credit terms, accounts and notes receivable are a significant part of working capital. In assessing the quality of your working capital, it is important to gain some knowledge of the quality and liquidity of the receivables. Quality and liquidity of accounts receivable are affected by the rate of turnover. The longer receivables remain outstanding, the lower the chance becomes of collecting them in full. Turnover is a reflection of the age of the receivables when compared with an expected turnover rate determined by the credit terms that have been granted.

USING THE TURNOVER RATIO

The receivable turnover rate is a ratio computed by dividing the net credit sales by the average accounts receivable for a specific period. To make the formula manageable, and because it is today’s practice to buy and sell primarily on credit, all sales are usually included in the numerator, which is net credit sales.

If you can separate cash sales from credit sales, so much the better for accuracy. Cash sales do not create accounts receivable. Average accounts receivable is derived by adding the outstanding receivables at the start of a period to the receivables outstanding at the end of the period and dividing by two. Using monthly or quarterly sales figures is proper. The use of longer periods may introduce distortion unless the average is more carefully calculated for the extended period. Notes receivable generated by normal sales activities also should be included in the accounts receivable total in computing the turnover ratio.

MAKING COMPARISONS TO DETERMINE TRENDS

As with most financial data, and certainly with ratios, it is important to watch for trends and analyze the causes and effects. The average receivables turnover rate
indicates how many times, on the average, the receivables revolve, that is, how often they are generated and collected during the year. For example, if net credit sales for a period are $60,000, accounts receivable at the start of the same period are $7,500, and accounts receivable at the end of this same period are $12,500, then the receivable turnover ratio is:

\[
\frac{60000}{(7500 + 12500)/2} = \frac{60000}{10000} = 6 \text{ times}
\]

The “6 times” figure can be used to compare the selected period with previous periods for trend analysis. There is, however, greater value in using the turnover ratio to calculate the collection period, which reflects the number of day’s sales in accounts receivable.

THE COLLECTION PERIOD

How many days, on the average, does it take to collect accounts and notes receivable? The number of days is computed by dividing the number of days in the accounting period under review by the receivable turnover ratio. It has been popular practice among accountants to use the number 360 when doing calculations for a full year. The number 360 was easier to deal with, manually, than 365 or 366 (leap year). With a computer, however, it is just as easy to use the actual term of the calendar year. Using the data developed in the preceding example and a hypothetical calendar year, the following formula can be developed:

\[
\frac{\text{Number of Days’ Sales}}{\text{Average Accounts Receivable Turnover}} = \text{Collection Period}
\]

\[
\frac{365}{6} = 60.8 \text{ days}
\]

It’s permissible to use any other value for the number of day’s sales as long as you are consistent in your treatment. Otherwise, valid comparisons cannot be made. Collection periods can be compared to industry averages or to credit terms granted by competitors and their receivables (if such valuable information can be obtained).

RUNNING THE DAYS CREDIT PROGRAM

DAYS CREDIT enables you to determine rapidly the data for any specific period of time, for any account, or for all accounts. To run the program, you need to know and enter only the net credit sales for the period and the starting and ending receivables. For example:
COLLECTION PERIOD FOR ACCOUNTS RECEIVABLE

(CREDIT SALES ONLY)

HOW MANY DAYS ARE IN THE SALES YOU WILL COMPUTE? 91
WHAT ARE THE NET SALES ($) ON CREDIT FOR THE PERIOD? 60000
AT THE START OF THE PERIOD, WHAT WERE ACCOUNTS RECEIVABLE? 20000
AT THE END OF THE PERIOD, WHAT WERE ACCOUNTS RECEIVABLE? 40000

Press the Return key on the Macintosh keyboard after the last entry, and the computer takes over. The screen clears and displays these additional lines of calculated data:

AVERAGE ACCOUNTS RECEIVABLE TURNOVER RATIO: 2.0
AVERAGE DAILY SALES FOR THE PERIOD: $659.34
DAYS SALES IN ACCOUNTS RECEIVABLE (COLLECTION PERIOD): 45.5

Try the same calculation again, but this time enter 180 instead of 91 for the number on the first line. The turnover ratio doesn’t change, of course, because it doesn’t consider “days’ sales” at this point in the calculation. The average daily sales for the period decreases to $333.33, approximately half that of the 91-day period because you have not adjusted the net-sales dollars entry; sales were effectively half as high, and the collection period increases to 90.0 days. Try 360 days and 365 days for the year’s computation period, whichever one makes you feel most comfortable or agrees with past calculations. Adjust the net sales figure and the accounts receivables to represent a full year of operations.

When making comparisons, always use the same relative period; for instance, the same 91-day period of the previous year should be compared with the current 91-day period.

The program directs all output to the default device, CONSOLE, which is the video monitor. It is a useful exercise to practice revising the code to make the program direct its output to the printer device. (Don’t forget the call to the PASPRINTER unit.) Try adding code to send the program’s output to both the CONSOLE and the PASPRINTER simultaneously.

CODE LISTING FOR DAYS CREDIT

{ DaysCredit analyzes your accounts-receivable. }

PROGRAM DaysCredit ;

TYPE
  StringType = String[80] ;
VAR
    Ch          : Char;
    End_Receivables : Real;  { Real enables decimal values. }
    Fiscal_Period : Real;
    Line_Length  : Integer;
    Sales_Dollars : Real;
    Start_Receivables : Real;
    Turnover_Ratio : Real;

PROCEDURE Deco_Line;  { Decorative line used as a border. }
    BEGIN  { Deco_Line }
        FOR Line_Length := 1 TO 80 DO
            BEGIN
                Write(Chr(124))  { The "stick" character for video display. }
                END;
        END;  { of Deco_Line }
    END;  { of Deco_Line }

PROCEDURE Sign_On;  { Display the startup message. }
    BEGIN  { Sign_On }
        GotoXY(15,6);
        FOR Line_Length := 1 TO 50 DO
            BEGIN
                Write(Chr(124));  { Write the character 50 times on one line. }
                END;
        GotoXY(20,8);
        Write('DAYS CREDIT -- AVERAGE AGE OF RECEIVABLES');  { The title. }
        GotoXY(15,10);
        FOR Line_Length := 1 TO 50 DO
            BEGIN
                Write(Chr(124))
            END;
        GotoXY(15,14);
        Write('Press Return to continue...');
        Read(Ch);
        ClearScreen;
    END;  { of Sign_On }

PROCEDURE Get_Data;  { Get the data from the keyboard. }
    BEGIN  { Get_Data }
        Deco_Line;  { Call the Deco_Line PROCEDURE; then return here. }
        GotoXY(20,2);
WriteLn('COLLECTION PERIOD FOR ACCOUNTS RECEIVABLE')
Deco_Line

GotoXY(20,5)
WriteLn('(Credit Sales_Dollars Only)')

WriteLn('(Using 'WriteLn' here is simpler than calculating 'GotoXY'.)
Write('Enter the number of days in the period to be computed: ')
ReadLn(Fiscal_Period)

WriteLn;
Write('Now, enter the net sales ($) on credit for the period: ')
ReadLn(Sales_Dollars)

WriteLn;
Write('And accounts receivable ($) at the start of the period: ')
ReadLn(Start_Receivables)

WriteLn;
Write('Then the accounts receivable ($) at the end of the period: ')
ReadLn(End_Receivables)

WriteLn;
WriteLn
END; \ ( of Get_Data )

PROCEDURE Make_Report; \ ( Generate the report with two-place arithmetic. )
BEGIN \ ( Make_Report )
WriteLn('Average accounts-receivable turnover ratio: ',
      Turnover_Ratio:8:2) ;
WriteLn;
WriteLn('Average daily sales for the period: $',
       Sales_Dollars / Fiscal_Period:8:2) ;
WriteLn;
WriteLn('Days sales in accounts receivable: ',
       Fiscal_Period / Turnover_Ratio:8:2) ;
WriteLn;
Deco_Line \ ( Display the border at the bottom of the report. )
END; \ ( of Make_Report )

BEGIN \ ( main part of DaysCredit )
   \ ( Note that this main part of the program calls )
   \ ( the PROCEDURES in their natural sequence. )
Sign_On; \ ( PROCEDURE-call to display the program's name. )
REPEAT \ ( REPEAT the program UNTIL... (see below) )
   ClearScreen;
   Get_Data ;
{ Initialize, or assign a starting value to Turnover_Ratio. }
Turnover_Ratio := Sales_Dollars / ((Start_Receivables +
End_Receivables) / 2);

Make_Report;
WriteLn;
WriteLn;
Write('Do another? (Press ''N'' for NO. ''Y'' for YES): ');
Read(Ch);
UNTIL Ch IN ['N','n']    { 'N' ends the program immediately. }
END.    ( of DaysCredit )
You may be able to maintain your checking account records more easily by hand than with a computer, especially if you have very few checkbook transactions to record. There is, however, always something for the would-be programmer to learn when code is written to perform tasks, especially if the tasks are familiar ones. So, I offer the source code in Turbo Pascal for this CHECKBOOK program without suggesting you actually use it for your own checking account. The program, however, does work well. I have heard that several small-business owners are using it to maintain and update records of their companies’ checking accounts. Even if you don’t put it to work as they have, you’ll certainly find the code interesting.

Although every business operation has a checking account, not all business managers with decision-making responsibility and accountability have easy access to the status of the company’s checking account. There are many questions whose answers should be but are not always within ready reach. Is the balance being maintained at a healthy level for the specific type of operation? When was the last deposit made? How much was it? What are the ledger and journal references (or locations) for the detail to support the sources of funds that were deposited? How many checks have been written during the same period of time? What was the total amount of all the checks written? What are the check numbers? What was the beginning balance? Was the account overdrawn at any time? What is the ending balance? What should your direction be with respect to managing the checking account? Should you change your self-designed standards for the minimum balance or for the number of checks you write during a given period? Do you have a problem at all? If a problem seems to be developing, how can you spot it in time to “cut it off at the pass?”
These questions, if not answered promptly and accurately, can present problems for the manager to solve in a crisis mode. And many decisions made during a crisis are bad or faulty. Such a situation, with its high-risk factors, can be prevented by keeping an up-to-date checking-transaction report close at hand. The computer, with appropriate software, is an ideal tool for providing the manager with data on a real-time basis.

**A FINANCIAL SNAPSHOT**

This program generates a snapshot view of the status of the company's checking account; it can show deposits and checks written for any point in time or for any fiscal period. It is designed to identify potential problems before they get out of control and to help point the direction for constructive management decision making.

When the deposit slip is prepared for checks that have been received and are ready for entry into the company's account, the bookkeeper, secretary, or you can enter the information into the computer using this program. In no time at all, the computer produces a single-page printed report that is a summary of the status of the checking account. Also, the report provides room to enter references to more-detailed records in the company's journals and ledgers. Thus, if the manager or the company's accountant wants to audit the books, the locations of the supporting details are in the report and point an auditor in the right direction.

Normally, as checks are written, it is common practice to enter the supporting details in a check register. As the manager, you may not need or want all the details. Very often, however, you will want a snapshot view of the number of checks written and a clear statement of what impact they had on the account's bottom-line balance on a specific date.

**MONITORING AND WATCHING FOR TRENDS**

You can use this program to determine whether you have problems or appear to be developing problems by regularly running the program and saving the hard copy reports in a binder or a folder. Read the important data they generate. Watch for trends or sudden changes in balances and in checks written, then take whatever action may be indicated to correct or preclude a crisis situation. Continue to generate and read the reports as a means of monitoring checking account activity.

This same program, CHECKBOOK, is well suited for use with your personal checking account. You can ignore the query that the computer makes for "journal or ledger references" in connection with deposits. Or you can insert coded responses of your own invention. You can insert abbreviations or full names as an aid to identifying the sources of funds. You could even enter individual dollar amounts. You can type in several lines of information, letters, numbers, and punctuation marks before the printer shouts "enough, already!"

Comments have been placed within the code to explain statements that might not be self-evident. It is also good practice to include comments in the source code you may write or in any modifications you may introduce so that, should you set the program aside for a while, you will have minimum difficulty in recalling the meanings or the purposes of the statements you wrote. It is easy to forget how or why a specific line or a series of lines were written. The comments eliminate this problem.
Each of us handles or treats comments differently. Note the way I did it in this program. Then, as an exercise after you have worked with the program for a bit, try to improve on the quantity and quality of the comments to suit your preferences or needs.

**RUNNING THE CHECKBOOK PROGRAM**

This sample run demonstrates how to use the program and what the printout looks like. Type in the code, then compile and run CHECKBOOK.

First, the SignOn procedure is called. It displays the *banner* and the name of the program, which appears at the center of the screen.

```
==================================================================
CHECKBOOK -- CHECKBOOK-TRANSACTION REPORT
==================================================================
```

The following advisory is displayed below the banner:

** *** THE PRINTER MUST BE TURNED ON ***

And, directly below it is the message:

```
... and press Return to continue ...
```

The printer, if you have indeed turned it on, automatically starts printing the heading for the printout. (If you have not turned the printer on, the program will run with information appearing only on the screen.) When the screen is cleared the GetTheFacts Procedure is called immediately.

In the following example, the words at the left represent the program's requests for information that appear on the screen. The underlined information represents the responses that you enter through the keyboard. Of course, you must press the Return key after completing the answer to each question. Do not use commas or dollar signs when entering financial information.

**Enter the date of the first transaction.**
(Use the Format MM/DD/YY): **06/01/87**

**Enter the date of the last transaction.**
(Use the Format MM/DD/YY): **06/31/87**

**Type the opening ($) balance:** **21435.29**

The information you enter is now being sent to the printer, and the next three lines displayed on the screen offer a selection of two categories in which to enter data or to quit the program.
To enter the total of a new deposit made... Type 'D'.
To enter total of all checks written... Type 'C'.
To quit this program and return to the desktop... Type 'Q'.

Then the screen instructs you to make a selection:

Enter one letter (D, C or Q): D

The selection is made through a CASE statement. If any key other than D, C, or Q is pressed, an error message appears on the screen. Then, the previous four lines reappear from which the selection of D, C, or Q is made. The process will continue endlessly until one of the three choices, D, C, or Q, is made. I choose D, the MakeDeposit procedure, and the program continues:

Enter the DATE of the deposit: 06/01/87
And the ($) amount of the deposit: 32796.55
Give the Journal references:
(Space between entries, please): ABC12 ABC13 CBA24

As soon as you press the Return key, the printer goes to work again and the screen simultaneously displays the new balance. The program asks you to select D, C, or Q. This time, in your sample run, you'll enter checks-written information. Select the letter C. The program calls the WriteChecks Procedure and continues:

What is the ($) amount of checks written? 39842.45
How many checks were written? 4
What are the check numbers?
(Put a space between each number): 1004 1005 1006 1007

The printer goes to work again, and adds the above information to the hard copy that is being generated:

Total amount of Checks written: $39842.45
The number of Checks written: 4
The numbers of the Checks are: 1004 1005 1006 1007

The familiar selection line reappears. This time press the Q and the Return keys to quit the program. The WrapUp Procedure is called. The printer responds, and the screen displays a message:

----> ENDING BALANCE IS $ 14389.39 <----

If you had entered data that caused the account to be overdrawn or had created a negative balance, the ENDING BALANCE message would have displayed a negative number representing the amount of the overdraft with a warning notice:
SAMPLE PRINTOUTS OF CHECKBOOK

Figure 6-1 and Fig. 6-2 show two sample reproductions of the hard copy printout of the information entered above. Note that there are differences in the presentation and layout because of the fact that the printout combines the questions and answers that were asked at the screen and responded to at the keyboard.

Figure 6-1 shows what can happen when you enter the RECORD OF DEPOSITS ahead of the RECORD OF CHECKS WRITTEN. Thus, the BALANCE was substantial,

CHECKBOOK -- CHECKBOOK-TRANSACTION REPORT

This report is from: 06/01/87 to: 06/30/87

--- Opening balance is: $ 21435.29 <---

RECORD OF DEPOSITS

The 06/01/87 deposit was: $ 32796.55
Journal or Ledger references are: ABC12 ABC13 CBA24

The balance now is: $ 54231.84

RECORD OF CHECKS WRITTEN

Total amount of Checks written: $ 39842.45
The number of Checks written: 4
The numbers of the Checks Are: 1004 1005 1006 1007

The balance now is: $ 14389.39

--- THE ENDING BALANCE IS: $ 14389.39 <---

Fig. 6-1. This sample run of CHECKBOOK clearly shows there was enough money in the account to cover checks that were written.
Fig. 6-2. The checking account was temporarily overdrawn, as this sample run of CHECKBOOK points out.

more than adequate to cover the amounts written for the checks. In the second sample, Fig. 6-2, you entered the checks before the deposits. The effect was to overdraw the account, at least until the deposits were entered into the record. This overdrawn condition is called out as a warning when the balance is negative.
CODE LISTING FOR CHECKBOOK

{ CheckBook record printing. }

PROGRAM CheckBook;
uses PasPrinter;

VAR
  Balance : Real; { The account balance. }
  Ch       : Char; { The user's menu choice. }
  EndPeriod: String[80]; { The ending date. }
  L        : Integer; { Utility for line lengths. }

{ FORWARD declarations are in alphabetical }
{ order only for convenience. }

FUNCTION MakeDeposit: Real; FORWARD;
FUNCTION WriteChecks: Real; FORWARD;

PROCEDURE SignOn; { Generate the startup message. }

BEGIN { SignOn }
  GotoXY(10,8); { Locate the next command at column 10, row 8. }
  FOR L := 1 TO 60 DO { Do the next action 60 times. }
    BEGIN
      Write(CHR(179)) { The border is composed of graphics char #179 }
    END;

  GotoXY(22,10); { Start the next display at column 22, row 10. }
  Write('CHECKBOOK -- CHECKBOOK-TRANSACTION REPORT'); { Title. }
  GotoXY(10,12); { Column 10, row 12. }
  FOR L := 1 TO 60 DO { Same as above. }
    BEGIN
      Write(CHR(178));
    END;

  GotoXY(20,16); { Column 20, row 16. }
  Write('*** THE PRINTER MUST BE TURNED ON ***');

  GotoXY(22,20); { Column 22, row 20. }
  Write('...and press Return to continue...');
  Read(Ch); { Get the keyboard character; do not display it. }
  FOR L := 1 TO 50 DO { Now, send to the printer. }
    BEGIN
      Write(Printer,'='); { Print a border of equal signs. }
    END;
PROCEDURE GetTheFacts ;  { Get the first and last transaction }  { dates and the beginning balance. }

VAR
  StartPeriod : String[8] ;
  StopPeriod : String[8] ;

BEGIN  ( GetTheFacts )
  ClearScreen ;
  GotoXY(1,5) ;
  WriteLn('Enter the date of the first transaction.' ) ;
  Write('  (Use the Format MM/DD/YY): ' ) ;
  ReadLn(StartPeriod) ;  { Get the response from the keyboard. }

  WriteLn ;  { Skip a line on the video display. }
  WriteLn ;  { Again. }
  WriteLn('Enter the date of the last transaction.' ) ;
  Write('  (Use the format MM/DD/YY): ' ) ;
  ReadLn(StopPeriod) ;  { Get the response from the keyboard. }

  WriteLn ;
  WriteLn ;
  Write('  Type the opening balance ($) : ' ) ;
  ReadLn(Balance) ;  { Get the response from the keyboard. }

  WriteLn(Printer) ;
  WriteLn(Printer, 'This report is from: ',StartPeriod,' to: ',StopPeriod) ;
  WriteLn(Printer) ;
  WriteLn(Printer, 'This report is from: ',StartPeriod,' to: ',StopPeriod) ;
  WriteLn(Printer, '--- Opening balance is: $',Balance:10:2, ' <---') ;
END ;  { of GetTheFacts }

FUNCTION MakeDeposit ;  { Make a deposit and update the balance. }

VAR
  DepAmt : Real ;
DepDate  : String[8] ;
JournalRef  : String[80] ;

BEGIN  { MakeDeposit }
  ClearScreen ;
  GotoXY(5,5) ;
  Write('Enter the DATE of the deposit: ') ;
  ReadLn(DepDate) ;

  GotoXY(5,7) ;
  Write('And the ($) amount of the deposit: ') ;
  ReadLn(DepAmt) ;

  GotoXY(5,9) ;
  WriteLn('Give the Journal references.') ;
  Write(' (Space between entries, please): ') ;
  ReadLn(JournalRef) ;

  WriteLn(Printer) ;  { Update the listing }
  WriteLn(Printer, 'RECORD OF DEPOSITS') ;
  WriteLn(Printer, '---------------------------') ;
  WriteLn(Printer, 'The ', DepDate, ' deposit was: $', DepAmt:10:2) ;
  WriteLn(Printer, ' Journal or Ledger references are: ', JournalRef) ;
  WriteLn(Printer) ;

  MakeDeposit := DepAmt ;  { MakeDeposit gets the value of DepAmt. }
END ;  { of MakeDeposit }

FUNCTION WriteChecks ;  { Enter the checks and report the balance. }

VAR
  CheckNumbers  : String[70] ;
  CheckAmt  : Real ;
  NumChecks  : String[70] ;

BEGIN
  ClearScreen ;

  GotoXY(1,5) ;
  Write('What is the ($) amount of checks written? ') ;
  ReadLn(CheckAmt) ;

  WriteLn ;
  WriteLn ;
  Write('How many checks were written? ') ;
  ReadLn(NumChecks) ;

  WriteLn ;
WriteLn;
WriteLn('What are the check numbers?');
Write('('Put a space after each number): ') ;
ReadLn(CheckNumbers);

WriteLn(Printer);
WriteLn(Printer,'RECORD OF CHECKS WRITTEN');
WriteLn(Printer,'------------------------');
WriteLn(Printer,'The number of Checks written: ',NumChecks);
WriteLn(Printer,'The numbers of the Checks Are: ',CheckNumbers);
WriteLn(Printer);

WriteChecks := CheckAmt; { WriteChecks gets the value of CheckAmt. }

PROCEDURE WrapUp; { This procedure sends a closing message }
BEGIN { Wrapup }
WriteLn;
WriteLn;
WriteLn('---> THE ENDING BALANCE IS: $',Balance:10:2,' <----');
WriteLn(Printer);
WriteLn(Printer,'---> THE ENDING BALANCE IS: $',Balance:10:2,' <----');
WriteLn(Printer);
FOR L := 1 TO 50 DO
BEGIN
  Write(Printer,'=');
END;
WriteLn(Printer,Chr(12));
END; { WrapUp }

BEGIN { CheckBook }
SignOn; { Execute the SignOn PROCEDURE. }
GetTheFacts; { Get the command. Note that we double the }
{ apostrophes to indicate a real apostrophe. }
WriteLn;
WriteLn;
WriteLn('To enter the total of a new deposit made........Type 'D''.');
WriteLn('To enter the total of all checks written........Type 'C''.');
WriteLn('To Quit this program and return to the desktop...Type 'Q''.');
REPEAT { Open a loop until UNTIL is satisfied. }
WriteLn;
Write('Enter one letter (D, C or Q): ');
ReadLn(Ch); { Get the user's response. }

50
Calculation of the balance is done here to make it relatively easy to make modifications. MakeDeposit and WriteCheck get the information from the user about transactions. Then they return the number so the balance can be updated here. If appropriate, an OVERDRAWN warning is displayed and printed.

Note these multiple selections can be processed with the CASE statement. Options are paired in the table.

```sql
CASE (Ch) OF
    'D', 'd': Balance := Balance + MakeDeposit;
    'C', 'c': Balance := Balance - WriteChecks;
    'Q', 'q':
        (This is a NULL - we don't want to do anything.)
OTHERWISE
    WriteLn;
    WriteLn;
    WriteLn('Wrong letter selected! Please retype the letter!');
END;  (of the CASE statement.)

IF (Ch <> 'Q') AND (Ch <> 'q') THEN
    (Continue the program if the letter is not a Q or q.)
BEGIN
    WriteLn;
    WriteLn;
    WriteLn('The balance now is: $' ,Balance:10:2);
    WriteLn;
    WriteLn(Printer);
    WriteLn(Printer, 'The balance now is: $' ,Balance:10:2);
    WriteLn(Printer);
END;

IF (Balance < 0.00) THEN
    (Display a warning, if the account is overdrawn.)
BEGIN
    WriteLn;
    WriteLn;
    WriteLn('----- THE ACCOUNT IS OVERDRAWN! <-----');
    WriteLn(Printer);
    WriteLn(Printer, '----- THE ACCOUNT IS OVERDRAWN! <-----');
    WriteLn(Printer);
END;

UNTIL (Ch = 'Q') OR (Ch = 'q');
    (Stop if the user enters a Q or a q;)
    (and then call the WrapUp PROCEDURE.)
WrapUp ; { Call the WrapUp PROCEDURE. }
{ Eject the paper to the top }
{ of the next form. }

END. { of PROGRAM CheckBook }
Chapter 7

DEPRECIATION: Depreciating Your Capital Assets

There’s a variety of IRS-approved methods for calculating depreciation for tax-reporting purposes. As with all processes and procedures, however, they aren’t all applicable to specific situations. Tax regulations are dynamic. (Need I tell you?) They are constantly changing or being changed. It takes a specialist to keep up with the IRS regulations. Further, it takes a specialist to comprehend them and know which depreciation method is best suited to a company’s situation and needs, within the guidelines of the IRS.

In my use of the term “assets,” I refer to tangible items used in the operation of a business but not held or intended for sale in the regular course of the business. These are classified on the balance sheet as plant or fixed assets; they may also be identified as property, plant, and equipment. Specifically, such assets include furniture, tools, machinery, buildings, and land.

There are no standard criteria for the minimum length of life necessary for classification as a plant asset. The asset must be capable of repeated use and ordinarily is expected to last or have a useful life of more than one year. The IRS has a serious interest in the application and calculation of the depreciation of assets. The IRS publishes a system of Class Life Asset Depreciation that serves as a guideline for what the IRS views as the normal useful life of various classes of assets, which is the minimum number of years allowed for full depreciation of a capital asset.

In effect, depreciation is an annual expense allowed by the IRS for the exhaustion, wear and tear, and obsolescence of property used in your income-producing activity. You may not charge the entire cost of acquiring an asset to operating expenses in the
Year of acquisition, but must take into account the number of years the asset will con-
tinue to be useful.

The acquisition cost may be computed as a separate line item in the balance sheet,
and should be proportionally included in the profit and loss (P&L) statement as an an-
nual depreciation expense. The IRS is concerned with the annual effect on taxes and
the depreciation method used in the computations. Unless the IRS declares a change
in regulations (you should check with a competent certified public accountant, the for-
mulas used to compute annual depreciation are not complex.

First, determine the asset’s total cost (cost basis), including all costs directly related
to purchasing, shipping, and installing the asset in your plant. Repairs and maintenance
made or performed after the installation is completed are not included in the cost basis.
Next, determine the salvage value of the asset at the end of its useful life. The salvage
value, the asset’s worth at the time you dispose of it or at the end of its useful life while
you own it, may be nothing more than its material content and, therefore, is often re-
ferred to as scrap value. Subtract the scrap value from the total acquisition cost, and
then deduct the remainder through annual charges over the useful life of the asset as
defined by the IRS guidelines.

The annual depreciation charges may be equal or they may be prorated geometri-
cally, depending on the allowable and desirable impact on taxes. For balance sheet data,
the annual charges can be displayed as a cumulative or an accrued depreciation figure.

The useful life of the asset may not be the same as its physical life or its life if
it were owned by some other company. You may make your own estimate of useful
life considering the frequency with which you intend to use the asset, its age, its condi-
tion when you acquired it, the environment in which it will be used, the environment’s
effect on asset life, and obsolescence that may be brought about through technical
developments.

SCRAP VALUE

The IRS permits you to use scrap value or net salvage (less cost of removal) in
determining depreciation, but you must be consistent in your treatment of the subject.
Normally, an asset may not be depreciated below salvage or scrap value. In some cases,
end-of-useful-life value can be disregarded; thus increasing the annual depreciation.
However, if this value is disregarded, the cost basis of the asset will be reduced at the
end of its useful life. This can cause an increase in gain or a decrease in loss when
the property is disposed of or is traded in on the acquisition cost of a new or replace-
ment asset.

REAL ESTATE AND DEPRECIATION

Generally, land or the site on which the plant is located is not depreciated accord-
ing to useful life. Exceptions are land that has been acquired specifically for its mineral,
oil, timber, or other natural resources that diminish in value with time and use. In such
cases, the allowance rules are similar to but not the same as depreciation; they are known
as depletion allowances.

The IRS allows you to use any reasonable method to compute depreciation, provided
it is consistently applied. Among the many methods are the following: straight line,
annuity, appraisal, appropriation, combined depreciation and upkeep, composite life, age life, declining balance or diminishing provision, policy, replacement, retirement, sinking fund, unit summation, working hours, service capacity, and sum-of-years-digits.

Each previously listed method is essentially a variation of depreciation having diverse degrees of refinement and sophistication. Therefore, my program DEPRECIATION computes three commonly used methods. The straight line method calculates a linear progression, allowing the same amount to be deducted for the class-life of the asset until the declared scrap value is reached. Sum-of-years-digits and Declining Balance methods compute a larger amount of depreciation in the early years of asset life, with the amount that is depreciated diminishing each year until the scrap value is reached. In no case, as the display and printout of the declining balance table will remind you, can the depreciation be taken below the stated scrap value.

If you acquire or dispose of an asset during the year, regular depreciation is allowed only for that part of the year you own the property. To simplify the accounting, three accounting conventions are generally used: acquisitions and disposals are effective on the first day of the following month; on the first day of the month if they occur in the first half of the month; or as of the last day of the month if they occur in the last half of the month.

You should take the allowable depreciation deduction in each tax year. You may not deduct unclaimed depreciation for prior years in the current year or in a later tax year, only in the year of the actual depreciation. The IRS does permit you to file an amended return to claim depreciation that may have been overlooked. Any failure to take the allowable depreciation in the appropriate year or by an amended return reduces the asset's adjusted basis and increases your gain when you dispose of the property.

DEPRECIATION is purely for computing the numbers in helping you decide the method of depreciation that seems best suited to your needs. Again, because the rules are dynamic and frequently changing, your tax adviser, tax attorney, or certified public accountant should be consulted prior to filing your return with the IRS.

RUNNING THE DEPRECIATION PROGRAM

The program displays the information on the screen and prints a hard copy of the results of its calculations at the same time. The series of queries shown next is displayed on the screen. The responses at the keyboard are underlined.

Name the item to be depreciated: Macintosh Sooper-Doopers
Describe the Macintosh Sooper-Doopers: Round w/center cutout
Give the acquisition cost of Macintosh Sooper-Doopers: 1234.56
Enter the useful life in years: 5
Type the scrap value at the end of 5 years: 123.45
Factor (%) for Declining Balance Calculations is: 200

Immediately after you respond to the last query and press the Return key, the calculations are performed. The results begin to appear in table form on the screen. At the same time, the printer is producing a complete form, including the tables, the queries, and your actual responses. Three tables are produced: (1) the straight-line method, (2) the sum-of-years-digits method, and (3) the declining balance method with the Factor based on straight-line calculations.
DEPRECIATION SCHEDULES

Name of the item to be depreciated: Macintosh Sooper-Doopers
Macintosh Sooper-Doopers are described as: Round w/center cutout
Acquisition cost: 1234.56
Useful life is 5 years.
Scrap value at end of 5 years: 123.45

=============== STRAIGHT-LINE METHOD

<table>
<thead>
<tr>
<th>Year</th>
<th>Current Year Depreciation</th>
<th>Cumulative Depreciation</th>
<th>Book Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>1234.56</td>
</tr>
<tr>
<td>1</td>
<td>222.22</td>
<td>222.22</td>
<td>1012.34</td>
</tr>
<tr>
<td>2</td>
<td>222.22</td>
<td>444.44</td>
<td>790.12</td>
</tr>
<tr>
<td>3</td>
<td>222.22</td>
<td>666.67</td>
<td>567.89</td>
</tr>
<tr>
<td>4</td>
<td>222.22</td>
<td>888.89</td>
<td>345.67</td>
</tr>
<tr>
<td>5</td>
<td>222.22</td>
<td>1111.11</td>
<td>123.45</td>
</tr>
</tbody>
</table>

=============== SUM-OF-YEARS-DIGITS

<table>
<thead>
<tr>
<th>Year</th>
<th>Current Year Depreciation</th>
<th>Cumulative Depreciation</th>
<th>Book Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>1234.56</td>
</tr>
<tr>
<td>1</td>
<td>370.37</td>
<td>370.37</td>
<td>864.19</td>
</tr>
<tr>
<td>2</td>
<td>296.30</td>
<td>666.67</td>
<td>567.89</td>
</tr>
<tr>
<td>3</td>
<td>222.22</td>
<td>888.89</td>
<td>345.67</td>
</tr>
<tr>
<td>4</td>
<td>148.15</td>
<td>1037.04</td>
<td>197.52</td>
</tr>
<tr>
<td>5</td>
<td>74.07</td>
<td>1111.11</td>
<td>123.45</td>
</tr>
</tbody>
</table>

=============== DECLINING BALANCE with 200.00 PERCENT FACTOR

<table>
<thead>
<tr>
<th>Year</th>
<th>Current Year Depreciation</th>
<th>Cumulative Depreciation</th>
<th>Book Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>1234.56</td>
</tr>
<tr>
<td>1</td>
<td>493.82</td>
<td>493.82</td>
<td>740.74</td>
</tr>
<tr>
<td>2</td>
<td>296.29</td>
<td>790.12</td>
<td>444.44</td>
</tr>
<tr>
<td>3</td>
<td>177.78</td>
<td>967.90</td>
<td>266.66</td>
</tr>
<tr>
<td>4</td>
<td>106.67</td>
<td>1074.56</td>
<td>160.00</td>
</tr>
<tr>
<td>5</td>
<td>64.00</td>
<td>1138.56</td>
<td>96.00</td>
</tr>
</tbody>
</table>

Cannot take depreciation below book value of 123.45

=================================================================

Fig. 7-1. Tables generated by DEPRECIATION for an asset with a five-year life.
DEPRECIATION SCHEDULES

Name of the item to be depreciated: Macintosh Sooper-Doopers
Macintosh Sooper-Doopers are described as: Round w/center cutout
Acquisition cost: 1234.56
Useful life is 7 years.
Scrap value at end of 7 years: 100.00

===== STRAIGHT-LINE METHOD ======

<table>
<thead>
<tr>
<th>Year</th>
<th>Current Year Depreciation</th>
<th>Cumulative Depreciation</th>
<th>Book Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>1234.56</td>
</tr>
<tr>
<td>1</td>
<td>162.08</td>
<td>162.08</td>
<td>1072.48</td>
</tr>
<tr>
<td>2</td>
<td>162.08</td>
<td>324.16</td>
<td>910.40</td>
</tr>
<tr>
<td>3</td>
<td>162.08</td>
<td>486.24</td>
<td>748.32</td>
</tr>
<tr>
<td>4</td>
<td>162.08</td>
<td>648.32</td>
<td>586.24</td>
</tr>
<tr>
<td>5</td>
<td>162.08</td>
<td>810.40</td>
<td>424.16</td>
</tr>
<tr>
<td>6</td>
<td>162.08</td>
<td>972.48</td>
<td>262.08</td>
</tr>
<tr>
<td>7</td>
<td>162.08</td>
<td>1134.56</td>
<td>100.00</td>
</tr>
</tbody>
</table>

===== SUM-OF-YEARS-DIGITS ======

<table>
<thead>
<tr>
<th>Year</th>
<th>Current Year Depreciation</th>
<th>Cumulative Depreciation</th>
<th>Book Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>1234.56</td>
</tr>
<tr>
<td>1</td>
<td>283.64</td>
<td>283.64</td>
<td>950.92</td>
</tr>
<tr>
<td>2</td>
<td>243.12</td>
<td>526.76</td>
<td>707.80</td>
</tr>
<tr>
<td>3</td>
<td>202.60</td>
<td>729.36</td>
<td>505.20</td>
</tr>
<tr>
<td>4</td>
<td>162.08</td>
<td>891.44</td>
<td>343.12</td>
</tr>
<tr>
<td>5</td>
<td>121.56</td>
<td>1013.00</td>
<td>221.56</td>
</tr>
<tr>
<td>6</td>
<td>81.04</td>
<td>1094.04</td>
<td>140.52</td>
</tr>
<tr>
<td>7</td>
<td>40.52</td>
<td>1134.56</td>
<td>100.00</td>
</tr>
</tbody>
</table>

===== DECLINING BALANCE with 200.00 PERCENT FACTOR ======

<table>
<thead>
<tr>
<th>Year</th>
<th>Current Year Depreciation</th>
<th>Cumulative Depreciation</th>
<th>Book Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>1234.56</td>
</tr>
<tr>
<td>1</td>
<td>352.73</td>
<td>352.73</td>
<td>881.83</td>
</tr>
<tr>
<td>2</td>
<td>251.95</td>
<td>604.68</td>
<td>629.88</td>
</tr>
<tr>
<td>3</td>
<td>179.97</td>
<td>784.65</td>
<td>449.91</td>
</tr>
<tr>
<td>4</td>
<td>128.55</td>
<td>913.19</td>
<td>321.37</td>
</tr>
<tr>
<td>5</td>
<td>91.82</td>
<td>1005.01</td>
<td>229.55</td>
</tr>
<tr>
<td>6</td>
<td>65.58</td>
<td>1070.60</td>
<td>163.96</td>
</tr>
<tr>
<td>7</td>
<td>46.85</td>
<td>1117.44</td>
<td>117.12</td>
</tr>
</tbody>
</table>

Fig. 7-2. Tables generated by DEPRECIATION for an asset with a seven-year life.
SAMPLE DEPRECIATION PRINTOUTS

Figure 7-1 is a printout for the sample run. I have also included another sample run, Fig. 7-2, for an asset with a longer useful life period. Note the precision with which the program calculates the depreciation down to the declared scrap value level.

You'll certainly find the algorithms and the code interesting to learn from and work with. Introduce your own variations in the screen's presentations. I must repeat the suggestion that for an actual tax report to the IRS, it is prudent to seek the counsel of a specialist in taxation or a C.P.A. who keeps abreast of IRS regulations and can advise you about the many methods of depreciation available and suggest the one best suited to your particular situation.

CODE LISTING FOR DEPRECIATION

{
  +++++++++++++++++++++++++++++++++++++

  This program generates three types of depreciation tables for capital assets:

  (1) Straight Line (SL)
  (2) Sum-Of-Years-Digits (SOYD)
  (3) Declining Balance with Factor (DB)

  +++++++++++++++++++++++++++++++++++++

PROGRAM Depreciation ;
uses PasPrinter ; { This unit must be called at the start whenever }
{ a program is expected to use the printer. }

TYPE
  DeprecType = (SL,SOYD,DB) ;
  StringType = String[80] ;

VAR
  AcquisCost : Real ;
  AllDone : Boolean ;
  BookValue : Real ;
  Ch : Char ;
  CumDeprec : Real ;
  CurrentYr : Real ;
  DBFactor : Real ;
  I : Integer ;
  ItemDescr : StringType ;
  ItemName : StringType ;
  L : Integer ;
  ScrapValue : Real ;
  StraightLn : Real ;
  UsefulLife : Real ;
  YrsLeft : Real ;
PROCEDURE Sign_On ;  { Display the startup message. }
BEGIN { Sign_On }
  GotoXY(15,6) ;  { Send cursor to column 15, row 10. }
  FOR L := 1 TO 50 DO
  BEGIN
    Write('-') ;  { Display a line of 50 dashes. }
    GotoXY(22,8) ;  { Now, cursor to column 22, row 8. }
    Write('THREE-METHOD DEPRECIATION CALCULATOR') ;
    GotoXY(15,10) ;  { And, cursor to column 15, row 10. }
    FOR L := 1 TO 50 DO
    BEGIN
      Write('-') ;  { Again, display a line of dashes. }
    END ;
    GotoXY(15,14) ;  { Send cursor to column 15, row 14. }
    Write('*** Printout is automatic. Do turn on the printer ***') ;
    GotoXY(15,16) ;  { Cursor to column 15, row 16. }
    Write('Press Return to continue...') ;
    Read(Ch) ;
    ClearScreen ;  { ...and, clear the screen. }
  END ;  { of Sign_On }

PROCEDURE Get_The_Info ;  { Get the data via the keyboard. }
BEGIN { Get_The_Info }
  GotoXY(1,5) ;
  Write('Name the item to be depreciated: ') ;
  ReadLn(ItemName) ;
  WriteLn ;
  Write('Describe the ',ItemName,': ') ;
  ReadLn(ItemDescr) ;
  WriteLn ;
  Write('Give the acquisition cost of ',ItemName,': ') ;
  ReadLn(AcquisCost) ;
  WriteLn ;
  Write('Enter the useful life in years: ') ;
  ReadLn(UseFullLife) ;
  WriteLn ;
  Write('Type the scrap value at the end of ',
       UseFullLife:2:0,' years: ') ;
ReadLn(ScrapValue);

WriteLn;
Write('Factor (%) for Declining Balance calculations is: ');
ReadLn(DBFactor);
END;  { of Get_The_Info }

PROCEDURE Print_Method_Headers(VAR Printer:Text;
WhatKind:DeprecType);
BEGIN  { Print_Method_Headers }
WriteLn(Printer);
CASE (WhatKind) OF  { Headers for the three methods. }
  SL : WriteLn(Printer,'===========> STRAIGHT-LINE METHOD');
  SOYD : WriteLn(Printer,'===========> SUM-OF-YEARS-DIGITS');
  DB : WriteLn(Printer,'===========> DECLINING BALANCE with ',
     DBFactor:5:2,' PERCENT FACTOR');
END;  { of CASE }
WriteLn(Printer);
WriteLn('Current Year Cumulative Book');
WriteLn('Year Depreciation Depreciation Value');
WriteLn('--------------------------------------------------------');
END;  { of Print_Method_Headers }

PROCEDURE Display_Method_Headers(VAR Output:Text; WhatKind:DeprecType);
BEGIN  { Display_Method_Headers }
WriteLn;
CASE (WhatKind) OF  { Headers for the three methods. }
  SL : WriteLn('===========> STRAIGHT-LINE METHOD');
  SOYD : WriteLn('===========> SUM-OF-YEARS-DIGITS');
  DB : WriteLn('===========> DECLINING BALANCE with ',
     DBFactor:5:2,' PERCENT FACTOR');
END;  { of CASE }
WriteLn;
WriteLn('Current Year Cumulative Book');
WriteLn('Year Depreciation Depreciation Value');
WriteLn('--------------------------------------------------------');
END;  { of Display_Method_Headers }
PROCEDURE Print_A_Line ((VAR Printer:Text) );

BEGIN  { Print_A_Line }
   FOR L := 1 TO 65 DO
      BEGIN
         Write(Printer,'=')
      END ;
      WriteLn(Printer) ;
   END ;  { of Print_A_Line }

PROCEDURE Display_A_Line ;

BEGIN  { Display_A_Line }
   FOR L:= 1 TO 65 DO
      BEGIN
         Write('=')
      END ;
      WriteLn ;
   END ;  { Display_A_Line }

PROCEDURE Print_Main_Headings ;

BEGIN  { Print_Main_Headings }
   Print_A_Line ;  { Print a border before the description. }
   WriteLn(Printer) ;
   Write(Printer,'') ;
   WriteLn(Printer,'DEPRECIATION SCHEDULES') ;
   WriteLn(Printer) ;
   Print_A_Line ;
   WriteLn(Printer) ;  { Print the description. }
   WriteLn(Printer,'Name of the item to be depreciated: ',ItemName) ;
   WriteLn(Printer,ItemName,' are described as: ',ItemDescr) ;
   WriteLn(Printer,'Acquisition cost: ',AcquisCost:6:2) ;
   WriteLn(Printer,'Useful life is ',UsefulLife:2:0,' years.') ;
   WriteLn(Printer,'Scrap value at end of ',UsefulLife:2:0,' years: ',ScrapValue:6:2) ;
   WriteLn(Printer) ;
   Print_A_Line ;  { Print a border after the description. }
   ClearScreen ;
   Display_A_Line ;
   WriteLn ;
   Write('') ;
   WriteLn('DEPRECIATION SCHEDULES') ;
   WriteLn ;
   Display_A_Line ;
WriteLn;
WriteLn('Name of the item to be depreciated: ',ItemName);
WriteLn(ItemName,' are described as: ',ItemDescr);
WriteLn('Acquisition cost: ',AcquisCost:6:2);
WriteLn('Useful life is ',UsefulLife:2:0,' years.');
WriteLn('Scrap value at end of ',UsefulLife:2:0,' years: ',ScrapValue:6:2);
WriteLn;
Display_A_Line;
WriteLn;
END ; { of Print_Main_Headings }

PROCEDURE Write_The_Values(VAR Printer:Text;
YearNum:Real;
CurYr:Real;
Cumul:Real;
Book:Real);
BEGIN { Write_The_Values }
WriteLn(Printer,YearNum:2:0,' ',CurYr:10:2,
' ',Cumul:10:2,' ',Book:10:2);
END ; { of Write_The_Values }

BEGIN { Depreciation }
Sign_On;
Get_The_Info;
Print_Main_Headings;
Print_Method_Headers(Printer,SL);
Display_Method_Headers(Output,SL);

CumDeprec := 0;
Write_The_Values(Printer,0,0,0,AcquisCost);
Write_The_Values(Output,0,0,0,AcquisCost);

FOR I := 1 TO TRUNC(UsefulLife) DO
BEGIN
  CurrentYr := (AcquisCost - ScrapValue) / UsefulLife;
  CumDeprec := CumDeprec + CurrentYr;
  BookValue := AcquisCost - CumDeprec;
  StraightLn := (UsefulLife * UsefulLife + UsefulLife) / 2.0;
  Write_The_Values(Output,I,CurrentYr,CumDeprec,BookValue);
  Write_The_Values(Printer,I,CurrentYr,CumDeprec,BookValue);
END;

Print_Method_Headers(Printer,SOYD);
Display_Method_Headers(Output,SOYD);

CumDeprec := 0;
FOR $I := 1$ TO TRUNC($UsefulLife$) DO

BEGIN
  $YrsLeft := Usefullife - I + 1;$
  $CurrentYr := YrsLeft / StraightLn * (AcquisCost - ScrapValue);$;
  $CumDeprec := CumDeprec + CurrentYr;$
  $BookValue := AcquisCost - CumDeprec;$
  Write_The_Values(Output,$I$,CurrentYr,CumDeprec,BookValue);
  Write_The_Values(Printer,$I$,CurrentYr,CumDeprec,BookValue);

END;

WriteLn;
Print_Method_Headers(Printer,DB);
Display_Method_Headers(Output,db);

CumDeprec := 0;
Write_The_Values(Output,0,0,0,AcquisCost);
Write_The_Values(Printer,0,0,0,AcquisCost);
$DBFactor := (DBFactor / 100.0) / Usefullife;$
$CurrentYr := AcquisCost * DBFactor;$
$I := 1;$
AllDone := False;

REPEAT
  $YrsLeft := Usefullife - I + 1;$
  $CumDeprec := CumDeprec + CurrentYr;$
  $BookValue := AcquisCost - CumDeprec;$
  Write_The_Values(Output,$I$,CurrentYr,CumDeprec,BookValue);
  Write_The_Values(Printer,$I$,CurrentYr,CumDeprec,BookValue);
  $CurrentYr := BookValue * DBFactor;$
  $I := I + 1;$

  IF (BookValue < ScrapValue) THEN
    (* Display and print a special note re the SOYD method. *)
    BEGIN
      AllDone := True;
      WriteLn;
      WriteLn(Printer);
      WriteLn('Cannot take depreciation below book value of ',
      ScrapValue:6:2);
      WriteLn(Printer,'Cannot take depreciation below book value of ',
      ScrapValue:6:2);
    END;

  IF ($I > TRUNC(Usefullife)) THEN
    AllDone := True
UNTIL (AllDone) ;
WriteLn(Printer) ;
Print_A_Line ;   { The border at the end of the printout. }
WriteLn ;
WriteLn ;

   { Display a message and sound the Macintosh's beep. }
WriteLn('*** ALL DONE ***',Chr(7)) ;
WriteLn ;
Write('--- Press Return to complete the program --- ') ;
Read(Ch) ;

   { Eject the paper from the printer to the top of the form. }
WriteLn(Printer,Chr(12)) ;
END.    { of Depreciation }
Chapter 8

COUNTWORDS, COUNTSTROKES & COUNTCHARACTERS: Counting Programs

The code for a program that counts the number of words in a text file provides an excellent exercise in the use of a programming language. At the same time, the program is quite useful to people who must write a stipulated amount of copy—a certain number of words—to fit a specific allocation of space. The "words" may be for a report limited to a stated maximum number of words, a column written to fit a predetermined amount of space in the company's or club's newsletter, or whatever.

Code that counts words is often written as a programmer's exercise. Such programs can be found in many programming languages and in program-teaching texts. To enable you to learn while generating a utility with value, this chapter gives you a word-counting program, COUNTWORDS.

How many times have you run into situations like the following? The boss says to you, "Write a brief report...oh, say, about 250 words, summarizing the situation as you see it...in your own words."

The advertising agency needs a paragraph of copy, "50 succinct words" that describe the high-technology product your department has just designed and marketing wants to advertise. An editor wants the public relations department to give him a 300-word update.

The tech writer has to fill two and a half pages, estimated at about 625 words of double-spaced text, in order to balance the documentation.

The sales manager insists that all weekly reports, excluding data, be held to 300 words maximum. Perhaps he also insists that the reports be no shorter than 200 words, to avoid the cryptic styles that are often adopted by those who hate to write.
Or, the editor of your professional association’s newsletter has asked you to write a short piece: “Oh, about 225 words; half a column.”

The old-fashioned time-consuming grunt work—write, stop, do a printout, manually count the words, redo, add words, cut words—can be put forever behind you with COUNTWORDS. With it, the problem of knowing how much you have written is simplified. And, because you have rid yourself of the onerous chore of manually counting the words, your style shows a noticeable improvement. Whatever the requirement, if your task includes counting words, COUNTWORDS can do it for you, and you don’t even have to do a hard-copy printout!

**HOW DO YOU COUNT WORDS?**

Before I review the specific source code for the programs, I first ask the question, how do you count words? Do you really count words when you scan the page with point of your pencil? No—you are really counting the space between words.

As you know, every written word is preceded by a space. Whether there is one space or many spaces (such as when you hit the TAB key and enter a multiple number of sequential spaces), you count the separation between individual words as a quantity of one. This is the key to the fundamentals of the scheme for writing a word counting program in any computer language.

Some languages are better suited to running certain routines and procedures than others. BASIC is well suited to calling and reading DATA statements. Pascal is not. But, Pascal is better suited than BASIC to reading, looking into, examining, analyzing, and reporting on a text file that is external to the program and that is identified by a unique name. Turbo Pascal is exceptionally well suited to the purpose.

The source code is heavily commented so that others who read it can comprehend its structure and process. Equally important, you will be able to go back to it any time and not wonder “Why in the name of inspiration did I write the code that way? What does this particular command mean or that function really do?”

Actually, it is not necessary to use comment delimiters at the start and end of each physical line that contains comments. One at the start and one at the end of the individual comment—provided no source or program code is included—are all that are required.

Remember, COUNTWORDS is actually doing the counting task the same way we count words. It counts spaces and tabs, determining that a space or tab followed by a character marks the start of a word; thus, the word count is incremented by one for each occurrence. When the program can find no more word starts in the text being examined, the End Of File (EOF, or Eof) is reached and the WHILE (not EOF(f)) DO loop is exited. The total word-count, which is the final value for COUNT in the program’s code, is displayed on the screen. The program sounds a beep, closes the file, and the program is at its end.

For additional practice in using Turbo Pascal for counting, I’ve included two additional utilities, COUNTSTROKES and COUNTCHARS. The former counts the number of keystrokes used in entering new text through the keyboard. The latter counts the number of printable characters contained in an existing text file.
This is a word counting program. It counts a sequence of characters contained in a text file. Spaces and Tabs indicate the end and start of words. After the text has been examined by this program, the total COUNT of WORDS found in the text is shown on the screen. Thus, the user is able to determine whether or not a required wordcount has been achieved in the text.

PROGRAM CountWords ; 

VAR Ch,Space,Tab,Beep : Char ;
Count : Integer ;
F : Text ;
FileName : STRING [63] ; ( The maximum filename length is 63. )
IOerr : Boolean ;
WordFound : Boolean ;

BEGIN ( CountWords. )
ClearScreen ;
{ Disable automatic error checking. }

REPEAT
GotoXY(5,6) ; 
{ Send the cursor to col 5, row 6. }
( Display the opening instruction. )
Write('Enter the name of the text file to be counted: ') ;

{ Wait for the name of the text file to be entered at the keyboard; then respond. }
ReadLn(Filename) ;
Reset(F,FileName) ;
IOerr := (IOresult <> 0) ;

IF IOerr THEN
BEGIN
GotoXY(5,9) ; 
{ Start the next line at col 5, row 9. }
WriteLn('OOPS! I CAN'T FIND: ',Filename) ;
Write(Chr(7)) ; 
{ Sound the Macintosh's beep. }
GotoXY(5,12) ;
Write('Press Return to continue...or..."Q" to quit: ') ;
Read(Ch) ;

IF Ch IN ['Q','q'] THEN
EXIT ;
{ An escape hatch! }
ClearScreen;
END

UNTIL NOT IOerr;
{SI+}

Space := ' ';
Tab := (Chr(9));
Count := 0;
WordFound := False;
Beep := (Chr(7));

{ Set up a DO loop to continue until }
{ the End Of Filename (EOF) is Read. }
WHILE ( NOT Eof(F)) DO
BEGIN

WHILE ( NOT Eoln(F)) DO
BEGIN

{ Read the file, character by character }
{ and test for TRUE/FALSE, Spaces, and }
{ Tabs with the "IF...THEN" statements. }
Read(F,Ch);

IF (Ch <> Space) AND (Ch <> Tab) THEN
WordFound := True;

IF (WordFound = True) AND ((Ch = Space) OR (Ch = Tab)) THEN
BEGIN
WordFound := False;
Count := Count + 1
END;

END;

IF (WordFound = True) THEN
BEGIN
WordFound := False;
Count := Count + 1
END;

ReadLn(F)
END;

{ Now that the text file has been }
{ Read and the DO loops have been }
{ completed, skip a line at the }
{ screen and display the results }
{ of the Count. Insert the name }
ClearScreen;
GotoXY(5,6);
Write('THE TOTAL NUMBER OF WORDS IN: ');
Write(FileName);  { Display the name of the file. }
Write(' = ') ;
WriteLn(Count);  { Display the word count in the file. }

Write(Beep);    { Sound the Macintosh's beep. }
GotoXY(5,9);
Write('Press Return to quit...');
Read(Ch);

Close(F);    { Close the text file we had been counting. }
END. { of CountWords }

CODE LISTING FOR COUNTSTROKES

{ CountStrokes counts strokes made at the keyboard.  
  It is useful in a timed test for speed. }

PROGRAM Count_Strokes ;

VAR
  Ch : Char;
  Character : Char;
  Stroke_Count : Integer;

BEGIN
  Stroke_Count := -1;  { -1 compensates for the backslash }
  GotoXY(1,2);    { character typed at the end. }
  writeln('----------------------------------------');
  writeln('This program counts key strokes.');
  writeln;
  writeln('Start by typing the text to be counted.');
  writeln('... and ...');
  writeln('The exact moment you are finished....');
  writeln('Type a backslash (\) to end the session.');
  writeln('----------------------------------------');
  writeln;
  writeln('>>>
START NOW AT THE BLINKING CURSOR<<<<');
  writeln;
  writeln('----------------------------------------');
  writeln;

69
REPEAT
    Read(Character);
    Stroke_Count := Stroke_Count +1;
UNTIL (Character) IN ['\'];
writeln;
writeln;
writeln('THE TOTAL NUMBER OF KEystrokes IS: ',Stroke_Count);
writeln;
writeln;
write('Press Return to exit the program. ');
readln(Ch);
END.

CODE LISTING FOR COUNTCHARS

(*
  CountChars counts characters in a TEXT file when the fileName is entered in response to the first request.
*)

PROGRAM CountChars ;

VAR
  ch   : char;
  line : integer;
  space : char;
  tab  : char;
  count : integer;
  f    : text;
  filename : string[63];

BEGIN
  Writeln;
  BEGIN
    FOR Line := 1 TO 75 DO
      Write('-') ;
    END;

  Writeln;
  Write('Enter the name of the FILE to be counted: ');
  Readln(filename);
  Reset(f,filename);

  space := ' ';
  tab  := chr(9);
  count := 0;

  REPEAT
    Read(Character);
    Stroke_Count := Stroke_Count +1;
    UNTIL (Character) IN ['\'];
  writeln;
  writeln;
  writeln('THE TOTAL NUMBER OF KEystrokes IS: ',Stroke_Count);
  writeln;
  writeln;
  write('Press Return to exit the program. ');
  readln(Ch);
END.
WHILE (NOT Eof(f)) DO
BEGIN
    WHILE (NOT Eoln(f)) DO
        BEGIN
            Read(f,ch);
            IF (ch <> space) AND (ch <> tab) THEN
                count := count +1
            END;
            ReadLn(f)
        END;
    BEGIN
        FOR Line := 1 TO 75 DO
            Write(' - ') ;
            ( A borderline display. )
        END ;
    WriteLn ;
    WriteLn('The total character count of ',filename,' is: ',count);
BEGIN
    FOR Line := 1 to 75 DO
        Write(' - ') ;
        ( A borderline display. )
    END ;
    WriteLn ;
    WriteLn ;
    Write('Press Return to end the program. ') ;
    Read(ch) ;
END.
Chapter 9

INVESTVAL: Evaluating Investments

You made an investment a while back. What’s its effective value today? The program INVESTVAL asks for the dollar value of an investment. It goes on to ask how long ago the investment was made, and what it is worth today. The program computes the amount of change in dollars gained or lost, and displays the equivalent or effective change as a percentage of the investment. While the primary objective of this program is to provide more Turbo Pascal source code for learning and practice, its secondary objective is to give the user a rapid way of comparing investments.

For instance, if the program calculates that my investment has increased 100% during the five years I’ve owned or held it, how does it compare with other investment opportunities that were available to me or that may still be available to me for the future? INVESTVAL can help me make a decision for the future that, on the basis of historic fact, is possibly the best one.

Whether you are an active investor, trader, speculator, or a “paper player” who enjoys the risk-free challenge of watching the dynamics of the market, you have many decisions to make. Should you hold? Buy? Sell? There are so many good-looking buys that it can be difficult to decide which way to go. How about the 200 shares of common stock you bought 10 years ago at $19 a share? You held it when it dropped to $16. You held it when it split and climbed rapidly to $154. You now hold 498 shares. But the bottom has fallen out of the market, and the investment’s peak of $76,692 has dropped to $45,318. What should you do?

One thing you can do is set a bottom dollar for the stock, at which point you will trade, selling all or some of the shares. To determine the bottom you might pick a reference point, say the interest rate currently being paid on savings accounts, which might
be in the order of 5% a year. As long as your stock is doing better than this, you de-
cide, you will hold onto the stock. You will periodically check the market price and
equate your holdings in this particular stock to the interest-earned value of your origi-
nal investment, which was for 200 shares multiplied by $19 per share plus fees and
taxes; roughly $3,850 is what you put into it 10 years ago. Not bad, but, on the other
hand, you don't want to just sit on it and lose out if the danger signals get very strong.

INVESTVAL was designed specifically to solve this problem. While it can't make
the decisions for you, it can give you a rapid statistical analysis that will help you decide.

A SAMPLE RUN OF INVESTVAL

Compile and run INVESTVAL. Try it out on a hypothetical 10-year-old $3,850
investment. Let's see what its growth has been, when or how it might compare with
other investment opportunities when effective interest is the measure of growth. The
program displays all queries and your responses on the screen. A printout is not essen-
tial and, therefore, has not been incorporated into the code. Here is the dialog you have
with the screen, as ordered by the Get__The__Facts procedure. In this illustration, the
screen's queries are shown on the left. Your responses are shown underlined:

Enter the ORIGINAL amount of the investment: 3850
Enter the MARKET value of the investment now: 45318
Enter the NUMBER OF YEARS it has been held: 10

When you complete the third entry and press the Return key, the screen clears
briefly while the calculations are being performed. The Show__Results procedure is
called and executed and, for our example given above, the screen shows the results:

=================================================================
The investment's original value has INCREASED!
During the 10 years it has been held
it has increased at the annual rate of: 27.96 percent
The original investment: $3850.00
The current market value: $45318.00
The total change in value: $41468.00
=================================================================

And then, below the above information, the screen displays:

Do you want to do another set of data? (Y/N):

Out of curiosity, let's do the same type of run with the stock's market value at
its peak when you had held it for nine and a half years:

Enter the original amount of the investment: 3850
Enter the market value of the investment now: 76692
Enter the number of years it has been held: 9.5
=================================================================
The investment's original value has INCREASED!
During the 9.5 years it has been held
it has increased at the annual rate of: 37.01 percent
The original investment: $3850.00
The current market value: $76692.00
The total change in value: $72842.00

As an interest earner it's a big winner; you knew that, but now you know exactly
how big it is.
As an additional demonstration of the program's action, let's reverse the situation.
It was purchased at $45,318, and the value is now $3,850. It's a financial disaster, all
right! Let's see how the program handles it:

Enter the original amount of the investment: 45318
Enter the market value of the investment now: 3850
Enter the number of years it has been held: 9.5

The investment's original value has DECREASED!
During the 9.5 years it has been held
it has decreased at the annual rate of: -21.85 percent
The original investment: $45318.00
The current market value: $3850.00
The total change in value: -$41468.00

CODE LISTING FOR INVESTVAL

{ Investment values can be compared with this program. }

PROGRAM InvestVal;

TYPE
  StringType = String[80];

VAR
  Ch : Char;
  ChangeValue, Interest : Real;
  L : Integer;
  NumYears, OriginalValue, ValueNow : Real;

PROCEDURE SignOn;
  ( Display the startup message on the screen. )
BEGIN { SignOn }
  GotoXY(15,8); { Go to column 15, row 8. }
  WriteLn('$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$');
  GotoXY(24,10);
  WriteLn('INVESTVAL -- INVESTMENT ANALYZER');
  GotoXY(15,12);
PROCEDURE Get_The_Facts;  { Ask for the data. }  
BEGIN { Get_The_Facts }

GotoXY(15,15);  
Write('Enter the ORIGINAL amount of the investment: $');  
ReadLn(OriginalValue);  

GotoXY(15,17);  
Write('Enter the MARKET value of the investment now: $');  
ReadLn(ValueNow);  

GotoXY(15,19);  
Write('Enter the NUMBER OF YEARS it has been held: ');  
ReadLn(NumYears);  

END;  { of Get_The_Facts }

PROCEDURE MakeBorder;  { Generate a graphic border. }
BEGIN { MakeBorder }

Write(' ');  { Insert a space before the next line. }
FOR L := 1 TO 60 DO  { Display it 60 times in a row. }
BEGIN
  Write(CHR(197));
END;
END;  { of MakeBorder }

PROCEDURE Show_Results;  { Display the results. }  
BEGIN { Show_Results }

MakeBorder;
WriteLn;
WriteLn;

IF (ChangeValue >= 0) THEN
BEGIN
  WriteLn('The investment''s original value has INCREASED!');
  WriteLn;
  WriteLn('During the ',NumYears:4:2,' years it has been held');
  Write(' it has increased at the annual rate of: ');  
  WriteLn(Interest:5:2,' percent');
END
If it hasn’t increased in value, it has decreased.

ELSE
BEGIN
  WriteLn('The investment’s original value has DECREASED!');
  WriteLn;
  WriteLn('During the ', NumYears:4:2, ' years it has been held');
  Write('it has decreased at the annual rate of: ');
  WriteLn(Interest:5:2,' percent');
END;

WriteLn;
WriteLn('The original investment: $',OriginalValue:10:2);
WriteLn('The current market value: $',ValueNow:10:2);
WriteLn('The total change in value: $',ChangeValue:10:2);
WriteLn;
MakeBorder;
WriteLn;
WriteLn;
WriteLn;
END; { Show_Results }

BEGIN { InvestVal }
REPEAT
  ClearScreen;
  SignOn;
  Get_The_Facts; { Call PROCEDURE Get_The_Facts. }
  { The calculations are next. }
  ChangeValue := ValueNow - OriginalValue;
  Interest := Exp(Ln(ValueNow / OriginalValue) * 
    (1.0 / NumYears)) - 1;
  Interest := Interest * 100.0;
  ClearScreen;
  Show_Results; { Call the Show_Results PROCEDURE. }
  Write('Do you want to do another set of data? (Y/N): ');
  Read(Ch); { Get the keyboard character; do not echo it. }
  { End when any character but 'Y' or 'y' is typed. }
UNTIL (Ch <> 'Y') AND (Ch <> 'y')
GotoXY(15, 22); { Go to column 15, row 22. }
  { Change the signoff message on the next line to suit yourself. }
  Write('Thank you. We hope the news was good!');
  ReadLn(Ch);
END. { InvestVal }
Chapter 10

LOANPAYMENTS: Mortgage and Loan Tables

Here’s a comprehensive mathematical-table generator that takes a few snatches of input data and produces a large amount of output data. Forward declarations of some of the procedures are used in the design of LOANPAYMENTS. They provide an opportunity to learn about and practice with the flowpaths of Pascal Procedures. The use of Forward declarations can eliminate the need to carefully plan the relative positions of Procedures in the program’s structure. Forward declarations are deliberately used in several of the programs in this book. Try to rewrite the code to eliminate the forward declarations in this program and in others in this book by reordering the relative positions of the Procedures.

This program has genuine applications in business, professional, and private affairs. There does come a time when additional cash infusions must be obtained. This holds true for the smallest and the largest organizations, whether they are start ups or veterans, sole proprietorships, partnerships, or publicly or closely held corporations.

In business and financial operations, as an aid to solving cash-flow problems, money can be borrowed for short terms, typically with due-dates of 30, 90, 180, or 360 days for repayment of the loan with interest earned by the lender. Capital equipment purchases may require loans extending into months or years.

As a borrower or a lender, your questions that need answers are usually, “How much will the money cost (or earn) for the term of the loan?” “What is the amount of the periodic repayments?” “How much of each payment goes toward reducing the principal amount of the loan? How much of the amount is interest?” And, “What is the total interest I will pay (or earn) by the time the loan is fully paid off?”

The LOANPAYMENTS program is designed to answer all these questions rapidly
and precisely based on the answers given to the program's questions.

The compiled program begins with a call to the SignOn procedure, which displays the program's banner heading and a message to turn on the printer.

The GetInfo Procedure is called and asks for data. The responses through the keyboard are underlined at the right:

Enter the principal amount of the mortgage/loan: 5000
And the interest rate in percent: 9.75
What is the term of the mortgage/loan in months: 6

The remainder of the procedures do the calculations and cause the table of data to be printed. The two sample runs, Fig. 10-1 and Fig. 10-2 show sample runs based on the brief amount of data entered in the above example. Figure 10-1 is for a 15-month term. Figure 10-2 uses the same dollars and percentage for a 36-month term.

**SAMPLE RUNS OF LOAN PAYMENTS**

<table>
<thead>
<tr>
<th>Month</th>
<th>Monthly Payment</th>
<th>Interest Paid</th>
<th>Principal Reduced</th>
<th>Principal Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>5000.00</td>
</tr>
<tr>
<td>1</td>
<td>355.41</td>
<td>40.62</td>
<td>314.78</td>
<td>4685.22</td>
</tr>
<tr>
<td>2</td>
<td>355.41</td>
<td>38.07</td>
<td>317.34</td>
<td>4367.87</td>
</tr>
<tr>
<td>3</td>
<td>355.41</td>
<td>35.49</td>
<td>319.92</td>
<td>4047.95</td>
</tr>
<tr>
<td>4</td>
<td>355.41</td>
<td>32.89</td>
<td>322.52</td>
<td>3725.43</td>
</tr>
<tr>
<td>5</td>
<td>355.41</td>
<td>30.27</td>
<td>325.14</td>
<td>3400.29</td>
</tr>
<tr>
<td>6</td>
<td>355.41</td>
<td>27.63</td>
<td>327.78</td>
<td>3072.51</td>
</tr>
<tr>
<td>7</td>
<td>355.41</td>
<td>24.96</td>
<td>330.44</td>
<td>2742.07</td>
</tr>
<tr>
<td>8</td>
<td>355.41</td>
<td>22.28</td>
<td>333.13</td>
<td>2408.94</td>
</tr>
<tr>
<td>9</td>
<td>355.41</td>
<td>19.57</td>
<td>335.84</td>
<td>2073.10</td>
</tr>
<tr>
<td>10</td>
<td>355.41</td>
<td>16.84</td>
<td>338.57</td>
<td>1734.54</td>
</tr>
<tr>
<td>11</td>
<td>355.41</td>
<td>14.09</td>
<td>341.32</td>
<td>1393.22</td>
</tr>
<tr>
<td>12</td>
<td>355.41</td>
<td>11.32</td>
<td>344.09</td>
<td>1049.13</td>
</tr>
<tr>
<td>13</td>
<td>355.41</td>
<td>8.52</td>
<td>346.88</td>
<td>702.25</td>
</tr>
<tr>
<td>14</td>
<td>355.41</td>
<td>5.71</td>
<td>349.70</td>
<td>352.54</td>
</tr>
<tr>
<td>15</td>
<td>355.41</td>
<td>2.86</td>
<td>352.54</td>
<td>-0.00</td>
</tr>
</tbody>
</table>

Total Interest: 331.13
Interest Rate (%): 9.75

---

Fig. 10-1. Payment table generated for a 15-month term loan.
### MORTGAGE/LOAN PAYMENT SCHEDULE

<table>
<thead>
<tr>
<th>Month</th>
<th>Monthly Payment</th>
<th>Interest Paid</th>
<th>Principal Reduced</th>
<th>Principal Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>15000.00</td>
</tr>
<tr>
<td>1</td>
<td>492.86</td>
<td>140.62</td>
<td>352.23</td>
<td>14647.77</td>
</tr>
<tr>
<td>2</td>
<td>492.86</td>
<td>137.32</td>
<td>355.54</td>
<td>14292.23</td>
</tr>
<tr>
<td>3</td>
<td>492.86</td>
<td>133.99</td>
<td>358.87</td>
<td>13933.36</td>
</tr>
<tr>
<td>4</td>
<td>492.86</td>
<td>130.63</td>
<td>362.23</td>
<td>13571.13</td>
</tr>
<tr>
<td>5</td>
<td>492.86</td>
<td>127.23</td>
<td>365.63</td>
<td>13205.50</td>
</tr>
<tr>
<td>6</td>
<td>492.86</td>
<td>123.80</td>
<td>369.06</td>
<td>12836.44</td>
</tr>
<tr>
<td>7</td>
<td>492.86</td>
<td>120.34</td>
<td>372.52</td>
<td>12463.93</td>
</tr>
<tr>
<td>8</td>
<td>492.86</td>
<td>116.85</td>
<td>376.01</td>
<td>12087.92</td>
</tr>
<tr>
<td>9</td>
<td>492.86</td>
<td>113.32</td>
<td>379.53</td>
<td>11708.38</td>
</tr>
<tr>
<td>10</td>
<td>492.86</td>
<td>109.77</td>
<td>383.09</td>
<td>11325.29</td>
</tr>
<tr>
<td>11</td>
<td>492.86</td>
<td>106.17</td>
<td>386.68</td>
<td>10938.61</td>
</tr>
<tr>
<td>12</td>
<td>492.86</td>
<td>102.55</td>
<td>390.31</td>
<td>10548.30</td>
</tr>
<tr>
<td>13</td>
<td>492.86</td>
<td>98.89</td>
<td>393.97</td>
<td>10154.33</td>
</tr>
<tr>
<td>14</td>
<td>492.86</td>
<td>95.20</td>
<td>397.66</td>
<td>9756.67</td>
</tr>
<tr>
<td>15</td>
<td>492.86</td>
<td>91.47</td>
<td>401.39</td>
<td>9355.28</td>
</tr>
<tr>
<td>16</td>
<td>492.86</td>
<td>87.71</td>
<td>405.15</td>
<td>8950.13</td>
</tr>
<tr>
<td>17</td>
<td>492.86</td>
<td>83.91</td>
<td>408.95</td>
<td>8541.17</td>
</tr>
<tr>
<td>18</td>
<td>492.86</td>
<td>80.07</td>
<td>412.78</td>
<td>8128.39</td>
</tr>
<tr>
<td>19</td>
<td>492.86</td>
<td>76.20</td>
<td>416.65</td>
<td>7711.73</td>
</tr>
<tr>
<td>20</td>
<td>492.86</td>
<td>72.30</td>
<td>420.56</td>
<td>7291.17</td>
</tr>
<tr>
<td>21</td>
<td>492.86</td>
<td>68.35</td>
<td>424.50</td>
<td>6866.67</td>
</tr>
<tr>
<td>22</td>
<td>492.86</td>
<td>64.38</td>
<td>428.48</td>
<td>6438.19</td>
</tr>
<tr>
<td>23</td>
<td>492.86</td>
<td>60.36</td>
<td>432.50</td>
<td>6005.69</td>
</tr>
<tr>
<td>24</td>
<td>492.86</td>
<td>56.30</td>
<td>436.56</td>
<td>5569.13</td>
</tr>
<tr>
<td>25</td>
<td>492.86</td>
<td>52.21</td>
<td>440.65</td>
<td>5128.48</td>
</tr>
<tr>
<td>26</td>
<td>492.86</td>
<td>48.08</td>
<td>444.78</td>
<td>4683.70</td>
</tr>
<tr>
<td>27</td>
<td>492.86</td>
<td>43.91</td>
<td>448.95</td>
<td>4234.76</td>
</tr>
<tr>
<td>28</td>
<td>492.86</td>
<td>39.70</td>
<td>453.16</td>
<td>3781.60</td>
</tr>
<tr>
<td>29</td>
<td>492.86</td>
<td>35.45</td>
<td>457.41</td>
<td>3324.19</td>
</tr>
<tr>
<td>30</td>
<td>492.86</td>
<td>31.16</td>
<td>461.69</td>
<td>2862.50</td>
</tr>
<tr>
<td>31</td>
<td>492.86</td>
<td>26.84</td>
<td>466.02</td>
<td>2396.48</td>
</tr>
<tr>
<td>32</td>
<td>492.86</td>
<td>22.47</td>
<td>470.39</td>
<td>1926.08</td>
</tr>
<tr>
<td>33</td>
<td>492.86</td>
<td>18.06</td>
<td>474.80</td>
<td>1451.28</td>
</tr>
<tr>
<td>34</td>
<td>492.86</td>
<td>13.61</td>
<td>479.25</td>
<td>972.03</td>
</tr>
<tr>
<td>35</td>
<td>492.86</td>
<td>9.11</td>
<td>483.75</td>
<td>488.28</td>
</tr>
<tr>
<td>36</td>
<td>492.86</td>
<td>4.58</td>
<td>488.28</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**Total Interest:** 2742.91  
**Interest Rate (%):** 11.25

Fig. 10-2. Payment table generated for a 36-month term loan.
CODE LISTING FOR LOANPAYMENTS

{ Mortgage and loan payment calculator. }

PROGRAM LoanPayments;
uses PasPrinter; { This program uses the printer. }

TYPE
  StringType = String[80];

VAR
  Ch : Char;
  F : Text;
  I : Integer;
  Interest : Real;
  IntrstPaid : Real;
  MonthlyPay : Real;
  Principal : Real;
  PrincPaid : Real;
  Temp : 
  Term : 
  Total : Real;

{ FORWARD declarations are listed in alphabetical order. }

PROCEDURE Border_Graphics(VAR F:Text);
  FORWARD;

PROCEDURE Do_The_Heading(VAR F:Text; IsStraightLine:Boolean);
  FORWARD;

PROCEDURE Format_Values(VAR F:Text;
  MonthNum,MonthPay,IntrPaid:Real;
  PrinReduced,PrinBal:Real);
  FORWARD;

PROCEDURE Summary(VAR F:Text);
  FORWARD;

PROCEDURE Sign_On; { Display a banner with the program's name. }
BEGIN { Sign_On }
  GotoXY(16,6); { Go to column 16, row 6 to start the display. }
  Write('$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$');
  GotoXY(24,8); { Now to column 24, row 8. }
  Write('MORTGAGE & LOAN PAYMENT SCHEDULE');

GotoXY(16,10);  { Column 16, row 10. } 
Write('$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$');

GotoXY(22,14);  { Column 22, row 14 advises about the printer. } 
Write('**** TURN ON THE PRINTER, PLEASE ****');

GotoXY(24,18);  { If everything's ready... } 
Write('Press the Return key when ready...'); 
Read(Ch);  { Read the keyboard character. } 
END;  { of Sign_On }

PROCEDURE Get_Info;  { Get the data. }
BEGIN  { Get_Info }
  ClearScreen;
  WriteLn;
  Write('Enter the principal amount of the mortgage/loan: '); 
  ReadLn(Principal);  { Read and store the keyboard entry. }
  WriteLn;
  Write('And the interest rate in percent: '); 
  ReadLn(Interest);
  Interest := Interest / 100;
  WriteLn;
  Write('What is the term of the mortgage/loan in months: '); 
  ReadLn(Term);
END;  { of Get_Info }

PROCEDURE Summary;  { Write a short summary. }
BEGIN  { Summary }
  WriteLn(F,',',Total:10:2);
  WriteLn(F,',',Interest*100.0:10:2);
  WriteLn(F);
  Border_Graphics(F);  { Call the Border_Graphics PROCEDURE. }
END;  { of Summary }

PROCEDURE Do_The_Heading;  { Display the headings. }
BEGIN  { Do_The_Heading }
  Border_Graphics(F);
  WriteLn(F,'MORTGAGE/LOAN PAYMENT SCHEDULE');
  Border_Graphics(F);
WriteLn(F);
WriteLn(F,
   'Month      Monthly      Interest     Principal     Principal');

{ ...and in the line immediately below the one above... }
WriteLn(F,
   'Payment      Paid      Reduced     Balance');

FOR I := 1 TO 60 DO
BEGIN
   Write(F,'-')  { Write a line of 60 hyphens. }
END;
WriteLn(F);
END;  { of Do_The_Heading }

PROCEDURE Border_Graphics;  { Write a border line... }  
BEGIN
{ Border_Graphics }
FOR I := 1 TO 60 DO
BEGIN
   Write(F,'=')  { ...of 60 equal signs. }
END;
WriteLn(F);
END;  { of Border_Graphics }

PROCEDURE Format_Values;  { Format and write values. }
BEGIN  { Format_Values }
   WriteLn(F,MonthNum:2:0,',
           MonthPay:10:2,',
           IntrPaid:10:2,',
           PrinReduced:10:2,',
           PrinBal:10:2);  
END;  { of Format_Values }

BEGIN  { LoanPayments...the main program }
   Sign_On;  { Call the Sign_On PROCEDURE. }
   Get_Info;  { Same for Get_Info. }

   ClearScreen;
   Do_The_Heading(Output,True);  { Output is the video display. }
   Do_The_Heading(Printer,True);  { Printer is the printer. }
{ Set the initial values. }
Temp := Exp(Term * Ln(1.0 + (Interest / 12.0))) ;
MonthlyPay := Principal * (Interest / 12.0) / (1.0 - (1.0 / Temp)) ;
Total := 0 ; { Set the counter to zero. }
Format.Values(Output,0,0,0,0,Principal) ;
Format.Values(Printer,0,0,0,0,Principal) ;

{ Do the calculations. }
FOR I := 1 TO Trunc(Term) DO
BEGIN
  IntrstPaid := Principal * Interest / 12.0 ;
  PrincPaid := MonthlyPay - IntrstPaid ;
  Principal := Principal - PrincPaid ;
  Total := Total + IntrstPaid ;
  Format.Values(Output,I,MonthlyPay,IntrstPaid,PrincPaid,Principal) ;
  Format.Values(Printer,I,MonthlyPay,IntrstPaid,PrincPaid,Principal) ;
END ;
Summary(Output) ; { Call Summary ... to the monitor. }
Summary(Printer) ; { Call Summary ... to the printer. }

WriteLn ; Write('Press the Return key to complete the program... ') ; Read(Ch) ;

{ Send a form feed to the printer to }
{ make certain the printer's buffer }
{ is emptied; and move the paper out }
{ of the printer with a form feed. }
WriteLn(Printer,Char(12)) ;
END. { of LoanPayments }
Chapter 11

NETWORTH:
Calculating Your Net Worth

Searching for a loan from a bank? If you are operating an enterprise as a sole proprietorship, a partnership, or a closely held corporation still in its start-up stages, the bank will most likely ask you to prepare a personal financial statement. If the loan is approved, it will probably be on the basis of your personal net worth rather than on the assets of the enterprise.

If you want to finance a major purchase that requires the use of credit, you usually have to prepare a detailed personal financial statement. If you should engage an estate planner, a tax attorney, or a C.P.A. to help you define, manage, and improve your financial affairs, you will have to prepare a personal financial statement before anything else can be done.

NETWORTH, the next program in Turbo Pascal, is designed to fit the needs of the majority of prospective lenders, creditors, and financial advisers. Its primary objective is to mathematically determine your net worth as an individual. As you may know, in a company's balance sheet total assets must always be equal to (in exact balance with) total liabilities. In the personal financial statement, which has the general appearance of a balance sheet, one expects there to be a difference between the two totals. That "difference" is your "financial net worth." The difference should be a positive value.

If your total liabilities (which are the debts you have, or the amount of money you owe) exceed your total assets (which are the things you own or which can be quickly converted to cash to pay off your debts), then your net worth in dollars is negative. As with a company's balance sheet, a personal financial statement that is periodically prepared clearly reveals trends in your material growth.
In preparing a company's balance sheet, there are two accepted formats: account and report. In the account format, the data are displayed side by side, assets on the left, liabilities on the right. The report format displays assets and liabilities one above the other. In preparing a personal financial statement, the report format is generally used. And this is exactly what I have done in designing the program NETWORTH.

**A TEST RUN OF NETWORTH**

NETWORTH is highly interactive and is set up to ensure that you correctly follow the process of completing the detailed input of data. The output is a neatly printed sheet that contains all the calculations and their results. Compile the code. When you call up the executable program, the printer must be turned on and the paper properly adjusted.

After the banner is displayed by the SignOn Procedure, the screen advises you:

*** TURN ON THE PRINTER ***

 *** Press the Return key when ready ***

When you press the Return key, the screen clears and a set of instructions is displayed, as in Fig. 11-1 (Shift-Command 4).

---

**PLEASE READ CAREFULLY ...**

Please enter 'amounts' when requested.
Do not enter commas. Do enter decimal points.
Respond 'Y' or 'N' when asked to confirm.

Don't be concerned about errors.
You will be able to correct them with 'N' when you are asked to confirm each entry.

**ASSETS**

(This section is for your assets....)
(Everything you own with cash value.)

Cash (Money On Hand):

---

Fig. 11-1. Simultaneously pressing the Shift, Command, and four keys at any time creates a printout of the video display, shown here for the instruction-screen for NETWORTH.
A series of queries that wait for responses from the user begins at the bottom of the display.

Please type your full name: CONFIDENTIAL
Enter the date of this statement (MM/DD/YY): 09/30/87

The heading, with the name and date you entered, is then printed.

The program continues as the Show_How procedure is called to display a set of instructions on the screen. (See the NetWorth code listing for the actual text.) The next procedure to be called is List_Assets. Individual queries are displayed for asset items. Each time you enter an amount in response to a query the screen displays another question:

Is the amount correct? (Y/N):

If you respond <Y> and press Return, the line with the amount you confirmed to be correct is sent to the printer. Because of the REPEAT . . . UNTIL loop in this procedure, if you respond <N>, the item is repeated and no action occurs at the printer until you respond <Y>, the amount is correct. Thus, you are able to correct any errors in data you have entered immediately. If the amount for any line is zero, just press the Return key or type the zero digit. In either case you will be prompted for confirmation before the computer accepts the amount and sends it to the printer.

After completing your responses to the queries concerning your assets, the screen displays and the printer prints the total assets in dollars.

The program continues with the List_Liabilities Procedure, displaying individual items and waiting for your entry before asking the same question each time:

Is the amount correct? (Y/N):

After completing your responses to the queries about your debts, the screen displays and the printer prints the total of your declared debts and the difference between your TOTAL ASSETS and your TOTAL DEBTS, which is your NET WORTH.

A sample of a complete printout of a run of NETWORTH is given in Fig. 11-2. The complete source code in Turbo Pascal is listed in Fig. 11-3.

SAMPLE PRINTOUT OF NETWORTH

=====================================
STATEMENT OF PERSONAL NET WORTH
FOR: CONFIDENTIAL
AS OF: 06/30/87
=====================================

Fig. 11-2. A sample run to calculate the net worth of "Confidential" generated this report.
ASSETS

- Cash (Money On Hand) 832.44
- Savings Accounts 2356.89
- Checking Accounts 1200.00
- CD's 0.00
- Money Mart 2500.00
- Stocks, Bonds 12575.00
- Other Securities 5000.00
- Notes Receivable 0.00
- Life Insurance Cash Val 25000.00
- Rebates/Refunds Owed You 175.84
- Vehicles 5695.00
- Real Estate 79500.00
- Annuities/Pensions 0.00
- Furniture/Appliances 7500.00
- Jewelry/Cameras/Tools 2500.00
- Valuable Collections 1000.00
- Other Assets 0.00

--------------------------
TOTAL ASSETS: 145835.17
--------------------------

LIABILITIES

- Contracts Outstanding 2976.88
- Promissory Notes Payable 3000.00
- Taxes Due and Unpaid 395.89
- Loans Outstanding 1000.00
- Mortgages (Princ. Amts) 55000.00
- Court-Ordered Payments 0.00
- Other Debts 2500.00

--------------------------
TOTAL LIABILITIES: 64872.77
--------------------------

======================================
NET WORTH: 80962.40
======================================

Fig. 11-2. Continued

CODE LISTING FOR NETWORTH

{ Calculate material net worth of an individual on }
{ the basis of physical assets and liabilities. }
{ This provides an example of string arrays. }
PROGRAM NetWorth;
uses PasPrinter;

TYPE  { List of labels to be used by the following arrays: }
  AssetType = (Cash,Savings,Checking,CD,MoneyMart,Stocks,
    Securities,Notes,LifeIns,Rebates,Vehicles,
    RealEstate,Annuities,Furniture,Jewelry,
    Collections,OtherAsset) ;
  LiabilType = (Contracts,PromisNotes,Taxes,Loans,
    Mortgages,Payments,Debts) ;
  StringType = String[80] ;

VAR
  Asset : ARRAY [AssetType] OF Real ;
  Asset_Name : ARRAY [AssetType] OF StringType ;
  Ch : Char ;
  Date : StringType ;
  L : Integer ;
  Liability : ARRAY [LiabilType] OF Real ;
  Liab_Name : ARRAY [LiabilType] OF StringType ;
  Name : StringType ;
  Total_Assets : Real ;
  Total_Liabilities : Real ;

PROCEDURE Initialize ;  { Assign String values to the arrays' labels. }
BEGIN ( Initialize )
  Asset_Name[Cash] := ' Cash (Money On Hand)' ;
  Asset_Name[Savings] := ' Savings Accounts' ;
  Asset_Name[Checking] := ' Checking Accounts' ;
  Asset_Name[CD] := ' CD's' ;
  Asset_Name[MoneyMart] := ' Money Mart' ;
  Asset_Name[Stocks] := ' Stocks, Bonds' ;
  Asset_Name[Securities] := ' Other Securities' ;
  Asset_Name[Notes] := ' Notes Receivable' ;
  Asset_Name[LifeIns] := ' Life Insurance Cash Val' ;
  Asset_Name[Rebates] := ' Rebates/Refunds Owed You' ;
  Asset_Name[Vehicles] := ' Vehicles' ;
  Asset_Name[RealEstate] := ' Real Estate' ;
  Asset_Name[Annuities] := ' Annuities/Pensions' ;
  Asset_Name[Furniture] := ' Furniture/Appliances' ;
  Asset_Name[Jewelry] := ' Jewelry/Cameras/Tools' ;
  Asset_Name[Collections] := ' Valuable Collections' ;
  Asset_Name[OtherAsset] := ' Other Assets' ;
  Liab_Name[Contracts] := ' Contracts Outstanding' ;
  Liab_Name[PromisNotes] := ' Promissory Notes Payable' ;
  Liab_Name[Taxes] := ' Taxes Due and Unpaid' ;
  Liab_Name[Loans] := ' Loans Outstanding' ;
Liab_Name[Mortgages] := 'Mortgages (Princ. Amts)';
Liab_Name[Payments] := 'Court-Ordered Payments';
Liab_Name[Debts] := 'Other Debts';
Total_Assets := 0.0; { Set the counter to zero. }
Total_Liabilities := 0.0; { Set the counter to zero. }

PROCEDURE Sign_On;
BEGIN  { Sign_On }
  GotoXY(12,10);
  FOR L := 1 TO 60 DO  { Do a task 30 times. }
    BEGIN  { Start the task: a loop. }
      Write(Char('*')) ;  { Display a line of asterisks. }
      END ;  { End the loop. }
  END ;  { of Initialize }

  GotoXY(17,12);
  Write('NETWORTH - STATEMENT OF PERSONAL FINANCIAL NET WORTH');

  GotoXY(12,14);
  FOR L := 1 TO 60 DO  { This is the same as the above. }
    BEGIN  { Start the task: a loop. }
      Write(Char('*')) ;  { Display a line of asterisks. }
      END ;  { End the loop. }
  END ;  { of Sign_On }

PROCEDURE Show_How;  { Display instructions for the user. }
BEGIN  { Show_How }
  ClearScreen;
  WriteLn;
  WriteLn('PLEASE READ CAREFULLY ...');  (* Instructions *)
  WriteLn('----------------------------------
  Please enter 'amounts'' when requested.'));
  WriteLn('Do not enter commas. Do enter decimal points.'));
  WriteLn('Respond 'Y'' or 'N'' when asked to confirm.'));
  WriteLn('Don''t be concerned about errors.'));
  WriteLn('You will be able to correct them with 'N'''));
  WriteLn('when you are asked to confirm each entry.'));
  END ;  { of Show_How }

PROCEDURE Broken_line(VAR OutFile:Text);  { Draw a "broken line." }
BEGIN  { Broken_line }
FOR L := 1 TO 38 DO  { Write the character 38 times. }
BEGIN
Write(OutFile,'-');
END;
WriteLn(OutFile);
END;  { of Broken_line }

PROCEDURE Double_Line(VAR OutFile:Text);  { Draw a "double line." }
BEGIN  { Double_Line }
FOR L := 1 TO 38 DO  { Write the character 38 times. }
BEGIN
Write(OutFile,'=');
END;
WriteLn(OutFile);
END;  { of Double_Line }

PROCEDURE List_Assets;  { Collect information about assets. }
VAR
AsstCtr : AssetType;

PROCEDURE Asset_Total(VAR OutFile:Text);
BEGIN  { Asset_Total }
Broken_line(OutFile);  { Call the PROCEDURE. }
WriteLn(OutFile,' TOTAL ASSETS: ':26,' ','
Total_Assets:10:2);
Broken_line(OutFile);  { Call it again. }
END;  { of Asset_Total }

BEGIN  { List_Assets }
WriteLn;
WriteLn('ASSETS');
WriteLn('------');
WriteLn(Printer,'ASSETS');
WriteLn(Printer,'------');
WriteLn;
WriteLn('This section is for your assets....');
WriteLn('Everything you own with cash value.');
WriteLn;

{ Collect data for each asset, by name. }
FOR AsstCtr := Cash TO OtherAsset DO
BEGIN
REPEAT
WriteLn;
Write(Asset_Name[AsstCtr],': ') ; ( Display the asset's name. )
ReadLn(Asset[AsstCtr]) ; ( Enter the value. )
Write(' > Is the amount correct? (Y/N) ') ; ( Verify the data. )
Read(Ch) ;
UNTIL (Ch) IN ['Y','y'] ; ( Accept the data, if correct. )

Total_Agents := Total_Agents + Asset[AsstCtr] ;
WriteLn(Printer,Asset_Name[AsstCtr]:26, ',
Asset[AsstCtr]:10:2) ;

Asset_Total(Output) ; ( Display the total for the assets. )
Asset_Total(Printer) ; ( Print the same information. )
END ; ( of List_Agents )

PROCEDURE List_Liabilities ; ( Collect information about liabilities.)

VAR
LiabCtr : LiabType ;

PROCEDURE Liab_Wrapup(VAR OutFile:Text) ;

BEGIN ( Liab_Wrapup )
Broken_line(OutFile) ;
WriteLn(OutFile,'TOTAL LIABILITIES: ':26,' ',Total_Liabilities:10:2) ;
Broken_line(OutFile) ;
WriteLn(OutFile) ;
END ; ( of Liab_Wrapup )

BEGIN ( List_Liabilities )
WriteLn;
WriteLn;
WriteLn('This section is for your debts (liabilities)') ;
WriteLn;
WriteLn(Printer) ;
WriteLn(Printer,'LIABILITIES') ;
WriteLn(Printer,'----------') ;
FOR LiabCtr := Contracts TO Debts DO
BEGIN
REPEAT
WriteLn ;
Write(Liab_Name[LiabCtr],': ') ; ( Display the item's name. )
ReadLn(Liability[LiabCtr]) ; ( Give the value of the item. )
Write(' > Is the amount correct? (Y/N) ') ; ( Verify it. )
Read(Ch) ;
UNTIL (Ch = 'Y') OR (Ch = 'y'); { Accept the data, if correct. }

Total_Liabilities := Total_Liabilities + Liability[LiabCtr];

WriteLn(Printer,Liab_Name[LiabCtr]:26,' ', Liability[LiabCtr]:10:2);

END;
Liab_Wrapup(Output); { Call the PROCEDURE; display the total. }
Liab_Wrapup(Printer); { Call the PROCEDURE; print the total. }
END; { of List_Liabilities }

PROCEDURE Finale(VAR OutFile:Text); { Finish the report. }

BEGIN { Finale }
WriteLn(OutFile);
Double_Line(OutFile);
WriteLn(OutFile);
WriteLn(OutFile,'NET WORTH: ':26,' ',
   Total_Assets - Total_Liabilities:10:2);
WriteLn(OutFile);
Double_Line(OutFile);
WriteLn(OutFile);
WriteLn(OutFile);
Write('*** Press Return to finish the printout *** ');
Read(Ch);
END; { of Finale }

BEGIN { NetWorth ... the main part of the program. }
Initialize; { The program actually starts here. }
Sign_On; { Execute the Sign_On PROCEDURE. }

GotoXY(25,18);
Write('*** TURN ON THE PRINTER ***'); { Do just what it says. }

GotoXY(20,20);
Write('*** Press the Return key when ready ***');
Read(Ch);

ClearScreen;

WriteIn;
Write('Please type your full name: ');
ReadLn(Name);

WriteIn;
Write('Enter the date of this statement (MM/DD/YY): ');
ReadLn(Date);
( The printer starts work at this point. )
Double_Line(Printer);
WriteLn(Printer);
WriteLn(Printer,'STATEMENT OF PERSONAL NET WORTH');
WriteLn(Printer,'FOR: ',Name);
WriteLn(Printer,'AS OF: ',Date);
WriteLn(Printer);
Double_Line(Printer);
WriteLn(Printer);
WriteLn;
Show_How;
List_Assets;
List_Liabilities;
Finale(Output); { Display the final number. }
Finale(Printer); { Print it, too. }
WriteLn;
Write('...press the Return key to end the report...');
ReadLn(Ch);
Write(Printer,Char(12)); { Form feed }
END. { of NetWorth }
Chapter 12

AUTOREPORT & TRIPREPORT: Generate Detailed Reports of Trip and Automobile Expenses

The methodical problem solver has an insatiable appetite for data, facts on which to base decisions that have high probabilities for success. Although the experienced business person knows that it will take time and cost money up front merely to get to and from the customer's location, he recognizes this as a necessary evil, one that must be controlled to ensure that the payback is a reasonable return on the time and money invested.

One of the best ways to exercise control over this investment is to itemize and record all reasonable details. (Are you aware that the IRS insists on this kind of written, detailed record keeping?) Then the data that have accumulated can be analyzed, the investment can be tallied in a form of direct-cost accounting, and when the business deal or the purpose of the trip has been realized the return on investment (ROI) can be quantified. Also, if an analysis indicates a budgeting problem, appropriate adjustments and alternative future paths can be defined. This can preclude problems or minimize the nonproductive use of time and money expended on business trips. The result is a probable increase in the ROI in business travel.

AUTOREPORT is designed to provide a record and an analysis of routine costs for the use of a personal auto in business. The program examines the data and calculates such items as average miles per gallon or litre for the trip, average speed, cost of fuel per mile, and total cost of operation per mile. By periodically recording and analyzing this information, you can track changing trends in the costs of operation on a trip basis.

TRIPREPORT deals directly with the many problems of controlling the all-inclusive costs of a trip beyond those of the operation of a company-owned or personal automo-
bile. The program prints a summary of nonautomobile expenses—car rentals, taxis, trains, airplanes, lodgings, meals, tips, and other familiar travel costs.

It's well worth the time to compare the source code for this program with the one in the previous chapter. There are certain similarities in the performance requirements of the two programs. This one, however, doesn't use arrays. The items are treated individually by name. Forward declarations are used in this program to provide you with an opportunity to learn to order the sequence of the procedures.

Forward declarations are a valid programming technique; however, they do add several lines of code to the listing. Normally, there are no differences in performance. As a learning experience, try your hand at revising the sequence of the procedures to preclude the need for the forward declarations.

SAMPLE RUN OF AUTOREPORT

Figure 12-1 is a reproduction of the Macintosh's screen display (Shift-Command 4) of the AUTOREPORT program's user instructions. Figure 12-2 is the screen's display of the requests for data and, in this case, includes the responses we entered through the keyboard. Figure 12-3 is a reproduction of the screen's display of the calculation performed by the program just prior to the printout. Figure 12-4 is an example of the

![AutoReport](image)

This program provides information concerned with travel expenses, specifically the costs of using an automobile...company or personal.

Answer the questions and allow the computer to do all calculations.

If no expenses were incurred for an item, enter a zero and press Return.

A hard-copy report will be provided.

*** BE SURE THE PRINTER IS READY ***

*** Press Return to continue ***

Fig. 12-1. A printout of the video display of the instructions for the AUTOREPORT program.
**AutoReport**

Briefly, state the trip's purpose? Sales calls on new accounts

<table>
<thead>
<tr>
<th>How many days did the trip take?</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>What was the starting date?</td>
<td>08/03/87</td>
</tr>
<tr>
<td>What was the ending date?</td>
<td>08/07/87</td>
</tr>
</tbody>
</table>

Name the cities visited: Austin Houston

<table>
<thead>
<tr>
<th>How many miles did you drive on this trip?</th>
<th>1247</th>
</tr>
</thead>
<tbody>
<tr>
<td>How many gallons or litres of fuel did you use?</td>
<td>118</td>
</tr>
<tr>
<td>What price did you pay per gallon or litre?</td>
<td>.99</td>
</tr>
<tr>
<td>What was the actual driving time in hours?</td>
<td>25</td>
</tr>
</tbody>
</table>

Enter the ($) Amounts Spent For:

- Tolls: 12.50
- Parking: 39.85
- Oil: 2.34
- Repairs or maintenance: 0
- Miscellaneous expenses: 12.15

**Fig. 12-2. This is the video display of the data entered that will become part of the final report for AUTOREPORT.**

**AutoReport**

REPORT FOR PERIOD: 08/03/87 THROUGH 08/07/87

Objective of the 5 day trip: Sales calls on new accounts

Cities visited: Austin Houston

The trip required: 1247 miles of driving.
The automobile used: 118.00 gallons or litres of fuel.
The automobile got: 10.57 miles per gallon or litre.
Average speed was: 49.88 miles per hour.
Cost of fuel/mile: $ 0.09

- Total fuel cost: $116.82
- Tolls paid for car: $12.50
- Parking fees: $39.85
- Oil added to car: $2.34
- Repairs to car: $0.00
- Miscellaneous: $12.15

Cost of operation per mile: $ 0.15

TOTAL OF ABOVE COSTS: 183.66

*** Press Return to continue ***

**Fig. 12-3. The video display of the report for AUTOREPORT.**
REPORT FOR PERIOD: 08/03/87 THROUGH 08/07/87

Objective of the 5 day trip: Sales calls on new accounts
Cities visited: Austin Houston

The trip required: 1247 miles of driving.
The automobile used: 118.00 gallons or litres of fuel.
The automobile got: 10.57 miles per gallon or litre.
Average speed was: 49.88 miles per hour.
Cost of fuel/mile: $0.09

Total fuel cost: $116.82
Tolls paid for car: $12.50
Parking fees: $39.85
Oil added to car: $2.34
Repairs to car: $0.00
Miscellaneous: $12.15

Cost of operation per mile: $0.15

TOTAL COST FOR 1247 MILES OF AUTOMOBILE OPERATION: $183.66

Fig. 12-4. A reproduction of the actual printout of the report generated by AUTOREPORT.

final printout. The complete source code for the AUTOREPORT program is given in Fig. 12-5.

CODE LISTING FOR AUTOREPORT

Figure 12-6 is an actual printout of the results generated by a sample run of the compiled code for TRIPREPORT. Then, the complete source code is given for the program in Fig. 12-7.
{ Create and print a report of automobile-operating expenses. }

PROGRAM AutoReport;
uses PasPrinter;

TYPE
  StringType = String[80];

VAR
  AutoCosts : Real;           { Total auto expenses }
  CarMaintain : Real;         { Car maintenance }
  CarRent : Real;             { Car rental }
  Ch : Char;                  { Keyboard character }
  Cities : StringType;        { Cities visited }
  Days : Integer;             { Number of days }
  DateEnd : StringType;       { When the trip ended }
  DateStart : StringType;     { When the trip started }
  DrivTime : Real;            { Driving time }
  Fuel : Real;                { Amt of fuel used }
  FuelPrice : Real;           { Price paid for fuel }
  I : Integer;                { Counter used in loops }
  Miles : Real;               { Miles driven }
  Misc : Real;                { Miscellaneous expense }
  Oil : Real;                 { Oil for car }
  Parking : Real;             { Parking fees }
  Tolls : Real;               { Tolls paid }
  TripPurpose : StringType;   { Purpose of trip }

{ The FORWARD declarations and PROCEDURES }
{ are listed in alphabetical order only }
{ for convenience in finding them by eye. }

PROCEDURE Auto_Out(VAR F:Text);
  FORWARD;

PROCEDURE Basic_Facts;
  FORWARD;

PROCEDURE Basic_Report(VAR F:Text);
  FORWARD;

PROCEDURE Car_Facts;
  FORWARD;

PROCEDURE Instruct;
  FORWARD;
PROCEDURE SignOn(VAR F:Text) ;
FORWARD ;

PROCEDURE Auto_Out ;  { Calculate time, speed, operating costs. }

BEGIN
  { Auto_Out }
  AutoCosts := 0 ;  { Start with the costs at zero. }
  WriteLn(F,'The trip required: ',Miles:4:0,
          ' miles of driving.');
  WriteLn(F,'The automobile used: ',Fuel:4:2,
          ' gallons or litres of fuel.');
  WriteLn(F,'The automobile got: ',Miles / Fuel:4:2,
          ' miles per gallon or litre.');
  WriteLn(F,'Average speed was: ',Miles / DrivTime:6:2,
          ' miles per hour.');
  WriteLn(F,'Cost of fuel/mile: $',
          FuelPrice /(Miles / Fuel):6:2)
  Total fuel cost: $',FuelPrice * Fuel:6:2)
  Tolls paid for car: $',Tolls:6:2)
  Parking fees: $',Parking:6:2)
  Oil added to car: $',Oil:6:2)
  Repairs to car: $',CarMaintain:6:2)
  Miscellaneous: $',Misc:6:2)
  AutoCosts := (FuelPrice * Fuel) + Tolls + Parking +
                Oil + CarMaintain + Misc ;
  WriteLn(F);
  WriteLn(F,'Cost of operation per mile: $',AutoCosts / Miles:6:2) ;
  WriteLn(F);
  WriteLn('TOTAL OF ABOVE COSTS: ',AutoCosts:6:2) ;  { To the screen }
  WriteLn ;
END ;   { of Auto_Out }

PROCEDURE Basic_Facts ;  { Get information from the keyboard. }

BEGIN    { Basic_Facts }
  GotoXY(1,5) ;
  Write('Briefly, state the trip''s purpose? ') ;
  ReadLn(TripPurpose) ;
  WriteLn ;
  Write('How many days did the trip take? ') ;
  ReadLn(Days) ;
  Write('What was the starting date? ') ;
  ReadLn(DateStart) ;
  Write('What was the ending date? ') ;
  ReadLn(DateEnd) ;
  WriteLn ;
Write('Name the cities visited: ');  
    [ This could be 'customers' as well as cities. ]
    [ Modify the words and the next ReadLn to suit ]
    [ your own requirements of preferences. ]
ReadLn(Cities);
WriteLn;
END;    { of Basic_Facts }

PROCEDURE Basic_Report;   { Print the information from Basic_Facts. }
BEGIN    ( Basic_Report )
    ClearScreen;
    WriteLn(F);
    WriteLn(F);
    WriteLn(F,
           'REPORT FOR PERIOD: ',DateStart,'
         THROUGH ',DateEnd);
    WriteLn(F);
    WriteLn(F,'
           Objective of the ',Days,'
           day trip: ',TripPurpose);
    WriteLn(F,'
           Cities visited: ',Cities);  { Or customers. Or ? }
    WriteLn(F);
END;    { of Basic_Report }

PROCEDURE Car_Facts;   { Collect facts about the use of the car. }
BEGIN    ( Car_Facts )
    Write('How many miles did you drive on this trip?  ');
ReadLn(Miles);
    Write('How many gallons or litres of fuel did you use?  ');
ReadLn(Fuel);
    Write('What price did you pay per gallon or litre?  ');
ReadLn(FuelPrice);
    Write('What was the actual driving time in hours?  ');
ReadLn(DrivTime);
    WriteLn;
    WriteLn('Enter the ($) Amounts Spent For: ');
WriteLn;
    Write('Tolls:  ');
ReadLn(Tolls);
    Write('Parking:  ');
ReadLn(Parking);
    Write('Oil:  ');
ReadLn(Oil);
    Write('Repairs or maintenance:  ');
ReadLn(CarMaintain);
    Write('Miscellaneous expenses:  ');
ReadLn(Misc);
    WriteLn;
END;    { of Car_Facts }
PROCEDURE Instruct;  { Display instructions for the user. }  
BEGIN   { Instruct }
ClearScreen;
GotoXY(20,5); { Start instructions at column 20, row 5. }
Write('This program provides information');
GotoXY(20,6);
Write('concerned with travel expenses,');
GotoXY(20,7);
Write('specifically the costs of using an');
GotoXY(20,8);
Write('automobile...company or personal.');
GotoXY(20,10);
Write('Answer the questions and allow the');
GotoXY(20,11);
Write('computer to do all calculations.');
GotoXY(20,13);
Write('If no expenses were incurred for an');
GotoXY(20,14);
Write('item, enter a zero and press Return. ');
GotoXY(20,16);
Write('A hard-copy report will be provided. ');
GotoXY(20,19);
Write('*** BE SURE THE PRINTER IS READY ***');
GotoXY(22,22);
Write('*** Press Return to continue ***');
Read(Ch);   { Wait for the key to be pressed; then continue. }
ClearScreen;
END;   { of Instruct }

PROCEDURE SignOn;   { Create the startup display. }  
BEGIN   { SignOn }
GotoXY(10,10);
FOR I := 1 TO 65 DO
BEGIN
   Write(F,CHR(42)) { 42 is the ASCII character. }  
END;
WriteLn(F);
GotoXY(18,12);
Write(F,'AUTOREPORT -- AUTOMOBILE OPERATIONS EXPENSE REPORT');
WriteLn(F);
GotoXY(10,14);
FOR I := 1 TO 65 DO
BEGIN
   Write(F,CHR(42))
END;
BEGIN { AutoReport } (* The main part of the program begins here. *)
  SignOn(Output);
  Instruct;
  Basic_Facts;             { Get the basic information. }
  Car_Facts;
  Basic_Report(Output);
  Auto_Out(Output);
  Write('*** Press Return to continue ***');
  Read(Ch);
  ClearScreen;
  GotoXY(10,8);            { Now get ready to do the printed report. }
  Write('*** Turn on the printer --');     { Remind the user. }
  WriteLn(' check and adjust the paper ***'); { Ready, get set... }
  GotoXY(18,12);
  Write('*** Press Return when ready to print ***');
  Read(Ch);              { ...and print it! }
  ClearScreen;
  GotoXY(20,10);
  { Display a notice. }
  WriteLn('*** Your printer is now at work ***');
  { Start printing. }
  SignOn(Printer);
  Basic_Report(Printer);
  Auto_Out(Printer);

FOR I := 1 TO 65 DO
BEGIN
  Write(Printer,Chr(42))
END;

WriteLn(Printer);
WriteLn(Printer,'TOTAL COST FOR ');
  Miles:4:0,
  ' MILES OF AUTOMOBILE OPERATION: $',
  AutoCosts:6:2);

FOR I := 1 TO 65 DO
BEGIN
  Write(Printer,Chr(42))
END;

ClearScreen;
GotoXY(25,10);
WriteLn('*** End of the Auto Expense Report ***');
WriteLn(Printer);  { Empty the printer's buffer.  }
GotoXY(23,15);
Write('*** Press Return to eject the report ***');
Read(Ch);
Write(Printer,Chr(12))  { Send a form feed to the printer.  }
END.  { of AutoReport }

SAMPLE RUN OF TRIPREPORT

*****************************************************************************
TRIP REPORT -- TRAVEL EXPENSE RECORD
*****************************************************************************

TRIP REPORT: 10/05/87 THROUGH 10/08/87

Objective of the 4 day trip: Service calls
Cities visited: WestPort Boston Providence

Airplane fares: $904.00
Train fares: $ 9.95
Taxicab fares: $ 35.85
Car rentals: $123.45
Meals for self: $189.95
Lodgings: $324.77
Telephone calls: $ 33.33
Tips paid: $ 53.50
Entertainment: $195.35
Valet services: $ 15.50
Miscellaneous: $ 10.00

*****************************************************************************
TOTAL COSTS REPORTED FOR THE TRIP: $1895.65
*****************************************************************************

Fig. 12-6. A reproduction of the actual printout of the report generated by TRIPREPORT.

CODE LISTING OF TRIPREPORT

{ TripReport provides a detailed trip-expense report. }

PROGRAM TripReport;
uses PasPrinter;

TYPE
StringType = String[80];
VAR
Airplane : Real ; ( Airplane fares paid )
CarRent : Real ; ( Car rental )
Ch : Char ; ( Keyboard character )
Cities : StringType ; ( Cities visited )
Days : Integer ; ( Number of days )
DateEnd : StringType ; ( When the trip ended )
DateStart : StringType ; ( When the trip started )
Entertain : Real ; ( Business entertain )
I : Integer ; ( Counter used in loops )
ItemCosts : Real ; ( Itemized expenses )
Laundry : Real ; ( Laundry/tailor/valet )
Lodging : Real ; ( Lodgings on trip )
Meals : Real ; ( Meals on the road )
Misc : Real ; ( Miscellaneous expense )
Phone : Real ; ( Phone charges )
Taxis : Real ; ( Taxicab fares paid )
Tips : Real ; ( Tips )
Trains : Real ; ( Train fares paid )
TripPurpose : StringType ; ( Purpose of trip )

( The FORWARD declarations and PROCEDURES )
( are listed in alphabetical order only )
( for convenience in finding them by eye. )

PROCEDURE Basic_Facts ;
  FORWARD ;

PROCEDURE Basic_Report(VAR F:Text) ;
  FORWARD ;

PROCEDURE Instruct ;
  FORWARD ;

PROCEDURE Items(VAR F:Text) ;
  FORWARD ;

PROCEDURE SignOn(VAR F:Text) ;
  FORWARD ;

PROCEDURE Trip_Facts ;
  FORWARD ;

PROCEDURE Basic_Facts ; ( Get information from the keyboard. )
BEGIN  
  ( Basic_Facts )
  GotoXY(1,2) ;
  Write('Briefly, state the trip''s purpose? ') ;
ReadLn(TripPurpose) ;
WriteLn ;

Write('How many days did the trip take? ') ;
ReadLn(Days) ;

Write('What was the starting date? ') ;
ReadLn(DateStart) ;

Write('What was the ending date? ') ;
ReadLn(DateEnd) ;
WriteLn ;

Write('Name the cities visited: ') ;
{ This could have been 'customers' as well as cities. }
{ You can modify the words to suit your preference. }
ReadLn(Cities) ;
WriteLn ;
END ;  { of Basic Facts }

PROCEDURE Basic_Report ; { Print the information from Basic_Facts. }
BEGIN   { Basic_Report }
  ClearScreen ;
  GotoXY(1,2) ;
  WriteLn(F) ;
  WriteLn(F) ;
  WriteLn(F,'TRIP REPORT: ',DateStart,' THROUGH ',DateEnd) ;
  WriteLn(F) ;
  WriteLn(F,'Objective of the ',Days,' day trip: ',TripPurpose) ;
  WriteLn(F,'Cities visited: ',Cities) ; { ...or customers. }
  WriteLn(F) ;
END ;   { of Basic_Report }

PROCEDURE Instruct ; { Display instructions for the user. }
BEGIN   { Instruct }
  ClearScreen ;
  GotoXY(20,5) ; { Start instructions at column 20, row 5. }
  Write('This program provides information') ;
  GotoXY(20,6) ;
  Write('concerned with travel expenses,'') ;
  GotoXY(20,7) ;
  Write('excluding the costs of using an') ;
  GotoXY(20,8) ;
  Write('automobile... company or personal.') ;
  GotoXY(20,10) ;
Write('Answer the questions and allow the');
GotoXY(20,11);
Write('computer to do all calculations.');
GotoXY(20,13);
Write('If no expenses were incurred for an');
GotoXY(20,14);
Write('item, enter a zero and press Return.');
GotoXY(20,16);
Write('A hard-copy report will be provided.');
GotoXY(20,19);
Write('*** BE SURE THE PRINTER IS READY ***');
GotoXY(17,22);
Write('... Press the Return key to continue ...');
Read(Ch);   { Wait for the key to be pressed, then continue. }
ClearScreen;
END;   { of Instruct }

PROCEDURE Items ;
BEGIN    { Items }
    ItemCosts := 0 ;   { Initialize to zero. }
    WriteLn(F,' Airplane fares: $',Airplane:6:2);
    WriteLn(F,' Train fares: $',Trains:6:2);
    WriteLn(F,' Taxicab fares: $',Taxis:6:2);
    WriteLn(F,' Car rentals: $',CarRent:6:2);
    WriteLn(F,' Meals for self: $',Meals:6:2);
    WriteLn(F,' Lodgings: $',Lodging:6:2);
    WriteLn(F,' Telephone calls: $',Phone:6:2);
    WriteLn(F,' Tips paid: $',Tips:6:2);
    WriteLn(F,' Entertainment: $',Entertain:6:2);
    WriteLn(F,' Valet services: $',Laundry:6:2);
    WriteLn(F,' Miscellaneous: $',Misc:6:2);
    ItemCosts := Meals + Misc + Airplane + Trains +
                   Taxis + Phone + Tips + Lodging +
                   Entertain + CarRent + Laundry ;
    WriteLn(F);
END;   { of Items }

PROCEDURE Trip_Facts ;   { Collect facts about the trip's costs. }
BEGIN    { Trip_Facts }
    WriteLn('Enter the amount for each item of expense:');
    WriteLn;
    Write(' Airplanes: ');
    ReadLn(Airplane);
Write(' Trains: ') ;
ReadLn(Trains) ;

Write(' Taxicabs: ') ;
ReadLn(Taxis) ;

Write(' Car rental: ') ;
ReadLn(CarRent) ;

Write(' Meals: ') ;
ReadLn(Meals) ;

Write(' Lodging: ') ;
ReadLn(Lodging) ;

Write(' Telephone: ') ;
ReadLn(Phone) ;

Write(' Tips: ') ;
ReadLn(Tips) ;

Write(' Entertainment: ') ;
ReadLn(Entertain) ;

Write(' Valet services: ') ;
ReadLn(Laundry) ;

Write(' Miscellaneous: ') ;
ReadLn(Misc) ;
WriteLn ;
END ;  { of Trip_Facts }

PROCEDURE SignOn ;  { Create the startup display. }
BEGIN  { SignOn }
GotoXY(20,7) ;
FOR I := 1 TO 45 DO
BEGIN
  Write(F,CHR(42)) ;
END ;
WriteLn(F) ;

GotoXY(25,9) ;
Write(F,'TRIP REPORT -- TRAVEL EXPENSE RECORD') ;
WriteLn(F) ;

GotoXY(20,11) ;
FOR I := 1 TO 45 DO
BEGIN
Write(F,CHR(42))
END;
Write(F);
GotoXY(15,16);
Write('*** Turn on the printer -- '){ Remind the user. }
WriteLn(' check and adjust the paper ***');

GotoXY(25,18);
Write('Press the Return key, to continue...');
Read(Ch);
END; { of SignOn }

BEGIN { TripReport. } (* The main part of the program begins here. *)
SignOn(Output);
Instruct;
Basic_Facts; { Get the basic information. }
Trip_Facts;
Basic_Report(Output);
Items(Output);

Write(' *** Press the Return key to continue *** ');
Read(Ch);
ClearScreen;

GotoXY(20,10); { Print the report now. }
SignOn(Printer);
Basic_Report(Printer);
Items(Printer);

FOR I := 1 TO 45 DO BEGIN
  Write(Printer,Chr(42));
END;

WriteLn(Printer);
Write(Printer,'TOTAL COSTS REPORTED FOR THE TRIP: $',ItemCosts:6:2);
WriteLn(Printer);

FOR I := 1 TO 45 DO BEGIN
  Write(Printer,Chr(42))
END;

WriteLn(Printer); { Empty the printer’s buffer. }
WriteLn(Printer,Chr(12)) { Send a form feed to the printer. }
END. { of TripReport }
Chapter 13

SETPRINTER: Send Control Codes to a Printer

Most popular printers, including the Apple ImageWriter, will accept external commands entered at the computer keyboard. Under software control these printers can be instructed to change fonts, faces, spacing, and other printing characteristics within the printers’ capabilities. The command sequences are usually found in the documentation supplied with the printers. This chapter’s program, SETPRINTER, provides a structure and code intended for the Apple ImageWriter; the code can be modified, however, for virtually any printer that will accept commands issued from a remote source such as the computer to which it is connected.

SETPRINTER clears the screen and displays a menu of a dozen commands. The program uses the Case construction for the responses to selected commands, which are letters of the alphabet. The Case structure recognizes both uppercase or lowercase letters. If a key is pressed that is not on the menu (not in the Case construction), nothing happens and the menu waits for a valid selection. When a letter is selected that is on the menu, the Macintosh sounds a beep to acknowledge the fact that a valid choice has been made and that the command has been sent to the printer.

The control codes given in the Case construction can be changed to conform to your specific printer, or to perform other printer functions. The menu can be expanded to include additional commands, the complete alphabet, or numbers, or other characters that then become part of the CASE statement. In SETPRINTER, the letter X is used for printing a four-line sample of the mode that has been selected from the menu. Y resets the printer to its default status. Z quits the program and returns the system to the desktop.
Fig. 13-1. The screen display of the menu for SETPRINTER.

Figure 13-1 is a reproduction (Shift-Command 4) of the Macintosh’s screen-display of the menu generated when SETPRINTER is executed.

CODE LISTING FOR SETPRINTER

{ Send control characters for an ImageWriter printer -- from a menu. }

PROGRAM SetPrinter ;
uses PasPrinter ;

CONST
   Width = 27 ;

VAR
   I   : Integer ;
   Ch  : Char ;

PROCEDURE SetCase ;   { Here’s the CASE construction. }

BEGIN
   CASE Ch OF
'A', 'a': BEGIN /* Expanded 9-CPI */
    { Chr(27) is the escape code. }
    { Chr(7) sounds the Macintosh beep. }
    Write(Printer,(Chr(27)),(Chr(110))) ;
    Write(Chr(7)) ;
    END ;
'B', 'b': BEGIN /* Pica 10-CPI */
    Write(Printer,(Chr(78))) ;
    Write(Chr(7)) ;
    END ;
'C', 'c': BEGIN /* Elite 12-CPI */
    Write(Printer,(Chr(27)),(Chr(69))) ;
    Write(Chr(7)) ;
    END ;
'D', 'd': BEGIN /* SemiCondensed 13.4-CPI */
    Write(Printer,(Chr(27)),(Chr(101))) ;
    Write(Chr(7)) ;
    END ;
'E', 'e': BEGIN /* Condensed 15-CPI */
    Write(Printer,(Chr(27)),(Chr(113))) ;
    Write(Chr(7)) ;
    END ;
'F', 'f': BEGIN /* UltraCondensed 17-CPI */
    Write(Printer,(Chr(27)),(Chr(81))) ;
    Write(Chr(7)) ;
    END ;
'G', 'g': BEGIN /* Pica Proportional */
    Write(Printer,(Chr(27)),(Chr(112))) ;
    Write(Chr(7)) ;
    END ;
'H', 'h': BEGIN /* Elite Proportional */
    Write(Printer,(Chr(27)),(Chr(80))) ;
    Write(Chr(7)) ;
    END ;
'I', 'i': BEGIN /* Bold Face */
    Write(Printer,(Chr(27)),(Chr(33))) ;
    Write(Chr(7)) ;
    END ;
'J', 'j': BEGIN /* Underline */
    Write(Printer,(Chr(27)),(Chr(88))) ;
    Write(Chr(7)) ;
    END ;
'K', 'k': BEGIN /* 6 Lines/Inch */
    Write(Printer,(Chr(27)),(Chr(65))) ;
PROCEDURE Header ;  (* This is the heading for the menu. *)

VAR
 I : integer ;

BEGIN
 GotoXY(1,1) ;

 FOR I := 1 TO Width DO  (* Width is a CONST value. *)
 Write(‘/’) ;
 FOR I := 1 TO Width DO
 Write(‘/’) ;
 WriteLn ;
 Write(‘ >>>>>>>> ’) ;
Write(' IMAGEWRITER CONFIGURATOR ') ;

WriteLn(' <<<<<<<<<' ) ;
FOR I := 1 TO Width DO
  Write('/' ) ;
for I := 1 to Width do
  Write('\' ) ;
  WriteLn ;
END ;

PROCEDURE Menu ;  { Display the menu }
BEGIN
  WriteLn ;
  WriteLn('----------------------------------------------') ;
  WriteLn ;
  WriteLn('A) Expanded 9-CPI
B) Pica 10-CPI
C) Elite 12-CPI
D) SemiCondensed 13.4-CPI
E) Condensed 15-CPI
F) Ultra Condensed 17-CPI
G) Pica Proportional
H) Elite Proportional
I) Bold Face
J) Underline
K) 6 Lines per Inch
L) 8 Lines per Inch
M) Line Feed
N) Form Feed

X) TEST PRINT
Y) RESET
Z) QUIT

----------------------------------------------') ;
  WriteLn ;
END ;  { of Menu }

BEGIN { SetPrinter }
  REPEAT
    Header ;
    Menu ;
    WriteLn ;
    Write(':>----------------Select one option at a time >----------------:<: ' ) ;
    Read(Ch) ;
    SetCase ;
    UNTIL Ch IN ['Z','z']
  END.  { of SetPrinter }
Chapter 14

TIMETREND: Use Time-Trend Analysis to Smooth the Rough Spots

When you examine the day-to-day or month-to-month histories of a company’s performance, you often discover that a graph of the data resembles the profile of an amusement park’s rollercoaster. Therefore, preparing a forecast or a budget on the basis of historic data can seriously test the expertise and know-how of the forecaster. But, you must build on historic data, and you can after it has been made meaningful or translated into a usable form.

Trends may lie hidden in the data and there are many techniques for discovering and quantifying those trends. One method is known as time-trend analysis. It is also known as calculating the moving average. This chapter’s program, TIMETREND, takes raw data for a series of quantitative events and continuously averages the data. The effect is to smooth out the peaks and valleys and in this way to make the trend of the data quite clear and unambiguous. If the smoothed data were to be plotted, you would see immediately whether the direction was up, down, or static.

TIMETREND queries the user for the raw data and the number of periods for the moving-average calculations. A complete table of data, raw and smoothed, is computed, displayed, and it can be printed, if desired.

Many companies, financial and marketing managers, and statisticians use this approach to data analysis. Moving averages developed for raw data that cover reasonably short periods of time, say 15 months or so, can be quite revealing. The calculation of moving averages, however, is not limited to short-term data analysis. Financial institutions may use 20-year moving averages to determine or test the validity of bad-debt provisions for federal income tax purposes.

Although the formulas and the process are not complex, they are time-consuming
and, at the very least, somewhat of a bore to do manually or even with the aid of a hand-held electronic calculator. TIMETREND does all the calculations for you, displays the results on the screen, and at your signal, makes as many printouts as you want.

A MARKETING MANAGER’S PROBLEM SOLVER

In forecasting sales of products, systems, components, or services, the most-recent historic data are the important bases for improving the accuracy of the forecast. Such data are always related to time—days, months, quarters, years, and so on. In a multi-product marketing company or in a service operation, sales and financial forecasts (budgets) must be developed for each item in the catalog. A forecast also must be developed for the aggregate of all products and services.

Historic reviews of the movement of products or services with respect to time periods, when tabulated and displayed or graphed as raw or actual data, often resemble the teeth of a defective saw blade or the jagged profile of a cliff more than a smooth, idealistic, upward-moving line. It can be difficult and sometimes impossible to determine the direction of the trend, up or down, from the raw data. TIMETREND accepts the data, regardless of how “sharp its teeth” or how “jagged its edges,” and, at the speed of the Macintosh, calculates and redispays it to reveal the unambiguous trend of the data. Moreover, on the basis of the computed trend, the program suggests and displays a forecast for the next time period as an aid to planning and budgeting.

In this chapter I will use moving averages to analyze relatively short-term trends in marketing and manufacturing. The numbers may relate to days, weeks, months, or years; and to pennies, dollars or units—whichever suits the need. Dollar signs are not used in the program. Therefore, the references or the classifications for the type of data are for you to define.

Moving averages can be used to reveal trends in productivity, be it the output produced by a single assembly-line, by a combination of facilities, or by a complete plant. The analysis of trends in sales, which have a significant effect on production, becomes a valuable aid in pinpointing problem areas, in comparing two or more product lines, in budgeting, and in forecasting. For an analysis of production, the data entered can be classified as units of time, head count, parts used, input, output, dollars, open jobs, and so on.

A SAMPLE RUN OF TIMETREND

Let’s simulate a typical run as might be created by a sales or marketing manager. In our example, for actual (raw) data we will use numbers that have distinctly sharp peaks and valleys and note how TIMETREND handles them.

The data will span a 16-month period of sales activity for fictitious company ABC. There is no magic to the number 16. It happens to be the number of time periods for which the hypothetical marketing or sales manager of company ABC has information that he knows is accurate and current. More important, it covers what has been a rather volatile period, one in which the sales volume moved up and down alarmingly, making it difficult to ascertain any sort of meaningful trend.

The requests for information that are displayed on the screen are shown below. Your responses are shown underlined, at the right of each request:

120
TIME TREND ANALYSIS WITH MOVING AVERAGES

How many TIME PERIODS are to be entered?: 16
Enter a value for each of the 16 time periods:
1: 11287
2: 15445
3: 23479
4: 14876
5: 15434
6: 27565
7: 22835
8: 9678
9: 11034
10: 10222
11: 14943
12: 17042
13: 38904
14: 16832
15: 50089
16: 34502

Enter the number of periods to be averaged: 3

When you press Return after the last entry, the screen clears and the table of data appears with three headings:

<table>
<thead>
<tr>
<th>PERIOD</th>
<th>RAW DATA</th>
<th>SMOOTHED DATA</th>
</tr>
</thead>
</table>

The numbers of the 16 individual time periods, in this case equivalent to months, appear in sequence below the PERIOD column. The RAW DATA, the actual, historic data that we entered previously, appears appropriately under the center heading. The SMOOTHED DATA that has been generated by the computer appears in the last column. Each smoothed data entry is derived from the average of the immediately previous raw data periods. In this example, three periods are averaged, because three was entered in response to the request to "Enter the number of periods to be averaged."

Based on the trend of the data that was smoothed for a moving average of three periods, a forecast is displayed for period 17. As you can see from the hard copy printout, this particular forecast is not terribly exciting. But, it is based on the average of the 14th, 15th, and 16th time periods, which are the most-recent historic facts. If the display that has been created is too long to appear in whole on the screen, an option is provided so that the table can be repeatedly brought back to the screen for study:

Repeat the display? (Y/N):

In addition to the option to repeatedly display the table of data, you can change the averaging period as many times as you wish. The screen asks:
Change the number of periods to be averaged? (Y/N):

This is useful for optimizing the smoothed data. If you type <Y>, the screen asks:

Enter the number of periods to be averaged:

You can change the number as many times as you want until you are satisfied with the SMOOTHED DATA, which changes each time you revise the number of the averaging period.

Then, when you have arrived at an acceptable set of data, you are given the option to make a hard copy. You can make as many printouts as you want by typing the letter <Y> in response to the screen's query, followed, as usual, by Return. Finally, the screen instructs:

*** For a printout, turn on the printer ***

When ready, type 'Y' to print ... 'N' to quit:

### TWO SAMPLE RUNS OF TIMETREND

<table>
<thead>
<tr>
<th>PERIOD</th>
<th>RAW DATA</th>
<th>SMOOTHED DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11287.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>15445.0</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>23479.0</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>14876.0</td>
<td>16737.0</td>
</tr>
<tr>
<td>5</td>
<td>15434.0</td>
<td>17933.3</td>
</tr>
<tr>
<td>6</td>
<td>27565.0</td>
<td>17929.7</td>
</tr>
<tr>
<td>7</td>
<td>22835.0</td>
<td>19291.7</td>
</tr>
<tr>
<td>8</td>
<td>9678.0</td>
<td>21944.7</td>
</tr>
<tr>
<td>9</td>
<td>11034.0</td>
<td>20026.0</td>
</tr>
<tr>
<td>10</td>
<td>10222.0</td>
<td>14515.7</td>
</tr>
<tr>
<td>11</td>
<td>14943.0</td>
<td>10311.3</td>
</tr>
<tr>
<td>12</td>
<td>17042.0</td>
<td>12066.3</td>
</tr>
<tr>
<td>13</td>
<td>38904.0</td>
<td>14069.0</td>
</tr>
<tr>
<td>14</td>
<td>16832.0</td>
<td>23629.7</td>
</tr>
<tr>
<td>15</td>
<td>50089.0</td>
<td>24259.3</td>
</tr>
<tr>
<td>16</td>
<td>34502.0</td>
<td>35275.0</td>
</tr>
</tbody>
</table>

THE TREND FORECASTS PERIOD 17 AS: 33807.7

NUMBER OF PERIODS AVERAGED: 3

Fig. 14-1. A table of raw and smoothed data for 16 periods of activity with averaging for three periods. Note the forecast for the 17th period.
MOVING AVERAGES - TIME TREND ANALYSIS

<table>
<thead>
<tr>
<th>PERIOD</th>
<th>RAW DATA</th>
<th>SMOOTHED DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11287.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>15445.0</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>23479.0</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>14876.0</td>
<td>0.0</td>
</tr>
<tr>
<td>5</td>
<td>15434.0</td>
<td>0.0</td>
</tr>
<tr>
<td>6</td>
<td>27565.0</td>
<td>16104.2</td>
</tr>
<tr>
<td>7</td>
<td>22835.0</td>
<td>19359.8</td>
</tr>
<tr>
<td>8</td>
<td>9678.0</td>
<td>20837.8</td>
</tr>
<tr>
<td>9</td>
<td>11034.0</td>
<td>18077.6</td>
</tr>
<tr>
<td>10</td>
<td>10222.0</td>
<td>17309.2</td>
</tr>
<tr>
<td>11</td>
<td>14943.0</td>
<td>16266.8</td>
</tr>
<tr>
<td>12</td>
<td>17042.0</td>
<td>13742.4</td>
</tr>
<tr>
<td>13</td>
<td>38904.0</td>
<td>12583.8</td>
</tr>
<tr>
<td>14</td>
<td>16832.0</td>
<td>18429.0</td>
</tr>
<tr>
<td>15</td>
<td>50089.0</td>
<td>19588.6</td>
</tr>
<tr>
<td>16</td>
<td>34502.0</td>
<td>27562.0</td>
</tr>
</tbody>
</table>

THE TREND FORECASTS PERIOD 17 AS: 31473.8

NUMBER OF PERIODS AVERAGED: 5

Fig. 14-2. The table of data shown in Fig. 14-1 has been revised to show the effects of averaging for five periods.

If we select <N> instead of <Y>, the program ends.

Figures 14-1 and 14-2 are two printouts that TIMETREND generated. They use the raw data from the above example of how to run the program.

The first printout, Fig. 14-1, demonstrates the smoothing effect when an averaging period of three is used. Figure 14-2 takes the same data and smooths it over five periods. Note that the forecast for period 17 is changed, as are the calculations for the smoothed data.

CODE LISTING FOR TIMETREND

```pascal
{ TimeTrend accepts raw, unsmoothed data, and generates a }  
{ smoothed set of data from which it generates a table of }  
{ moving averages that reveal trends related to time. }  

PROGRAM TimeTrend ;
uses PasPrinter ;

```

123
CONST
MaxNumPeriod = 50 ;
{ Change the CONST to modify the }  
{ maximum number of periods the }  
{ program will deal with. }  
{ If this is changed, be sure to }  
{ make similar changes in the }  
{ error-trap message displayed by }  
{ PROCEDURE Facts_Only ; }  

TYPE
PerNumType = 1 .. MaxNumPeriod ;
 StringType = String[80] ;

VAR
Ch : Char ;  
{ Keyboard response character. }  
I,J,K,L : Integer ;  
{ Identifiers for the FOR loops. }  
NumAveraged : Integer ;
NumPeriods : Integer ;
Period_Value : ARRAY [PerNumType] OF Real ;
Trend_Value : Real ;

PROCEDURE SignOn ;  
{ Display the startup message. }  
BEGIN  
{ BEGIN the SignOn PROCEDURE. }  
GotoXY(15,8) ;  
{ Send the cursor to C15/R8. }  
FOR I := 1 TO 50 DO  
{ This loop has 50 iterations. }  
BEGIN  
{ Start the FOR...DO loop. }  
Write(Char('*')) ;  
{ The character to be written. }  
END ;  
{ END of the FOR...DO loop. }  
GotoXY(20,10) ;  
{ Send the cursor to C20/R10. }  
Write('TIME TREND ANALYSIS with MOVING AVERAGES') ;
GotoXY(15,12) ;  
{ Send the cursor to C15/R12. }  
FOR J := 1 TO 50 DO  
{ Same as first FOR...DO loop. }  
BEGIN  
{ Begin the FOR loop. }  
Write(Char('*')) ;  
{ End the FOR loop. }  
END ;  
GotoXY(20,16) ;  
Write('*** Press the Return key to continue *** ') ;
Read(Ch) ;
END ;  
{ of SignOn }  

PROCEDURE Averaging_Period ;  
{ Get the number for averaging. }  
BEGIN  
WriteLn ;  
Write(' Enter the number of periods to be averaged: ') ;
ReadLn(NumAveraged) ;
WriteLn
END ;  
{ of Averaging_Period }
PROCEDURE Facts_Only;  
  \{ Get the facts from the keyboard. \}  

VAR  
  Period_Counter : PerNumType;  

BEGIN  \{ Facts_Only \}  
  \{ "maximum" In the next line is the same as the CONST value. \}  
  Write('How many TIME PERIODS are to be entered? (50 maximum): ');  
  ReadLn(NumPeriods);  \{ Get the keyboard entry. \}  

IF (NumPeriods > 50) THEN  \{ The 50 must be the same as CONST, \}  
  BEGIN  \{ otherwise this error-trap is wrong. \}  
    WriteLn;  
    Write('That''s more than 50. Please choose another quantity: ');  
    ReadLn(NumPeriods);  \{ Get the keyboard entry, if needed. \}  
  END;  

WriteLn;  
WriteLn('Enter a value for each of the ',NumPeriods,' periods:');  
WriteLn;  

FOR Period_Counter := 1 TO NumPeriods DO  
  BEGIN  \{ Loop for counting up to NumPeriods. \}  
    Write(' ',Period_Counter):3,': ');  
    ReadLn(Period_Value[Period_Counter]);  
  END;  

Averaging_Period;  \{ Execute the PROCEDURE Averaging_Period. \}  
WriteLn;  
WriteLn;  
WriteLn;  
END;  \{ of Facts_Only \}  

PROCEDURE Pattern_1 (VAR F:Text);  

BEGIN  \{ Pattern_1 \}  
  Write(F,')');  
  FOR L := 1 TO 70 DO  \{ Repeat the 'Write' expression 70 times. \}  
    BEGIN  
      Write(F,Char('-'));  
    END;  
  WriteLn(F);  
END;  \{ of Pattern_1 \}  

PROCEDURE Pattern_2 (VAR F:Text);  \{ Send a pattern to F. \}  

BEGIN  \{ Pattern_2 \}  
  Write(F,'');  
  FOR K := 1 TO 50 DO  \{ Repeat the 'Write' expression 50 times. \}  
    BEGIN  
      Write(F,'-');  \{ Write a series of 50 hyphen characters. \}  
    END;  
END;  

125
PROCEDURE Make_The_Table (VAR F:Text); { Generate the time trend table. }

VAR
  Counter : PerNumType;
  NumSequence : Integer;

BEGIN  { Make_The_Table }
  Pattern_1(F);   { Execute the PROCEDURE. }
  WriteLn(F);
  WriteLn(F,
    MOVING AVERAGES - TIME TREND ANALYSIS’);
  WriteLn(F);
  Pattern_1(F);   { Execute the PROCEDURE. }
  WriteLn(F);
  WriteLn(F,
    ’PERIOD RAW DATA SMOOTHED DATA’);
  Pattern_2(F);   { Execute the PROCEDURE. }
  WriteLn(F);

FOR NumSequence := 1 TO NumPeriods + 1 DO
  BEGIN
    Trend_Value := 0;   { Set Trend_Value to zero. }
    IF (NumSequence > NumAveraged) THEN
      BEGIN
        FOR Counter := 1 TO NumAveraged DO
          { Set Trend_Value’s value. }
        Trend_Value := Trend_Value +
            Period_Value[NumSequence - Counter];
          { Set Trend_Value to a new value. }
        Trend_Value := Trend_Value / NumAveraged;
      END;
    ELSE
      BEGIN
        Write(F,
          ’,NumSequence:7);
        Write(F,
          ’,Period_Value[NumSequence]:7:1);
        WriteLn(F,
          ’,Trend_Value:7:1);
      END

ELSE

BEGIN
  WriteLn(F);
  Pattern_2(F);   { Execute the PROCEDURE. }
  Write(F,
    ’,‘THE TREND FORECASTS PERIOD ’);
  WriteLn(F,NumSequence,’ AS: ’,Trend_Value:7:1);
  Pattern_2(F);   { Execute the PROCEDURE. }
  Write(F,
    ’);
  WriteLn(F,‘NUMBER OF PERIODS AVERAGED: ’,NumAveraged);
END;
BEGIN
  { TimeTrend }
  { This is the main-part of the code. }
  { Each of the PROCEDURES is called }
  { and executed in turn. }
  SignOn;
  ClearScreen;
  Facts_Only;  { Execute the PROCEDURE. }
REPEAT
  ClearScreen;
  Make_The_Table(Output);  { Display the moving averages table. }
  WriteLn;
  Write(' Repeat the Display? (Y/N): ');
  ReadLn(Ch);
  IF Ch IN ['Y', 'y'] THEN
    BEGIN
      WriteLn;
      Write(' Change the number of periods to be averaged? (Y/N): ');
      Read(Ch);
      IF Ch IN ['Y', 'y'] THEN
        BEGIN
          Averaging_Period;  { Recalculate for the averaging period. }
        END
      ELSE
        BEGIN
          ClearScreen;
          Make_The_Table(Output);  { Generate the revised table. }
        END
    END;
  UNTIL Ch IN ['N', 'n'];
REPEAT
  WriteLn;
  WriteLn('*** For a printout, turn on the printer ***');
  WriteLn;
  Write('When ready, type "Y" to print ... "N" to quit: ');
  ReadLn(Ch);
  IF Ch IN ['Y', 'y'] THEN
    BEGIN
      Make_The_Table(Printer);  { Print the moving averages table. }
      Write(Printer, Chr(12));  { Do a form feed after the printout. }
    END;
  UNTIL Ch IN ['N', 'n'];
END.  { of TimeTrend }
Chapter 15

PLOTTER: A Graph With Equidistant Data Points

There are many ways to make visual presentations of numeric information. You have already looked at programs in this book that generate tables of data and histograms. Another option is offered with PLOTTER, a program that presents data as a series of points within the boundaries of an X-axis and a Y-axis. By connecting the data points with a straight edge, one data point to the next data point in succession, you gain a view of the data that enables rapid trend analysis, or other series-related examinations and evaluations.

PLOTTER contains algorithms that make the code interesting to the Turbo Pascal programmer. As an applications program, it can be exceptionally valuable to the marketer, manufacturer, engineer, or student who deals with data. PLOTTER queries the user for basic parameters such as the width of the graph and the number of data points to be plotted. Then it asks for the data to be entered. When the declared number of data points has been entered, the program immediately prints the graph. The data points are placed equidistant from one another in the sequence in which they were entered.

Each data point is indicated by an asterisk character placed in its appropriate X-Y axis by the program. The character used to indicate the X-Y positions of the data points can be changed to any character within your printer’s capability. The completed plot is displayed on the screen. A <Yes/No> option to print hard copies of the plot is offered. The plot is printed horizontally. It is properly oriented by rotating the printed plot 90-degrees counterclockwise; the printed width is actually the height of the plot. Connect the data points using a straightedge to complete the plot.

PLOTTER is easy to use. Instructions are comprehensively displayed on the screen.
as you move through the program. PLOTTER provides both screen displays and printouts. Because of the physical limitations of the screen, a printout is essential for a proper examination of the graph and its data points. Therefore, output to a printer is a feature incorporated into the code. You can make any number of printouts of the graph.

**A SAMPLE RUN OF PLOTTER**

When you compile and execute the program, the SignOn procedure is the first one called. This displays the name of the program. Press the Return key and the screen clears and this notice is displayed:

*** TURN ON THE PRINTER BEFORE ENTERING ANY DATA ***

The next instruction and your response (shown with an underline) is:

**Give the width of the graph to be printed in inches:** 4

If your printer produces 10 characters per inch, the printout of the graph will actually be four inches wide. The length of the graph will be related to the number of data points you enter. For 10 data-point entries, a reasonable number, you can generally estimate a length-to-width ratio of about 1.2:1, which is adequate for many purposes. Each data point is equidistant from its preceding and succeeding data points.

It is practical to make the width of the graph proportional to the largest data-value that you will enter. For example, if you are going to make a series of entries with the largest value being 530, respond to the “width of the graph” question with the number 5.3. This will enable you to produce a graph that will be 5.3 inches wide with the value 530 at the end of the Y-axis. This adequately fits a sheet of printer paper that is 8.5 inches wide. When you are printing at the normal 10 characters-per-inch scale, the maximum graph size is 7.9 inches.

Let’s continue to create our sample graph with a width of four.

**How many data points do you want to plot? (50 Max): 10**

Do NOT enter negative values.

Now, you can start entering values for the data points. The program keeps count until it reaches the number you entered; in the code is assigned as the value of NumPoints. The letter I in the FOR...DO loop carries the count, which appears as part of the statement in the BEGIN...END expression that follows.

**ENTER THE VALUES FOR:**

1: 100  
2: 150  
3: 375  
4: 350  
5: 400  
6: 300
The code formats the numbers, which are of the real type, to four places. The printer repeats each value when you press the Return key. After the value for the 10th data point has been entered, the PrintStatistics procedure is called. The screen and the printer summarize simultaneously:

- Largest data value: 400.
- Top of the Graph has a value of: 400.
- Width of Graph: 4 inches at 10 cpi.

The printing and displaying of the graph, with the data points placed where you had indicated, start at this point. The screen and the printer keep pace with one another. After the printout is completed, the screen asks:

Do another printout of the graph? (Y/N):

The REPEAT...UNTIL Ch IN['N','n'] routine controls the exit or continuation of the program. Y calls the WriteGraph Procedure, which sends the output of WriteGraph to the video display and to the printer. The N response ends the program.

SAMPLE PRINTOUTS OF PLOTTER

In printing the plot of the data points within the graph’s boundaries, the X- and Y-axes appear transposed. The width of the graph, four inches in this example, is actually the height. Merely turn the graph sideways when you use it. Draw lines connecting each asterisk to the next. You will have a graphic display of the data that is plotted by this program as a series of points, and you didn’t have to buy, rent, or borrow special drawing tools. Your ImageWriter does very nicely in this application, simulating a plotter-printer.

Two printouts of the data we entered earlier are shown. In Fig. 15-1, the data points are plotted in their relative positions on the graph. In Fig. 15-2, the data points have been linked together on the graph, and the page has been properly oriented, rotated 90-degrees relative to the first graph.
DATA TO BE PLOTTED:

100.0000
150.0000
375.0000
350.0000
400.0000
300.0000
175.0000
250.0000
275.0000
325.0000

Largest data value: 400.0000
Top of the graph has a value of: 400.0000
Relative width of the graph: 4.00

Fig. 15-1. The plot for the data as generated and printed by the PLOTTER program.
CODE LISTING FOR PLOTTER

{ This program places plot a set of equidistant } 
{ data points on a calibrated graph. The finished } 
{ plot is displayed on the screen and a printout } 
{ is made. }

PROGRAM Plotter ;
uses PasPrinter ;

CONST
MaxNumPoints = 50 ; { 50 is the maximum number of data points. }
{ To increase or decrease the "maximum," }
{ change the number 50 to the desired one. }

VAR
    Ch : Char ;
    Factor,Largest,TensWidth,Width : Real ;
    I,L,NumPoints : Integer ;
    Points : ARRAY [1..MaxNumPoints] OF Real ;

PROCEDURE Sign_On ; { Display the startup message. }
BEGIN    { Sign_On }
GotoXY(18,8); { Go to column 18, row 10. } 
FOR L := 1 TO 50 DO 
BEGIN { Display a line of fifty asterisk. } 
Write('*'); 
END;

GotoXY(21,10); 
WriteLn('LINEPLOT - PLOT A GRAPH OF EQUIDISTANT POINTS');

GotoXY(18,12); 
FOR L := 1 TO 50 DO 
BEGIN { Same as above. } 
Write('*');
END;

GotoXY(18,16); 
Write('Press Return to continue...'); 
Read(Ch); 
ClearScreen; { Clear the screen. } 
END; { of Sign_On }

PROCEDURE Do_The_Statistics (VAR OutFile:Text); 
BEGIN { Do_The_Statistics } 
WriteLn(OutFile); 
WriteLn(OutFile,'Largest data value: ', Largest:10:4); 
WriteLn(OutFile,'Top of the graph has a value of: ', Largest:10:4); 
{ Recommended: use a max width of 7.5 to keep the } 
{ width of the video display completely in view. } 
{ The max for the printout on an Imagewriter II } 
{ should be 9.5; otherwise the graph will wrap on } 
{ the hard copy as well as on the screen. } 
WriteLn(OutFile,'Relative width of the graph: ', Width:2:2); 
WriteLn(OutFile); 
END; { of Do_The_Statistics }

PROCEDURE Draw_The_Graph (VAR OutFile:Text); 

VAR 
I,K : Integer; 
NumSpaces : Integer;

PROCEDURE Draw_The_Axis; { Print and display the Y axis. }
VAR
  L : Integer;
BEGIN  { Draw_The_Axis }
  FOR L := 1 TO Trunc(TensWidth / 5.0) DO
    Write(OutFile, '+-----');
    { The X-axis is composed of plus and minus signs. }
    WriteLn(OutFile, '+');
END;  { of Draw_The_Axis }

PROCEDURE Write_Spaces (Num:Integer);
BEGIN  { Write_Spaces }
  FOR L := 1 TO Num DO
    Write(OutFile, ' ');
END;  { Write_Spaces }
BEGIN  { Draw_The_Graph } { Print & display the data points }
  WriteLn(OutFile);
  Draw_The_Axis;
  FOR I := 1 TO NumPoints DO
    BEGIN { Truncate the data points in plotting the graph. }
      FOR K := 1 TO Trunc(0.5 * TensWidth / NumPoints) DO
        BEGIN
          { The colon character outlines the graph. }
          Write(OutFile, ':');
          Write_Spaces(Trunc(TensWidth - 1));
          WriteLn(OutFile, ':');
        END;
      NumSpaces := Trunc(Factor * Points[I]);
      IF (NumSpaces = 0) THEN  { Point at start. }
        BEGIN
          { The letter 'o' is the datapoint marker. }
          { It may be changed in the next line to }
          { represent any printable character. }
          Write(OutFile, 'o');
          Write_Spaces(Trunc(TensWidth - 1));
          { The colon character outlines the graph. }
          WriteLn(OutFile, ':');
        END
      ELSE
        BEGIN
          IF (NumSpaces = Trunc(TensWidth)) THEN
            BEGIN
...
BEGIN { Plotter }
Sign_On;

GotoXY(15,3);
{ Important advice for the user. }
{ The program will stall, if the }
{ printer is off when the program }
{ tries to send to the printer. }
WriteLn('*** TURN ON THE PRINTER BEFORE ENTERING DATA ***');

GotoXY(5,6);
{ Make the requests for data to be entered. }
Write('Give the relative width of the graph to be printed : ');
ReadLn(Width);

WriteLn;
GotoXY(5,8);
{ The "50 Max" should match the CONST MaxNumPoints = 50, or ? }
Write('How many data points do you want to plot? (50 Max): ');
ReadLn(NumPoints);  { Enter the number of data points. }

{ If the user enters a number greater than MaxNumPoints, }

BEGIN { The colon character outlines the graph. }
Write(OutFile,':') ;
Write_Spaces(Trunc(TensWidth - 1)) ;
{ The letter 'o' is the datapoint marker. }
{ It may be changed to any printable character. }
WriteLn(OutFile,'o') ;

END

ELSE

BEGIN
{ The colon character outlines the graph. }
Write(OutFile,':') ;
Write_Spaces(NumSpaces - 1) ;
{ The letter 'o' is the datapoint marker. }
{ It may be changed to any printable character. }
Write(OutFile,'o') ;
Write_Spaces(Trunc(TensWidth - 1 - NumSpaces)) ;
{ The colon character outlines the graph. }
WriteLn(OutFile,':') ;

END ;

END ;

Draw_The_Axis;
END;    { Draw_The_Graph }

SIGN

GotoXY(15,3);
{ Important advice for the user. }
{ The program will stall, if the }
{ printer is off when the program }
{ tries to send to the printer. }
WriteLn('*** TURN ON THE PRINTER BEFORE ENTERING DATA ***');

GotoXY(5,6);
{ Make the requests for data to be entered. }
Write('Give the relative width of the graph to be printed : ');
ReadLn(Width);

WriteLn;
GotoXY(5,8);
{ The "50 Max" should match the CONST MaxNumPoints = 50, or ? }
Write('How many data points do you want to plot? (50 Max): ');
ReadLn(NumPoints);  { Enter the number of data points. }

{ If the user enters a number greater than MaxNumPoints, }
{ it will be given the value of the CONST MaxNumPoints. }
IF (NumPoints > MaxNumPoints) THEN
  NumPoints := MaxNumPoints;

  GotoXY(10,12);
  WriteLn('(Do not enter negative values.)');
  GotoXY(10,14);
  WriteLn('Enter the values for the data points:');
  WriteLn;
  WriteLn(Printer,'DATA TO BE PLOTTED: ');
  WriteLn(Printer);

  FOR I := 1 TO NumPoints DO
    BEGIN
      Write('Point ',I,': ');
      ReadLn(Points[I]);
      WriteLn(Printer,',Points[I]:10:4);"
    END;

  Largest := 0.0; { Find the greatest data value for scaling. }
  TensWidth := Width * 10.0;

  FOR I := 1 TO NumPoints DO
    IF (Largest < Points[I]) THEN
      Largest := Points[I];
      Factor := TensWidth / Largest;

  Do_The_Statistics(Output);
  Do_The_Statistics(Printer);

  REPEAT
    Draw_The_Graph(Output); { Display the graph and do }
    Draw_The_Graph(Printer); { a printout, too. Then, }
    Write(Printer,CHR(12)); { after printing, do a form }
    WriteLn; { feed and offer an option. }
    WriteLn;

    Write('Do another printout of the graph? (Y/N): ');
    ReadLn(Ch);
    WriteLn;
  UNTIL Ch IN ['N','n'] { If "N" or "n" are not typed, }
  { The program assumes the user }
  { wants another printout. }
  { "N" or "n" ends the program. }
END. { of Plotter }
Chapter 16

METRICSENGLISH: Use of the CASE Statement in a Program for English/Metric Conversions

Here's a fast-running program that helps with the changeover from the English (Standard) system of weights and measures to the metric system of data expression. It takes less than six seconds from the moment you execute the program to see the main menu of the METRICSENGLISH program displayed on the screen.

METRICSENGLISH generates a full-screen menu offering 25 user options. Options 1 through 12 are for converting quantities from English or Standard to Metric values. Options 13 through 24 are the converse, converting quantities from Metrics to English or Standard values. Option 25 is for exiting the program and returning instantly to the Macintosh desktop.

The precision of the calculations is two places to the right of the decimal point. The precision can be extended, or reduced, by changing the formatting in the CASE statements. Note the last digit in each statement is :2. The :2 means display the calculations two places to the right of the decimal point. This can be changed to :4, for example, providing a format that includes four places to the right of the decimal point. Note too, the out-of-range error trap coded into the main part of the program.

The first procedure called is SignOn. This displays the name of the program. The BigMenu Procedure is called next, which makes a call from within itself to the very short ArtsyLine Procedure that displays a line of 38 characters. The title is displayed surrounded above and below by the ArtsyLine border.

And, now, the menu itself appears; shown in Fig. 16-1.

THE MENU OFFERS 25 CHOICES

When you choose an item from 1 to 24, the menu remains in place and a new in-
structure appears just below it. Let’s choose item 11, converting English pounds to metric kilograms:

Enter the quantity to be converted: 3.6789

3.6789 is entered as the number of pounds to be converted to kilograms. Press Enter and the computer responds instantly with a new line of information displayed below the previous one:

3.6789 Pounds = 1.67 kilograms

And, below that line appears the instruction:

*** Press Return to continue ***

When you do press the Return key, a series of spaces or blanks is written over the areas that contained the information for the calculation just completed and displayed at column 16 of rows 21, 22, and 25. This ends the Convert_Quantity procedure. The program breaks out of the REPEAT ... UNTIL loop when 25 is chosen from the menu.

An error trap is incorporated. For example, suppose you inadvertently choose 26, which doesn’t exist on the menu. The following line appears below the menu:
The out-of-range message clears, and the program is ready for another choice to be made from the menu. Choosing 25 at the menu ends the program.

**CODE LISTING FOR METRICS-ENGLISH**

( Conversions of quantities to-and-from Metrics and English. )

PROGRAM Metrics_English ; { The underscore between the two words of the program's name can also be used in Procedure names. It can help readability. }

VAR
  Ch : Char ;
  L,Choice : Integer ;

PROCEDURE Sign_On ; { Generate and display the program's banner. } BEGIN { Sign_On }
  GotoXY(15,8) ; { Banner starts at column 15, row 10. }
  FOR L := 1 TO 50 DO { Do the next task 50 times. }
      BEGIN
        Write('|') ; { The task: write the stick character. }
      END ;
  GotoXY(17,10) ;
  Write('METRICS -- METRICS-ENGLISH-METRICS CONVERSIONS') ;
  GotoXY(15,12) ;
  FOR L := 1 TO 50 DO { Again, do the next Write 50 times. }
      BEGIN
        Write('|')
      END ;
  GotoXY(22,20) ;
  Write('Press Return to continue... ') ;
  Read(Ch) ;
END ; { of Sign_On }

PROCEDURE Artsy_Line (VAR OutFile:Text) ; { Generate a graphics border. }
BEGIN { Artsy_Line }
  FOR L := 1 TO 76 DO { Loop 76 times. }
      BEGIN { Begin the loop. }
      FOR L := 1 TO 40 DO { Loop 40 times. }
          BEGIN { Begin the inner loop. }
          END ;
      END ;
  END ;
  Write('') ;
  Read(Ch) ;
END ;

141
PROCEDURE Big_Menu;  { Display a menu of options. }

BEGIN  ( Big_Menu )
Artsy_Line(Output);  { Call the PROCEDURE. }
GotoXY(22,2);
WriteLn('METRICS-ENGLISH-METRICS CONVERSIONS');
Artsy_Line(Output);  { Call it again. }

WriteLn;

(* Display 24 options in two columns. *)
WriteLn(' 1 - Inches to millimeters',
     '13 - Millimeters to inches');
WriteLn(' 2 - Inches to meters',
     '14 - Meters to inches');
WriteLn(' 3 - Feet to meters',
     '15 - Meters to feet');
WriteLn(' 4 - Miles to kilometers',
     '16 - Kilometers to miles');
WriteLn(' 5 - Teaspoons to milliliters',
     '17 - Milliliters to teaspoons');
WriteLn(' 6 - Tablespoons to milliliters',
     '18 - Milliliters to tablespoons');
WriteLn(' 7 - Pints to liters',
     '19 - Liters to pints');
WriteLn(' 8 - Quarts to liters',
     '20 - Liters to quarts');
WriteLn(' 9 - Gallons to liters',
     '21 - Liters to gallons');
WriteLn('10 - Ounces to grams',
     '22 - Grams to ounces');
WriteLn('11 - Pounds to kilograms',
     '23 - Kilograms to pounds');
WriteLn('12 - Fahrenheit to Celsius',
     '24 - Celsius to Fahrenheit');

FOR L := 1 TO 41 DO
BEGIN
   Write(' ');
END;

(* A 25th option is for quitting. *)
Write('25 - End the program');
PROCEDURE Convert_Qty (WhichOne:Integer) ;
{ All conversion-formulas are contained in this procedure, }
{ which also performs the calculations and the formatting }
{ of the results that are displayed on the Macintosh screen. }

VAR
  Quantity : Real ;
  Ch        : Char ;

BEGIN  { Convert_Qty }
  WriteLn ;
  GotoXY(16,21) ;
  Write('Enter the quantity to be converted: '); 
  ReadLn(Quantity) ;
  WriteLn ;
  Write('  
CASE (WhichOne) OF
  1 : WriteLn('','
               Quantity * 25.4:8:2,
               ' millimeters') ;
  2 : WriteLn('','
               Quantity * 0.0254:8:2,
               ' meters') ;
  3 : WriteLn('','
               Quantity * 0.3048:8:2,
               ' meters') ;
  4 : WriteLn('','
               Quantity * 1.6093:8:2,
               ' kilometers') ;
  5 : WriteLn('','
               Quantity * 5.0:8:2,
               ' milliliters') ;
  6 : WriteLn('','
               Quantity * 15:8:2,
               ' milliliters') ;
  7 : WriteLn('','
               Quantity * 0.4732:8:2,
               ' liters') ;
  8 : WriteLn('','
               Quantity * 0.9463:8:2,
               ' liters') ;
  9 : WriteLn('','
               Quantity * 3.7853:8:2,
               ' liters') ;
  END ; { of Convert_Qty }
10 : WriteLn('      ',
          Quantity * 28.3495:8:2, 
          ' grams'); 
11 : WriteLn('      ',
          Quantity * 0.453:8:2, 
          ' kilograms'); 
12 : WriteLn('      ',
          (Quantity - 32.0) * 5.0 / 9.0:8:2, 
          ' deg Celsius'); 
13 : WriteLn('      ',
          Quantity * 0.0394:8:2, 
          ' inches'); 
14 : WriteLn('      ',
          Quantity * 39.37:8 :2 , 
          ' inches'); 
15 : WriteLn('      ',
          Quantity * 3.2808:8:2, 
          ' feet'); 
16 : WriteLn('      ',
          Quantity * 0.6214:8:2, 
          ' miles'); 
17 : WriteLn('      ',
          Quantity * 0.2:8:2, 
          ' teaspoons'); 
18 : WriteLn('      ',
          Quantity * 0.0667:8:2, 
          ' tablespoons'); 
19 : WriteLn('      ',
          Quantity * 2.1134:8:2, 
          ' pints'); 
20 : WriteLn('      ',
          Quantity * 1.0567:8:2, 
          ' quarts'); 
21 : WriteLn('      ',
          Quantity * 0.2642:8:2, 
          ' gallons'); 
22 : WriteLn('      ',
          Quantity * 0.0353:8:2, 
          ' ounces'); 
23 : WriteLn('      ',
          Quantity * 2.2046:8:2, 
          ' pounds'); 
24 : WriteLn('      ',
          (Quantity * 9.0 / 5.0) + 32.0:8:2, 
          ' deg Fahrenheit'); 
25 : { DUMMY } 
END ; { of the CASE statement }

GotoXY(20,25) ;
BEGIN { Metrics }
SIGN_ON ; { Begin the sign on. }
REPEAT { Open a REPEAT...UNTIL loop. }
  CLEARSCREEN ;
  BIG_MENU ;
  GOTOXY(25,19) ; { Display the make-a-choice message. }
  WRITE('Choose 1 to 25: ') ;
  READLN(Choice) ; { Get the choice from the keyboard. }
IF (Choice < 1) OR (Choice > 25) THEN
  { If the choice is out of the range of 1 to 25... }
  BEGIN
    WRITELN ;
    GOTOXY(25,21) ; { ...display the error message. }
    WRITE('*** OUT OF RANGE ***') ;
    GOTOXY(20,23) ;
    WRITE('*** Press Return to continue *** ') ;
    READ(Ch) ;
  END
ELSE
  IF (Choice >= 1) OR (Choice < 25) THEN
    { If the number chosen is within the range, }
    { use the quantity-conversion formula. }
    CONVERT_QTY(Choice)
    { If the choice is 25, end the program. }
    UNTIL (Choice = 25) ;
END. { of Metrics }
Chapter 17

SORTER: Alphanumeric Sorting . . .
Garbage in, Neatness out

People often need to assemble a list of customer names or account numbers, payroll numbers, addresses, dates, parts numbers, zip codes, routing slips, salespersons’ hunt lists, friends’ names, association members’ card numbers, and other such lists. Alphanumeric information, when placed in an ordered sequence, becomes significantly more useful and usable than they were originally. SORTER eliminates the time-consuming and error-prone procedure of manually bringing order to such lists.

This program will sort and order up to 50 separate items, each up to 30 characters long, whether numeric or alphabetic. If an entry starts with a number and there are other items beginning with a letter, the items that start with a number are given priority in the sorting hierarchy.

The ASCII collating sequence prioritizes the digits “0” through “9,” the uppercase letters “A” through “Z,” and then lowercase “a” through “z.” This sequence is universally acceptable in developing a sort of a list of alphanumeric names and data. (DeLongine, for example, would appear ahead of deLongine, but not before DeLongine on a list of names sorted according to the ASCII collating sequence.)

The maximum number of lines that can be sorted by SORTER is established by the CONST statement “MaxNumInSort = 50 ;’. You may change the number 50 to a value you prefer.

Use a text editor that saves the file in ASCII form to create a file with the list of names (or numbers, or both) to be sorted. Give the file a distinctive name when you save it. When SORTER asks for the name of the file to be sorted, type the name of the file you created, press the Return key, and the program takes care of the rest. Be
sure to turn on the printer. The sorted list will be displayed on the video monitor and printed automatically and simultaneously.

The code for this sorting program is surprisingly simple. The value of such a program, when used as a stand-alone, is limited only by the individual’s need. For example, if you have a document that must be routed to a large number of people in an organization (I’ve seen routing lists with more than a 100 names) and you want to avoid the appearance of favoritism (or internal politics), the best way to list the names is in strict alphabetic sequence.

When you execute the program, the brief how-to-use instruction is displayed, and you are reminded to turn on the printer. The instructions also point out that the program sorts a file of characters you enter at the keyboard.

Sorting is performed in the SortBuffer procedure. The Procedure WriteBuffer then displays the sorted file on the screen and prints a hard copy. The printer does a line feed and carriage return. WriteLn(Printer) ensures that the printer’s buffer is emptied, and the program ends.

The sorting process, by some standards, may be slow. However, you should not be disturbed by the fact that SORTER takes just a few seconds to sort a 50-entry list; the task is still completed more quickly and more accurately than if done by hand. The program’s code is commented so that you can follow the sequences and actions of the procedures.

CODE LISTING FOR SORTER

(* Sorter reads unordered information from the *)
(* keyboard, sorts it, prints, and displays an *)
(* ordered, sorted list. *)

PROGRAM Sorter ;
uses PasPrinter ;

CONST
MaxNumInSort = 50 ; { Maximum number of lines to be sorted. }

TYPE
SortNum = 1..MaxNumInSort ;
StringType = String[80] ; { String length is 80 characters. }

VAR
Ch : Char ; { Keyboard response character. }
L : Integer ;
Buffer : ARRAY [SortNum] OF StringType ;
LineCnt : Integer ; { Counter for sorted lines. }

PROCEDURE SignOn ; { Display the signon banner. }
BEGIN   { SignOn }
GotoXY(15,5) ;
FOR L := 1 TO 50 DO
BEGIN
   Write(Chr(176)) ;
END ;
GotoXY(18,7) ;

Write('SORTLIST -- ALPHANUMERIC SORTING & PRINTING') ;
GotoXY(15,9) ;

FOR L := 1 TO 50 DO   { Do the next action 50 times.    }  
BEGIN               { Begin the FOR-loop action.      }  
   Write(Chr(176)) ;  { A character from the Macintosh set.  }  
END ;               { End the FOR loop-action.   }  

GotoXY(20,12) ;   { Display the instructions. }  
Write('This program automatically sorts ') ;
GotoXY(20,13) ;
Write('up to 50 lines of alphanumerics.') ;

GotoXY(18,16) ;
Write('!!! Be sure the printer is turned on !!!') ;

GotoXY(20,20) ;
Write('Press Return to continue... ') ;
Read(Ch) ;
ClearScreen ;
END ;   { of SignOn }  

PROCEDURE ReadUser ;  { ReadUser fills the buffer with keyboard info.  }  

VAR
   AllDone : Boolean ;

BEGIN   { ReadUser }
   AllDone := False ;   { Initialize the line counter to start at 1.  }
   LineCnt := 1 ;
GotoXY(20,2) ;
WriteLn('To QUIT entering information and start sorting, ') ;
GotoXY(20,3) ;
WriteLn('press Return at the beginning of any new line.') ;
WriteLn ;

WHILE ( NOT AllDone AND (LineCnt <= MaxNumInSort)) DO
BEGIN
   Write('Enter info ',LineCnt:2,':') ;
   Read(Ch) ;
   ClearScreen ;
   GotoXY(20,2) ;
   WriteLn('Press Return to continue... ') ;
   Read(Ch) ;
   ClearScreen ;
   GotoXY(20,2) ;
   WriteLn('To QUIT entering information and start sorting, ') ;
   Read(Ch) ;
   ClearScreen ;
   GotoXY(20,2) ;
   WriteLn('press Return at the beginning of any new line.') ;
   WriteLn ;
   LineCnt := LineCnt + 1 ;
END ;

WHILE ( NOT AllDone AND (LineCnt <= MaxNumInSort)) DO
BEGIN
   Write('Enter info ',LineCnt:2,':') ;
   Read(Ch) ;
   ClearScreen ;
   GotoXY(20,2) ;
   WriteLn('Press Return to continue... ') ;
   Read(Ch) ;
   ClearScreen ;
   GotoXY(20,2) ;
   WriteLn('To QUIT entering information and start sorting, ') ;
   Read(Ch) ;
   ClearScreen ;
   GotoXY(20,2) ;
   WriteLn('press Return at the beginning of any new line.') ;
   WriteLn ;
   LineCnt := LineCnt + 1 ;
END ;
ReadLn(Buffer[LineCnt])
AllDone := (Ord(Buffer[LineCnt][0]) = 0)
LineCnt := LineCnt + 1 ;  
   \(\text{Increment the line counter.}\) \end
END
LineCnt := LineCnt - 2
\end ; \text{( of ReadUser )}

PROCEDURE SortBuffer ; \text{( Swap-sort the buffer's contents. )}

VAR

HaveExchanged : Boolean
K : SortNum
Temp : StringType

BEGIN \text{( SortBuffer )}
REPEAT
HaveExchanged := False
FOR K := 2 TO LineCnt DO
   IF (Buffer[K - 1] > Buffer[K]) THEN
      BEGIN
         \text{( Do the exchange. )}
         HaveExchanged := TRUE ;
         Temp := Buffer[K] ;
         Buffer[K - 1] := Temp ;
      END ;
   UNTIL ( NOT HaveExchanged ) ;
END ; \text{( of SortBuffer )}

PROCEDURE WriteBuffer(VAR F:Text) ;

VAR

I : SortNum
J : Integer

BEGIN \text{( WriteBuffer )}
WriteLn(F, 'Listed in alphanumeric order: ')
WriteLn(F, '----------------------------------')
WriteLn(F)
FOR I := 1 TO LineCnt DO
   WriteLn(F, ',I:3:', 'Buffer[I]')
END ; \text{( of WriteBuffer )}

BEGIN \text{( This is the main section that calls the PROCEDURES. )}
SignOn
ReadUser ;

\text{150}
SortBuffer;
ClearScreen;
WriteBuffer(Output);  { Display the buffer's contents. }
WriteBuffer(Printer);  { Print the buffer' contents. }
WriteLn(Printer);
WriteLn(Printer,Chr(12));  { Eject the paper...a form feed. }
END. ( of Sorter )
They say, "When you appraise someone's job performance, you must be wary of the halo effect." The halo effect functions when the person appraising or evaluating another's performance is unreasonably influenced by superficial or extraneous aspects of the one being appraised. The aspects are not requirements for the job, and, therefore, have no place in the evaluation. Although the concept of a halo implies a positive effect, it can also cause a negative reaction. In either case, the appraiser may be responding subjectively to factors unrelated to the task at hand.

For example, suppose an evaluator enjoys a specific sense of humor and the person being evaluated demonstrates that brand of humor. It is conceivable the evaluator might, without conscious recognition, tend to overlook areas of performance on the job that are unrelated to the ability to entertain. The evaluator may give an undeserved high rating to the person being appraised. And, of course, the converse could be true; he could give an undeserved low rating to the person.

There are many ways to avoid or minimize the misleading halo effect. One of them is to depend entirely on a quantitative appraisal that can, with a properly created reference table, be converted to qualitative factors. Ideally, the appraiser is not aware of the algorithm that converts the quantitative to the qualitative.

This chapter's APPRAISAL program, written in Turbo Pascal for the Macintosh, is an attempt to minimize the halo effect by generating a quantitative and qualitative appraisal.

The program queries the appraiser about 11 job-related characteristics. The appraiser assigns a value from one through five (one is lowest, five is highest) to each characteristic. The program totals the individual scores and assigns a qualitative value
to the total score. The total score is valid if it is between 11 and 55; otherwise the appraiser has made an error, an error message is displayed, and the appraisal must be redone.

An option offers a detailed printout of the appraisal. The printed report is automatically dated and has space for the appraiser’s initials. APPRAISAL is not intended to solve all the problem areas inherent in the evaluation and management of human resources. It is intended to provide the Turbo Pascal user with source code that provides instruction, guidance, and practice with the language and the compiler.

The program signs on with requests for the information identifying the person being appraised. This is collected from the keyboard by the IDStuff procedure, and the instructions are displayed as the ShowHow procedure is called into action.

The next procedure, RateThePerson, is called, and the 11 characteristics are assigned values by keying in the appropriate assessment for the level of performance. Figure 18-1, a reproduction (Shift-Command-4) of the screen that appears shortly after the program is executed, displays the instructions and starts the appraisal for a fictitious “Jonathan C. Okleberry” of department #1465.

Error-trapping is provided. Warning messages are displayed for total scores that are less than 11 (11 * 1) or greater than 55 (11 * 5). If the total score is within the range of 11 to 55, the score is displayed. The screen continues with:

Print a hard copy for full details? (Y/N):

---

Enter the name of the person to be rated:    Jonathan C. Okleberry
Enter Jonathan C. Okleberry’s department number or name:   1465
Enter the date of this appraisal  (MM/DD/YY):    09/15/87

You will now appraise Jonathan C. Okleberry for 11 key characteristics. Each characteristic is to be assigned a value from ’1’ to ’5’.

‘1’ means Jonathan C. Okleberry performs Poorly.
‘2’ means Jonathan C. Okleberry just Barely Gets By.
‘4’ means Jonathan C. Okleberry delivers Good Performance.
‘5’ means Jonathan C. Okleberry does Exceptional Work.

You must rate each of the 11 characteristics in the range of ’1’ to ’5’.

After you have rated the 11th characteristic, all details of the appraisal can be printed.

A - QUALITY
Meets quality standards of the job:

---

Fig. 18-1. The screen that appears shortly after the Appraisal Program is run.
If the response is <Y>, the printer generates a more-comprehensive report with quantitative and qualitative details. An <N> response causes this query to be displayed:

Do you want to do another appraisal? (Y/N):

At this point, an <N> response ends the program.

SAMPLE REPORT PRODUCED BY APPRAISAL

+++++++++++++++++++++++++++++++++++++++++++++

Appraisal of: Jonathan C. Okleberry
Department: 1465
Appraised on: 09/15/87

+++++++++++++++++++++++++++++++++++++++++++++

Quality: 3
Knowledge: 3
Productivity: 4
Dependability: 3
Initiative: 2
Adaptability: 5
Attitude: 4
Attendance: 3
Safety: 2
Potential: 3
Personality: 4

The highest possible rating is 55.
The lowest possible rating is 11.
The mid-range rating is 33.
Jonathan C. Okleberry is rated at: 36

Jonathan C. Okleberry does an Acceptable Job.

+++++++++++++++++++++++++++++++++++++++++++++

Initials of the Appraiser: ____________

+++++++++++++++++++++++++++++++++++++++++++++

CODE LISTING FOR APPRAISAL

( Appraisal is for appraising, scoring, and quantifying )
( the work performance of a manager, supervisor, or an )
( employee on the basis of 11 characteristics. )
PROGRAM Appraisal;
uses PasPrinter;

TYPE
  StringType = String[80];

VAR
  Ch : Char;
  I,Adaptability,Attendance,Attitude : Integer;
  Dependability,Initiative,Knowledge : Integer;
  Personality,Potential,Quality : Integer;
  Quantity,Safety,Score : Integer;
  Date,Department,Name : StringType;

PROCEDURE SignOn;   ( Display the startup message. )
BEGIN   ( SignOn )
  GotoXY(15,8);   ( Send the cursor to column 15, row 8. )
  FOR I := 1 TO 50 DO   ( For 50 times, display the )
    Write(Chr(205));   ( graphics character, #205. )
  GotoXY(19,10);   ( Send the cursor to column 19, row 10. )
  WriteLn('APPRAISAL -- PERFORMANCE MERIT-RATING FORM');
  GotoXY(15,12);   ( Send the cursor to column 15, row 12. )
  FOR I := 1 TO 50 DO   ( For 50 times, display the )
    Write(Chr(205));   ( graphics character, #205. )
  GotoXY(15,18);
  Write('Press Return to start the appraisal procedures : ');
  ReadLn;
END;   ( of SignOn )

PROCEDURE Show_How;
{ Show the instructions for doing an appraisal. }
BEGIN
  ( Show_How )
  WriteLn('You will now appraise ',Name,' for 11 key characteristics. ');
  Write('Each characteristic is to be assigned ');
  WriteLn(' a value from ''1 to 5''. ');
  WriteLn;
  WriteLn(' 1'' means ',Name,' performs Poorly.' );
  WriteLn(' 2'' means ',Name,' just Barely Gets By.' );
  WriteLn(' 3'' means ',Name,' does Acceptable Work.' );
  WriteLn(' 4'' means ',Name,' delivers Good Performance.' );
  WriteLn(' 5'' means ',Name,' does Exceptional Work.' );
  WriteLn;
  Write('You must rate each of the 11 ');
  WriteLn(' characteristics in the range of ' 1'' to ' 5''. ' );
WriteLn;
WriteLn('After you have rated the 11th characteristic,');
WriteLn('all details of the appraisal can be printed.');
WriteLn;
WriteLn;
END;  { of Show_How }

PROCEDURE ID_Stuff ;
{ Collect identification information from the keyboard. }
BEGIN    { ID_Stuff }
  ClearScreen;
  GotoXY(1,5);
  Write('Enter the name of the person to be rated:   ') ;
  ReadLn(Name) ;
  Write('Enter ',Name,'''s department number or name:   ') ;
  ReadLn(Department) ;
  Write('Enter the date of this appraisal (MM/DD/YY):   ') ;
  ReadLn(Date) ;
  WriteLn;
  WriteLn;
END;  { of ID_Stuff }

PROCEDURE Print_A_Border (VAR OutFile:Text) ;

BEGIN    { Print_A_Border }
  FOR I := 1 TO 45 DO
    Write(Printer,'+') ;  { The border character is a plus sign. }
    WriteLn(Printer) ;  { Change it to suit your preferences. }
  END;  { of Print_A_Border }

PROCEDURE Initial_It ;  { Print a line for the appraiser's initials. }

BEGIN    { Initial_It }
  WriteLn(Printer);
  Print_A_Border(Printer) ;  { Call the PROCEDURE Print_A_Border. }
  WriteLn(Printer);
  WriteLn(Printer);
  WriteLn(Printer,'Initials of the Appraiser: ______________') ;
  WriteLn(Printer);
  Print_A_Border(Printer) ;  { Call the procedure. }
  Write(Printer,chr(12)) ;  { Eject paper to the top of the form. }
END;  { of Initial_It }

PROCEDURE Print_It ;  { Print the final report. }
VAR
  Sum : Real ;

BEGIN  { Print_It }
  Print_A_Border(Printer) ;   { Call the PROCEDURE }
                            { to print the border. }
                            { Then return here. }
  WriteLn(Printer) ;
  WriteLn(Printer, , Appraisal of: ,Name) ;
  WriteLn(Printer, , Department: ,Department) ;
  WriteLn(Printer, , Appraised on: ,Date) ;
  Print_A_Border(Printer) ;   { Print the border again. }
  WriteLn(Printer) ;
                            { Print the characteristics }
                            { and the scores. }
  WriteLn(Printer, , Quality: ,Quality) ;
  WriteLn(Printer, , Knowledge: ,Knowledge) ;
  WriteLn(Printer, , Productivity: ,Quantity) ;
  WriteLn(Printer, , Dependability: ,Dependability) ;
  WriteLn(Printer, , Initiative: ,Initiative) ;
  WriteLn(Printer, , Adaptability: ,Adaptability) ;
  WriteLn(Printer, , Attitude: ,Attitude) ;
  WriteLn(Printer, , Attendance: ,Attendance) ;
  WriteLn(Printer, , Safety: ,Safety) ;
  WriteLn(Printer, , Potential: ,Potential) ;
  WriteLn(Printer, , Personality: ,Personality) ;
  WriteLn(Printer) ;   { Print the ranges. }
  WriteLn(Printer, , The highest possible rating is 55.' ) ;
  WriteLn(Printer, , The lowest possible rating is 11.' ) ;
  WriteLn(Printer, , The mid-range rating is 33.' ) ;
  WriteLn(Printer, ,Name, is rated at: ,Score) ;
  WriteLn(Printer) ;

  ( Select and print a statement related to the Score. )
IF (Score <= 13) THEN
  WriteLn(Printer, ,Name, is a 'Poor' performer.' )
ELSE
  IF (Score <= 22) THEN
    WriteLn(Printer, ,Name, Just Gets By. Needs Help.' )
  ELSE
    IF (Score <= 43) THEN
      WriteLn(Printer, ,Name, does an Acceptable Job.' )
    ELSE
      IF (Score <= 49) THEN
        WriteLn(Printer, ,Name, is a Good Performer.' )
      ELSE
        WriteLn(Printer, ,Name, does Exceptional Work.' )
  END IF
END IF
END IF
END IF
END IF
END IF
END IF
END IF
END IF
{ Now, call the PROCEDURE for the signature line. }
Initial It
END ;  { of Print_It }

PROCEDURE Rate_The_Person ;  { Assign a value to each characteristic. }
BEGIN   { Rate_The_Person }
WriteLn('A - QUALITY') ;
Write(' Meets quality standards of the job: ') ;
ReadLn(Quality) ;
WriteLn ;
WriteLn('B - JOB KNOWLEDGE') ;
Write(' Understanding of all phases of the work: ') ;
ReadLn(Knowledge) ;
WriteLn ;
WriteLn('C - QUANTITY') ;
Write(' Level of productivity or output: ') ;
ReadLn(Quantity) ;
WriteLn ;
WriteLn('D - DEPENDABILITY') ;
Write(' Works according to instructions: ') ;
ReadLn(Dependability) ;
WriteLn ;
WriteLn('E - INITIATIVE') ;
Write(' Originates constructive actions: ') ;
ReadLn(Initiative) ;
WriteLn ;
WriteLn('F - ADAPTABILITY') ;
Write(' Ability to learn and adapt to changes: ') ;
ReadLn(Adaptability) ;
WriteLn ;
WriteLn('G - ATTITUDE') ;
Write(' Willingness to cooperate and meet demands: ') ;
ReadLn(Attitude) ;
WriteLn ;
WriteLn('H - ATTENDANCE') ;
Write(' Attendance and promptness on the job: ') ;
ReadLn(Attendance) ;
WriteLn ;
WriteLn('I - SAFETY AND ORDERLINESS') ;
Write(' Compliance with rules for safety/housekeeping: ') ;
ReadLn(Safety) ;

WriteLn ;
WriteLn('J - POTENTIAL FOR ADVANCEMENT') ;
Write(' Rate the person for leadership qualities: ') ;
ReadLn(Potential) ;

WriteLn ;
WriteLn('K - PERSONALITY') ;
Write(' Ability to get along with others: ') ;
ReadLn(Personality) ;

WriteLn
END ; { of Rate_The_Person }

BEGIN { main part of the Appraisal program }
SignOn ; { Execute the PROCEDURE. }
REPEAT
ID_Stuff ; { Execute the PROCEDURE. }
Show_How ; { Execute the PROCEDURE. }
Rate_The_Person ; { Execute the PROCEDURE. }

{ Assign the total value of the }
{ 11 characteristics to: Score. }
Score := Quality + Knowledge + Quantity +
Dependability + Initiative +
Adaptability + Attitude +
Attendance + Safety +
Potential + Personality ;

IF (Score < 11) THEN
BEGIN { This traps a scoring error and explains it. }
WriteLn ;
WriteLn('The score for ',Name,' is below 11.') ;
WriteLn ;
Write('An error has been made.') ;
WriteLn(' Please redo the appraisal.') ;
WriteLn ;
Write('Press Return to start over: ') ;
Read(Ch) ;
END
ELSE
IF (Score > SS) THEN
BEGIN { A similar trap with an explanation. }
WriteLn ;
WriteLn('The score for ',Name,' is above SS.') ;
WriteLn ;
Write('An error has been made.');
WriteLn('Please redo the appraisal.');
WriteLn;
Write('Press Return to start over: ');
Read(Ch);
END

ELSE

BEGIN  // If the score is in range, display the score. 
  WriteLn;
  WriteLn;
  WriteLn(Name,' has a score of ',Score);
  WriteLn;
  Write('Print a hard copy for full details? (Y/N): ');
  ReadLn(Ch);  // Get the keyboard's character.
END

IF Ch IN ['Y','y'] THEN
  // If the keyboard character is a 'Y' or a 'y', then BEGIN.
BEGIN
  ClearScreen;
  GotoXY(25,10);  // Display a message.
  WriteLn('*** Printing now being done ***');
  Print_It;
  WriteLn;
  WriteLn;
END
ELSE
  WriteLn;
  WriteLn;
  Write('Do you want to do another appraisal? (Y/N): ');
  ReadLn(Ch);  // Get the keyboard character.
END

UNTIL Ch IN ['N','n'];  // END on the option 'N' or 'n'.
END.  // of Appraisal
Chapter 19

BREAKEVEN: When Revenues and Costs Come Together

Race track aficionado:
“Well, I broke even today.
Just when I needed it!”

Break-even is the point at which an enterprise’s revenues and costs are exactly equal, and operating income is neither positive (gain) nor negative (loss). It can be assumed that, when operating below the break-even point, there is a loss. Conversely, when operating above the break-even point, there is a profit. A break-even point can be calculated for a company’s entire operations, or for a specific product or service.

Although reaching break-even is not the primary objective of a commercial enterprise, it is important for managers to be able to forecast and determine the factors that influence the operational position of the finite point known as break-even. The analysis of break-even can be used in reviewing past history, but it is most useful when applied to future periods as a guide to business planning, pricing, and developing cost standards.

The break-even point can be computed by using certain minimum data, which include: (1) total estimated or actual fixed costs that will be or have been incurred, without regard for quantity or plant capacity; (2) total estimated or actual costs that vary with the quantity of units produced; (3) the number of units that have been sold or are forecasted to be sold; and (4) the unit selling price.

The assumption is that all costs and revenues are for the same period of time. It is then possible to estimate the sales volume and selling price required to yield a specific,
targeted profit. Some managers prefer chart presentations for detailing break-even information. Tables of data with specific details and finite numbers, can provide a clearly defined profile of the facts essential to good planning.

Although break-even analysis can be a very important problem-solving tool, it should be used with caution. You must certainly take into account the fact that market and competitive conditions can erode selling prices or decrease the demand for consumable quantities of units, and that the costs of materials, labor, and overhead can rise through inflation. Also any combination of these factors can change, with one item rising and another falling to upset the neatly planned break-even point. The prudent problem solver periodically updates his break-even analysis.

The purpose of BREAKEVEN is to make the task of computing the break-even point for an endless series of costs and sales assumptions easy and rapid. Sample displays and printouts of the analyses that are accurately recalculated by entering changes in data assumptions can be generated in a few minutes.

EXECUTING THE BREAKEVEN PROGRAM

A sample execution of the compiled code demonstrates the capabilities of the program in dealing with assumptions made in a hypothetical medium-size project at a medium-size company. After the automatic sign-on, the screen displays the banner heading for the program and requests a series of data inputs. The responses at the keyboard are shown underlined to the right of the screen's queries.

The first procedure called in the main part of the program is SignOn, which clears the screen and displays the program's name and brief description. Next, the GetInfo procedure is called and the basic facts, the data on which the algorithms will operate, are collected.

Enter the TOTAL FIXED COST: 1000000
Now the VARIABLE COSTS per unit: 12345.67
Then the SELLING PRICE per unit: 23456.78

Starting Quantity for the table: 10
Ending Quantity for the table: 150
Increments of Quantity to show: 10

The screen clears, and the MakeReport procedure is called. First, titles for five columns of data are displayed at the top of the screen within a border of equal signs created by the BorderLine procedure:

<table>
<thead>
<tr>
<th>QTY</th>
<th>TOTAL COST</th>
<th>TOTAL SALES</th>
<th>GAIN/LOSS</th>
<th>UNIT COST</th>
</tr>
</thead>
</table>

The computed data is immediately displayed below the appropriate column headings. (These are shown in Fig. 19-1.) The screen then invites you to select an option:

Repeat the display? (Y/N):
### BREAKEVEN TABLE

<table>
<thead>
<tr>
<th>QTY</th>
<th>TOTAL COST</th>
<th>TOTAL SALES</th>
<th>GAIN/LOSS</th>
<th>UNIT COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1123456.75</td>
<td>234567.80</td>
<td>-888888.94</td>
<td>112345.67</td>
</tr>
<tr>
<td>20</td>
<td>1246913.38</td>
<td>469135.59</td>
<td>-777777.75</td>
<td>62345.67</td>
</tr>
<tr>
<td>30</td>
<td>1370370.12</td>
<td>703703.38</td>
<td>-666666.75</td>
<td>45679.00</td>
</tr>
<tr>
<td>40</td>
<td>1493826.75</td>
<td>938271.19</td>
<td>-555555.66</td>
<td>37345.67</td>
</tr>
<tr>
<td>50</td>
<td>1617283.50</td>
<td>1172839.00</td>
<td>-444444.50</td>
<td>32345.67</td>
</tr>
<tr>
<td>60</td>
<td>1740740.25</td>
<td>1407406.75</td>
<td>-333333.50</td>
<td>29012.34</td>
</tr>
<tr>
<td>70</td>
<td>1864196.88</td>
<td>1641974.50</td>
<td>-222222.38</td>
<td>26631.38</td>
</tr>
<tr>
<td>80</td>
<td>1987653.62</td>
<td>1876542.38</td>
<td>-111111.25</td>
<td>24845.67</td>
</tr>
<tr>
<td>90</td>
<td>2111110.50</td>
<td>2111110.25</td>
<td>0.00</td>
<td>23456.78</td>
</tr>
</tbody>
</table>

**Fig. 19-1.** BREAKEVEN generates an accurate and comprehensive table of data. The code offers good programming experience, and its output becomes a useful business tool.

If you type any letter but <N>, the screen clears and the display is repeated on the screen. The option to "Repeat the display" is shown repeatedly until you respond by entering the letter <N>. Then, a new option is displayed:

**For a printout, turn on the printer.**
**If "Yes," press 'Y.' (Any other key to quit):**

If you select any key other than <Y>, the program ends. If you select <Y>, the printer duplicates the table of data that is displayed on the screen. When the printout is finished, a form feed is sent to the printer and you are invited to:

**Do Another Printout? (Y/N)**
You can repeat the printouts, as you were able to repeat the data at the screen, as long as you respond by typing the letter <Y>.

**CODE LISTING FOR BREAKEVEN**

```pascal
{ BreakEven determines the point in operations }
{ where total costs equal total revenues, and }
{ there is neither profit nor loss. }

PROGRAM BreakEven;
uses PasPrinter;

TYPE
  StringType = STRING[80];

VAR
  { For the benefit of human eyes only, the program's } 
  { variables are listed in alphabetic order. The } 
  { compiler couldn't care less about the VAR order. }
  BreakEven : Real;
  Ch : Char;
  CostBrkEven : Real;
  EndQty : Integer;
  FixedCost : Real;
  IncrQty : Integer;
  L : Integer;
  ProfLoss : Real;
  RevBrkEven : Real;
  SellPrice : Real;
  StartQty : Integer;
  TotalCost : Real;
  TotalSales : Real;
  UnitCost : Real;
  VariableCost : Real;

PROCEDURE SignOn;
BEGIN
  { Display the startup message. }
  { Put the title in a fancy box. }
  ClearScreen;
  GoToXY(12,6);
  FOR L := 1 TO 56 DO
    BEGIN
      Write('$');
      END;
  GotoXY(12,7);
  Write('$$');
  { Set the cursor at column 12, row 6. }
  { Begin the FOR loop to draw a series }
  { of 56 dollar-signs. }
  { Draw the dollar-signs. }
  { End the FOR...DO loop. }
  { Position $ signs for the borders. }
END;
```
BEGIN { GetInfo }
  GotoXY(12,14);
  WriteLn('FIRST, THE BASIC FACTS ...');
  GotoXY(12,16);
  Write('Enter the TOTAL FIXED COST:       ');
  ReadLn(FixedCost);
  GotoXY(12,17);
  Write('Now the VARIABLE COSTS per unit:   ');
  ReadLn(VariableCost);
  GotoXY(12,18);
  Write('Then the SELLING PRICE per unit:    ');
  ReadLn(SellPrice);

  ClearScreen;
  GotoXY(12,4);
  WriteLn('THIS IS NEEDED FOR CALCULATIONS ...');
  GotoXY(12,6);
  Write('Starting Quantity for the table?   ');
  ReadLn(StartQty);
  GotoXY(12,7);
  Write('Ending Quantity for the table?     ');
  ReadLn(EndQty);
  GotoXY(12,8);
  Write('Increments of Quantity to show?    ');
  ReadLn(IncrQty);
  WriteLn;
  WriteLn

PROCEDURE GetInfo;  { Get the user’s data from the keyboard. }
PROCEDURE DisplayBorder ;  { Generate the "cosmetic" border line. }

BEGIN ( DisplayBorder )
    Write(' ');
    FOR L := 1 TO 68 DO
        BEGIN
            Write('=')  { The "border" is a line of "=" }
            END ;  { signs. It can be changed to }
                   { any printable character of }
                   { your choice. }
    WriteLn;
END ;  { of DisplayBorder }

PROCEDURE DisplayReport ;  { Display the breakeven table. }

VAR
    CurQuan : Integer ;

BEGIN ( DisplayReport )
    ClearScreen ;
    DisplayBorder ;  { Call the DisplayBorder PROCEDURE. }
    WriteLn ;
    Write(' ');
    WriteLn('BREAKEVEN TABLE') ;
    WriteLn ;
    DisplayBorder ;
    Write(' QTY TOTAL COST TOTAL SALES') ;
    WriteLn(' GAIN/LOSS UNIT COST') ;
    DisplayBorder ;
    WriteLn ;

    { Assign values to the variables for the computations. }
    BreakEven := FixedCost / (SellPrice - VariableCost) ;
    RevBrkEven := SellPrice * BreakEven ;
    CostBrkEven := FixedCost + (VariableCost * BreakEven) ;
    { Now perform the computations and present the results. }
    CurQuan := StartQty ;

    WHILE (CurQuan <= EndQty) DO
    BEGIN
        { For simplicity, combine calculations and }
        { assign new names to the new values. }
        TotalSales := SellPrice * CurQuan ;
        TotalCost := FixedCost + (VariableCost * CurQuan) ;
UnitCost := TotalCost / CurQuan;
ProfLoss := TotalSales - TotalCost;
WriteLn( ' ,
   CurQuan:7,' ,
   TotalCost:12:2,' ,
   TotalSales:12:2,' ,
   ProfLoss:12:2,' ,
   UnitCost:12:2); 
CurQuan := CurQuan + IncrQty
END;

WriteLn ;
DisplayBorder ;
CurQuan := Trunc(BreakEven); { Supress the decimal point. }
WriteLn( ' ,
   CurQuan:7,' ,
   CostBrkEven:12:2,' ,
   RevBrkEven:12:2,
   '=' BREAKEVEN POINT');
DisplayBorder ;
WriteLn ; { Write the last lines of data. }
WriteLn( ' TOTAL FIXED COST: ',FixedCost:12:2); 
WriteLn( ' VARIABLE COSTS PER UNIT: ',VariableCost:12:2);
WriteLn( ' SELLING PRICE PER UNIT: ',SellPrice:12:2);
WriteLn ;
DisplayBorder ;
END ; { of DisplayReport }

PROCEDURE PrintBorder ; { Print the "cosmetic" border line. }
BEGIN ( PrintBorder )
Write(Printer,' '); 
FOR L := 1 TO 68 DO 
BEGIN
   Write(Printer,'=') { The "border" is a line of "-" }
END ; { signs. It can be changed to }
   { any printable character of }
   { your choice. }
WriteLn(Printer);
END ; { of PrintBorder }

PROCEDURE PrintReport ; { Print the breakeven table. }

VAR
   CurQuan : Integer ;
BEGIN { PrintReport }
PrintBorder;   { Call the PrintBorder PROCEDURE. }
WriteLn(Printer);
Write(Printer, 'QTY TOTAL COST TOTAL SALES');
WriteLn(Printer);
Write(Printer, 'GAIN/LOSS UNIT COST');
PrintBorder;
Write(Printer, 'BREAKEVEN TABLE');
WriteLn(Printer);

{ Assign values to the variables for the computations. }
BreakEven := FixedCost / (SellPrice - VariableCost);
RevBrkEven := SellPrice * BreakEven;
CostBrkEven := FixedCost + (VariableCost * BreakEven);

{ Now perform the computations and present the results. }
CurQuan := StartQty;

WHILE (CurQuan <= EndQty) DO
BEGIN
  { For simplicity, combine calculations and assign new names to the new values. }
  TotalSales := SellPrice * CurQuan;
  TotalCost := FixedCost + (VariableCost * CurQuan);
  UnitCost := TotalCost / CurQuan;
  ProfLoss := TotalSales - TotalCost;
  WriteLn(Printer, CurQuan:7, TotalCost:12:2, TotalSales:12:2, ProfLoss:12:2, UnitCost:12:2);
  CurQuan := CurQuan + IncrQty;
END;

WriteLn(Printer);
PrintBorder;
CurQuan := Trunc(BreakEven); { Suppress the decimal point. }
WriteLn(Printer, CurQuan:7, CostBrkEven:12:2, RevBrkEven:12:2, '
  ' = BREAKEVEN POINT);

PrintBorder;
WriteLn(Printer);  { Write the last lines of data. }
WriteLn(Printer, TOTAL FIXED COST: FixedCost:12:2);
WriteLn(Printer, VARIABLE COSTS PER UNIT: VariableCost:12:2);
WriteLn(Printer,'SELLING PRICE PER UNIT: ',SellPrice:12:2);
WriteLn(Printer);
PrintBorder;
WriteLn(Printer,Char(12)); { Send a formfeed code to the printer. }
WriteLn(Char(7)); { Beep the Macintosh. }
END; { of PrintReport }

BEGIN { BreakEven } { the main part of the program. }
  SignOn;
  GetInfo;
  REPEAT
    DisplayReport; { Prepare and display the report. }
    WriteLn;
    Write(' Repeat the display? (Y/N): '); { An option. }
    ReadLn(Ch); { Get the response from the keyboard. }
    UNTIL Ch IN [ 'N', 'n' ]; { This is a short-form response. }
    WriteLn;
    WriteLn;
    WriteLn(' (For a printout, turn on the printer.)');
    Write(' If "Yes," press 'Y.' (Return key to quit): ');
    ReadLn(Ch);
  WHILE Ch IN [ 'Y', 'y' ] DO
    BEGIN
      PrintReport; { Print the report. }
      WriteLn;
      Write(' Do another printout? (Y/N): ');
      ReadLn(Ch);
    END;
  END. { of BreakEven }
Every person accountable for the sales activities of a group of manufacturers' representatives (or other independent sales agencies) knows the importance of having up-to-date records for each of his sales representatives or agents. At a minimum, the records must provide rapid access to data for sales volume, commission rates, how much has been earned in commissions, what part of the earned commissions have been paid out, and what amounts are still unpaid.

Such records are maintained in many ways. Frequently they are manual log books that depend on cross-references to other internal records. The program in this chapter provides a workable solution to the problem of keeping records up-to-date and close-at-hand for immediate reference, review, and discussion. It is especially useful when a sales rep calls to gripe about the delay in his commission payment. REPCOMMISSION provides a ready design for a sales and commission record that can be prepared and printed in a few minutes.

After the automatic sign on is displayed, the four variables for Total_Sales, Total_Earned, Total_Paid, and Total_Owed are initialized to zero to prevent "garbage" in the calculations. The program asks for the date of the report and then prints it with a header for the printed page. Three lines of instructions are displayed:

Enter the sales volume for each rep when prompted. When all entries are completed, type -1 in response to the next request for "Sales Volume."
Your responses to a series of queries supplies the data for the Macintosh's calculations:

Enter the sales volume for rep 1: 123542.76
What is the commission % - rate for rep 1: 10

At this point the screen fills in the data:

Commissions earned: 12354.28

Then another request follows:

What commissions have been paid to rep 1: 12000

The computer does a rapid calculation and displays:

Commissions still due: 354.28

The WriteReport procedure is called and sends the information just entered to the printer. Totals are accumulated in the main part of the program. The program moves on to the next Rep and asks the same series of questions. Entries at the keyboard are shown at the right, underlined:

Enter the sales volume for rep 2: 27555.50
What is the commission % - rate for rep 2: 7.5
Commissions earned: 2066.66
What commissions have been paid to rep 2: 2066.66
Commissions still due: 0.00

The above data are sent to the printer, totals are accumulated, and the program continues:

Enter the sales volume for rep 3: 49500
What is the commission % - rate for rep 23: 5
Commissions earned: 2475.00
What commissions have been paid to rep 3: 1500
Commissions still due: 975.00

To end the entries, follow the instructions by entering -1 for the sales volume.

Enter the sales volume for rep 4: -1

The WrapUp procedure is called twice, once to display the totals on the screen, and again to send the information to the printer, which completes the program's execution.
A SAMPLE REP COMMISSION PRINTOUT

Figure 20-1 is the report generated by the printer for Reps one through three using data from the example. The report could have continued for as many as 25 Reps, according to the CONSTANT value assigned to MaxNumReps at the start of the code; and, of course, you can easily change the constant value for the maximum number.

<table>
<thead>
<tr>
<th>Rep 1</th>
<th>Rep 2</th>
<th>Rep 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales volume:</td>
<td>123452.76</td>
<td>27555.50</td>
</tr>
<tr>
<td>Commission rate: 10.00%</td>
<td>12345.28</td>
<td>2066.66</td>
</tr>
<tr>
<td>Commissions earned:</td>
<td>12000.00</td>
<td>2066.66</td>
</tr>
<tr>
<td>Commissions paid:</td>
<td>345.28</td>
<td>0.00</td>
</tr>
<tr>
<td>Commissions unpaid:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total sales for all reps: 200508.25
Total commissions earned: 16886.94
Total commissions paid: 15566.66
Total commissions unpaid: 1320.28

Fig. 20-1. A report generated by a sample run of REP COMMISSION.
CODE LISTING FOR REPCOMMISSION

{ Generate reports of reps' sales volume, commissions earned } 
{ commissions paid, commissions still owed, and a summary. } 

PROGRAM RepCommission ; 
uses PasPrinter ;

CONST 
MaxNumReps 25 ; { Limit the number of reps to 25, or } 
{ revise MaxNumReps = n, if desired. } 

TYPE 
Rep_NumberType = 1..MaxNumReps ; 
StringType = String[80] ;

VAR 
Ch : Char ; 
L : Integer ; 
Date : StringType ; 
Rep_Number : Rep_NumberType ; 
Commissions_Owed : Real ; 
Commissions_Earned : Real ; 
Commissions_Paid : Real ; 
Commission_Rate : Real ; 
Reps : ARRAY [Rep_NumberType] OF Real ; 
Sales_Volume : Real ; 
Total_Owed : Real ; 
Total_Earned : Real ; 
Total_Paid : Real ; 
Total_Sales : Real ;

PROCEDURE SignOn ; { Create and display the startup banner. } 
BEGIN ( SignOn ) 
GotoXY(17,8) ;

FOR L := 1 TO 6 DO { Do six iterations in } 
Write('REPCOMM') ; { a row of 'REPCOMM'. } 
GotoXY(17,9) ; 
Write('R') ; { Let's get fancy and } 
GotoXY(58,9) ; { start a column of letters } 
Write('R') ; { for the border design. } 
GotoXY(17,10) ; 
Write('E') ; 
GotoXY(58,10) ; 
Write('E') ;
GotoXY(25,10);
Write('SALES COMMISSION REPORTER'); { Title. }

GotoXY(17,11);
Write('P');
GotoXY(58,11);
Write('P');
GotoXY(17,12);

FOR L := 1 TO 6 DO { Same as above. }
  Write('REPCOMM');

GotoXY(17,18);
WriteLn(':::::: Turn on the printer, please :::::');
GotoXY(17,20);
Write(':::::: Press Return to continue :::::');
Read(Ch);
END; { of SignOn }

PROCEDURE Divider_Line(VAR Destination:Text); 
BEGIN { Divider_Line }
  FOR L := 1 TO 40 DO { The quantity is 40 for a ... }
    BEGIN
      Write(Destination,'*'); { ... line of asterisks or any }
      END; { any printable character. }
  END; { of Divider_Line }

PROCEDURE WrapUp(VAR Destination:Text); { Do the summary. }
BEGIN { WrapUp }
  Divider_Line(Destination); { Call the PROCEDURE. }
  WriteLn(Destination);
  WriteLn(Destination,'Total sales for all reps: ',
             Total_Sales:12:2);
  WriteLn(Destination,'Total commissions earned: ',
             Total_Earned:12:2);
  WriteLn(Destination,'Total commissions paid: ',
             Total_Paid:12:2);
  WriteLn(Destination,'Total commissions unpaid: ',
             Total_Owed:12:2);
  Divider_Line(Destination);
  WriteLn(Destination);
END; { of WrapUp }

PROCEDURE WriteRepReport; { Send reps' statistics to the printer. }

177
BEGIN { WriteRepReport }
  WriteLn(Printer);
  WriteLn(Printer,
          '---------- Report for rep ',Rep_Number:2,' ----------');
  WriteLn(Printer);
  WriteLn(Printer, 'Sales volume: ',
              Sales_Volume:12:2);
  WriteLn(Printer, 'Commission rate: ',
              Commission_Rate:5:2,'%');

  { The value of Commissions_Earned is established }
  { by the formula given in the main block. }
  WriteLn(Printer, 'Commissions paid: ',
            Commissions_Paid:12:2);
  WriteLn(Printer, 'Commissions unpaid: ',
            Commissions_Owed:12:2);
  WriteLn(Printer);
END; { of WriteRepReport }

BEGIN { of main }
  SignOn;
  Total_Sales := 0; { Initialize the Totals to start at zero. }
  Total_Earned := 0;
  Total_Paid := 0;
  Total_Owed := 0;

  ClearScreen;
  Write('Date of this Report (MM/DD/YY): ');
  ReadLn(Date);

  Divider_Line(Printer); { Execute the PROCEDURE. }
  WriteLn(Printer);
  WriteLn(Printer);
  WriteLn(Printer, 'COMMISSIONS REPORT AS OF: ',Date);
  WriteLn(Printer);
  Divider_Line(Printer); { Execute the PROCEDURE. }
  WriteLn(Printer);
  WriteLn(Printer);
  WriteLn;
  WriteLn;
  WriteLn('Enter the sales volume for each rep when prompted. ');
  WriteLn('When all entries are completed, type -1 in ');
  WriteLn('response to the next request for "Sales Volume". ');
  WriteLn;
  WriteLn;
  Rep_Number := 1; { Set the value of Rep_Number to 1. }

  REPEAT
    Write('Enter the Sales Volume for rep ',Rep_Number,': ');

REPEAT
ReadLn(Sales_Volume);

IF (Sales_Volume > -1) THEN
   { Sales_Volume of -1 ends the input of data. }
   { If Sales_Volume is greater than -1, BEGIN. }
BEGIN
   { Collect data, calculate, assign, and display values. }
   Write('What is the commission %-rate for rep ','
         Rep_Number:2','': ');
   ReadLn(Commission_Rate);

   Commissions_Earned := Sales_Volume * (Commission_Rate / 100.0);
   WriteLn('Commissions earned: ',
            Commissions_Earned:12:2);
   Write('What commissions have been paid to rep ','
         Rep_Number:2','': ');
   ReadLn(Commissions_Paid);

   Commissions_Owed := Commissions_Earned - Commissions_Paid;
   WriteLn('Commissions still due: ',
            Commissions_Owed:12:2);
   WriteLn;
   WriteRepReport;
   Total_Sales := Total_Sales + Sales_Volume;
   Total_Earned := Total_Earned + Commissions_Earned;
   Total_Paid := Total_Paid + Commissions_Paid;
   Total_Owed := Total_Owed + Commissions_Owed;
   Rep_Number := Rep_Number + 1;
END;
UNTIL (Sales_Volume = -1); { REPEAT...UNTIL Sales_Volume is -1. }
WriteLn;
WrapUp(Output); { Execute the PROCEDURE; send to the video display. }
WrapUp(Printer); { Execute the PROCEDURE; send to the printer. }
WriteLn;
Write('=-> Press Return to complete the report <== ');
Read(Ch);
Write(Printer,Chr(12)); { The old familiar formfeed. }
END. { of main }
Chapter 21

SALESREPORT: Analyzing Sales Performance

The program SALESREPORT demonstrates the use of arrays and number crunching. At the same time it provides a program that can either be used in the form presented here or revised, customized, and modified to fit special needs.

The program is a "sales performance analyzer." It compares the performances of individuals against the average level, reports the amount of dollars by which each is above or below average, and generates a printed, detailed report for further study. The program's name and purpose can easily be changed in the source code given here to refer to any kind of group and individual achievement in business, social, community, and sports events.

SALESREPORT is highly versatile and exceptionally useful for the sales, marketing, financial, and senior managers of any enterprise. This program can be used to obtain a visual display and a hard-copy printout of sales data related to a variety of sources that you identify by your own confidential code number. The sources could be: (1) individual salespeople within a sales section; (2) sales districts; (3) sales regions and sales areas; (4) retailers, wholesalers, or dealers and distributors; (5) customers, users, national accounts, house accounts; (6) industries, markets, and products; (7) divisions and subsidiaries; or (8) individual or groups of products and services.

Sounds like a lot of power—and it is. What's more, it all fits into a compact Turbo Pascal program. The program signs on, and the Starting Values Procedure asks for data.

Let's simulate a run. The regular type represents the requests for data that are displayed on the Macintosh screen. The underlined type at the right represents your responses entered at the keyboard:
WHAT PERIOD OF TIME DOES THIS ANALYSIS COVER?
(Please enter all dates in the format MM/DD/YY):
Enter the Start Date: 10/01/87
Enter the End Date: 10/31/87
Enter Today's Date: 11/15/87

ENTER THE SALES AMOUNT FOR EACH ID NUMBER.
(To end the program, type -1 for 'Sales Dollars."

Sales dollars for ID Number 1: 1301.36
Sales dollars for ID Number 2: 1333.52
Sales dollars for ID Number 3: 1344.57
Sales dollars for ID Number 4: 1355.89
Sales dollars for ID Number 5: 1426.50
Sales dollars for ID Number 6: 1223.35
Sales dollars for ID Number 7: 1223.01
Sales dollars for ID Number 8: 1546.58
Sales dollars for ID Number 9: 1555.58
Sales dollars for ID Number 10: 1565.68
Sales dollars for ID Number 11: 1213.24
Sales dollars for ID Number 12: 1267.89
Sales dollars for ID Number 13: -1

After the -1 is entered, the screen clears, the computations are made instantly by the Macintosh, and the results appear on the screen, called by the Make Report Procedure. At the bottom of the display you are asked:

Printout? Turn on the printer.
If YES, type 'Y' (Any other key to quit):

The actual printout is reproduced in Fig. 21-1. Next you are asked whether or not you want to repeat the printout. Any response other than <Y> ends the program.

A SAMPLE PRINTOUT OF SALESREPORT

<table>
<thead>
<tr>
<th><strong>SALES PERFORMANCE ANALYSIS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>PERIOD COVERED BY THIS REPORT: 10/01/87 to 10/31/87</td>
</tr>
<tr>
<td>DATE THIS REPORT WAS PREPARED: 11/15/87</td>
</tr>
<tr>
<td>TOTAL SALES VOLUME: 16357.17</td>
</tr>
<tr>
<td>AVERAGE SALES VOLUME: 1363.10</td>
</tr>
</tbody>
</table>

Fig. 21-1. The printout generated by SALESREPORT saves much time and minimizes human error.
<table>
<thead>
<tr>
<th>IDENTIFIER #</th>
<th>SALES $</th>
<th>$ ABOVE AVGE</th>
<th>$ BELOW AVGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1301.36</td>
<td></td>
<td>61.74</td>
</tr>
<tr>
<td>2</td>
<td>1333.52</td>
<td></td>
<td>29.58</td>
</tr>
<tr>
<td>3</td>
<td>1344.57</td>
<td></td>
<td>18.53</td>
</tr>
<tr>
<td>4</td>
<td>1355.89</td>
<td></td>
<td>7.21</td>
</tr>
<tr>
<td>5</td>
<td>1426.50</td>
<td>63.40</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1223.35</td>
<td></td>
<td>139.75</td>
</tr>
<tr>
<td>7</td>
<td>1223.01</td>
<td></td>
<td>140.09</td>
</tr>
<tr>
<td>8</td>
<td>1546.58</td>
<td>183.48</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1555.58</td>
<td>192.48</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1565.68</td>
<td>202.58</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1213.24</td>
<td></td>
<td>149.86</td>
</tr>
<tr>
<td>12</td>
<td>1267.89</td>
<td></td>
<td>95.21</td>
</tr>
</tbody>
</table>

**CODE LISTING FOR SALESREPORT**

( Generate a report of individual and group sales performance. )
( Calculate total, average, above, and below average levels. )
( The name and code may be revised to fit other than sales. )

PROGRAM SalesReport;
uses PasPrinter;

CONST
  { Establish the maximum number of entries }
  { at the desired number + 1. }    
Max_Num_Id = 51;

TYPE
  Id_Num_Type = 1..Max_Num_Id;
  String_Type = STRING[80];

VAR
  Avg_Sales : Real;
  Ch         : Char;
  End_Date   : String_Type;
  Id_Number  : Id_Num_Type;
  Sales_Amt  : ARRAY [Id_Num_Type] OF Real;
  Start_Date : String_Type;
  Today_Date : String_Type;
  Total_Sales: Real;

PROCEDURE Sign_On;    { Display the sign on message. }
  { Create a border of asterisks. }
BEGIN { Sign_On }
GotoXY(15,8);
Write('**************************************************');
GotoXY(15,9);
Write('*');
GotoXY(15,10);
Write('* SALES REPORT -- Sales Performance Analyzer *');
GotoXY(15,11);
Write('*');
GotoXY(15,12);
Write('**************************************************');
GotoXY(15,16);
Write('Press the Return key to start the processes... ');
Read(Ch);
END; { of Sign_On }

PROCEDURE Draw_A_Line(VAR F:Text); { Draw a line of asterisks. }
BEGIN { Draw_A_Line }
{ Note: the next two lines display and print on one line. }
Write(F,' ***********************************');
WriteLn(F,'***********************************');
END; { of Draw_A_Line }

PROCEDURE Starting_Values ; { Collect data for initial calculations. }
BEGIN { Starting_Values }
GotoXY(1,5);
WriteLn(' WHAT PERIOD OF TIME DOES THIS ANALYSIS COVER?');
WriteLn(' (Please Enter All Dates In The Format MM/DD/YY.)');
GotoXY(1,8);
Write(' Enter the Start Date: ');
ReadLn(Start_Date);
Write(' Enter the End Date: ');
ReadLn(End_Date);
Write(' Enter Today's Date: ');
ReadLn(Today_Date);
GotoXY(1,12);
Write(' ENTER THE SALES AMOUNT FOR EACH ID NUMBER. ');
GotoXY(1,13);
Write(' (To end the program, type -1 for 'Sales Dollars.')');
WriteLn;
WriteLn;
Id_Number := 1; { Initialize the Id_Number count to 1. } 
Total_Sales := 0; { Initialize Total_Sales to zero dollars. }
REPEAT
Write(' Sales Dollars for ID Number ',Id_Number,': ');
ReadLn(Sales_Amt[Id_Number]);
{ Restore LowVideo for the next query. }

IF (Sales_Amt[Id_Number] > -1) THEN
BEGIN
  Total_Sales := Total_Sales + Sales_Amt[Id_Number];
  Id_Number := Id_Number + 1;
END;

UNTIL ((Sales_Amt[Id_Number] = -1) OR (Id_Number = Max_Num_Id)) ;

Id_Number := Id_Number - 1;
Avg_Sales := Total_Sales / Id_Number; { Straightforward math. }
END; { of Starting_Values }

PROCEDURE Make_Report(VAR F:Text); { Create the sales report. }

VAR
NumInRpt : Id_Num_Type;

BEGIN { Make_Report }
Draw_A_Line(F);
WriteLn(F);
WriteLn(F, ' SALES PERFORMANCE ANALYSIS');
WriteLn(F);
Draw_A_Line(F);
WriteLn(F, ' PERIOD COVERED BY THIS REPORT: ',Start_Date,
  ' to ',End_Date);
WriteLn(F, ' DATE THIS REPORT WAS PREPARED: ',Today_Date);
WriteLn(F);
WriteLn(F, ' TOTAL SALES VOLUME: ',Total_Sales:10:2);
WriteLn(F, ' AVERAGE SALES VOLUME: ',Avg_Sales:10:2);
Draw_A_Line(F);
WriteLn(F, ' IDENTIFIER # SALES $ ',
  '$ ABOVE AVGE $ BELOW AVGE');
Draw_A_Line(F);
FOR NumInRpt := 1 TO Id_Number DO
BEGIN
Write(F, ',NumInRpt:3,',
  Sales_Amt[NumInRpt]:10:2);

IF (Sales_Amt[NumInRpt] > Avg_Sales) THEN
  WriteLn(F, ',
    Sales_Amt[NumInRpt] - Avg_Sales:10:2)
ELSE
BEGIN  { SalesReport...main section }
  Sign_On;    { Execute the PROCEDURES in the order used. }
  ClearScreen;
  Starting_Values;
  ClearScreen;
  Make_Report(Output);
  WriteLn;
  WriteLn;
  WriteLn(' Printout? Turn on the printer.');
  Write(' If YES, type 'Y''. (Press the Return key to quit): ');
  Read(Ch);
  WHILE (Ch = 'Y') OR (Ch = 'y') DO 
    BEGIN 
      Make_Report(Printer);   { Print the final report. }
      Write(Printer,Chr(12)); { Eject the paper. }
      WriteLn;
      WriteLn;
      WriteLn(' Repeat the printout ('Y'' or 'N''),'? ');
      Write(' If YES, type 'Y''. (Press the Return key to quit): ');
      Read(Ch);
    END;
  END;  { of SalesReport }

  WriteLn(F,
    'Avg_Sales - Sales_Amt[NumInRpt]:10:2');
  END ;
  Draw_A_Line(F);
END;  { of Make_Report }

186
Chapter 22

MINICALC: A Four-Function Calculator

This program, MINICALC, may never replace any of the hand-held calculators one can buy for less than $5. I'm not, however, trying to replace anything. I'm just writing code to demonstrate how to get things done using Turbo Pascal.

MINICALC is a short program, but it demonstrates how simple it is to do basic calculations within a few dozen lines of code. The operators: + (plus sign for addition), - (minus sign for subtraction), * (asterisk for multiplication), and / (forward slash for division) are used in this program within a group of IF...THEN statements that recognize the four operators.

The program signs on and displays one line of instructions, which tells you that you may use any of the four operators, + - * / . Then, you are told to enter the first number followed by a <CR>, which is the Return key. “A” is assigned the value of the first number. Next, you enter the operator, followed by a <CR> - . Finally, you enter the second number followed by a <CR>. “B” is assigned the value of the second number.

The IF...THEN clauses test for the operator you entered. The program drops through each of the clauses, but stops and takes the appropriate action when it reaches the appropriate operator, which is the one you had selected. “C” is assigned the value of the result of “A” and “B” after they have been affected by or acted upon by the operator.

A, B, and C have been declared to be variables of the type real. The statement “WriteLn(C:10:16)” causes the screen to display the calculated result, which is the value that has been assigned to “C” formatted as a real number with 10 integers to the left, and 16 places to the right of the decimal point.
Finally, the program asks "Do you want to do another calculation? Y)es or N)o." If you enter any character other than "N" (in either uppercase or lowercase), the program goes back to the line below the REPEAT statement and begins the process all over again. If you enter a lowercase "n," the statement "UNTIL (Ch) IN ['N', 'n']" converts it to an uppercase "N" and the program ends.

You can expand the utility of the MINICALC program by adding code for operators that use some of the mathematical functions that are built into Turbo Pascal and are listed in Appendix D of this book.

CODE LISTING FOR MINICALC

(* A four-operator two-number calculator. *)
(* Precision is real numbers to 16 places. *)

PROGRAM MiniCalc;

VAR
  A, B, C : Real;
  Ch : Char;
  Operator : Char;

BEGIN MiniCalc...this is the entire program. }
{ Separate procedures are not used. }
REPEAT { Repeat until the UNTIL condition is met. }
ClearScreen;
GotoXY(5,5);
Write(‘THIS PROGRAM TURNS YOUR EXPENSIVE COMPUTER’);
WriteLn(‘ INTO AN INEXPENSIVE CALCULATOR!’);
GotoXY(5,8); { Start the instructions. }
Write(‘The math operators you may use are: ’);
WriteLn(‘+ * /’); { End the instructions. }
GotoXY(5,10);
Write(‘Enter the first number and press Return: ’);
ReadLn(A);
GotoXY(5,12);
Write(‘Now enter the operator and press Return: ’);
ReadLn(Operator);
GotoXY(5,14);
Write(‘Now the second number and press Return: ’);
ReadLn(B);
{ Test for the appropriate operator. }
{ Assign a value to C, the result. }
IF Operator = '+' THEN
  C := A + B ;
IF Operator = '-' THEN
  C := A - B ;
IF Operator = '*' THEN
  C := A * B ;
IF Operator = '/' THEN
  C := A / B ;

GotoXY(5,16) ;
Write('The calculated result is: ') ;
WriteLn(C:10:16); { Format the response to the 16th decimal. }

GotoXY(5,18) ;
Write('Do you want to do another calculation? Y)es or N)o: ') ;
ReadLn(Ch) ;
UNTIL (Ch) IN ['N','n'] ; { "N" or "n" satisfies the UNTIL; quit. } END.
Chapter 23

PRINTERTEST1 & PRINTERTEST2: Programs That Will Test and Benchmark Your Printer’s Printing Quality

Ever thought of benchmarking the printing quality of your favorite printer? It’s easy, if you have a program that puts the printer through its ASCII-character printing paces. This chapter offers two versions in TURBO Pascal of a program that will do just that. In addition, if the printer you are using has the capability, the programs can be modified to print the extended character or graphics set of characters above ASCII decimal 127.

The first, PRINTERTEST1, consists of a series of eight procedures. The first procedure, Headings, prints headings for the columns of sample printing. The other seven procedures are ASCII groups of printable characters that range from decimal 33 through decimal 126. The procedures are named to provide easy identification of the nature of the group of ASCII characters each procedure deals with. For example, it is obvious that the procedure Numbers prints the numbers in the ASCII table, and the procedure Upper_Case_Alpha deals with the capital or uppercase letters of the alphabet.

The constant, LineLen, is declared to be equal to 20. Each of the procedure’s FOR . . . DO clauses contains the LineLen constant. Therefore, the length of each line printed is 20 characters. This is a good illustration of one utility of the CONST declaration. The line length can be declared individually in each procedure; however, if you should want to adjust the length (or the width) of the character-printing line, you would have to change it in each of the seven procedures. By declaring it as a CONSTANT at the start of the program, all you have to do is change the value (20 in this case) to whatever reasonable length you desire. In this way each procedure is automatically adjusted, adopting the value declared as the constant.

PRINTERTEST2, the second version of the print-test program, consists of the
main program without calls to separate procedures. The CONstant and the VARiables
are the same in both programs, serve the same purposes, and are treated in the same
manner.

From an operational standpoint, the programs produce identical results. Both pro-
grams start by displaying a one-line message announcing the start of the tests. The
printer, which must be connected and turned on at the time, starts its job immediately.
As soon as the printer is finished and the program's output to the printer has been
dumped to the printer's buffer, the one-line message announcing that the tests are fin-
ished overwrites the starting message.

Note in the listings of the code for PRINTERTEST1 and PRINTERTEST2 which
of the two approaches you find easier for eyeball reading. Clearly, PRINTERTEST1
is easier. It also is easier to modify or maintain, which is one of the purposes of using
groups of procedures in writing code in a structured language.

To illustrate the action, Fig. 23-1 is a portion of an actual sample of the printout
generated with an ImageWriter II printer when the PRINTERTEST1 program is ex-
ecuted.

A SAMPLE PRINTOUT OF PRINTERTEST1

<table>
<thead>
<tr>
<th>Mac</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>33:</td>
<td>!!!!!!!!!!!!!!!!!!!!!!!</td>
</tr>
</tbody>
</table>
| 34: | """"""""""""""""""""""""""""""""""
| 35: | #######
| 36: | $$$$$$$
| 37: | %%%%%%%
| 38: | &@@@@@@
| 39: | ())))))))
| 40: | )))))))))))))))
| 41: | )))))))))))))))
| 42: | **************
| 43: | +++++++++++
| 44: | ,,,,,,,,,,,
| 45: |,,,,,,,,,,,
| 46: | ,,,,,,,,,,,
| 47: | /////////////////
| 48: | 000000000000000000000000
| 49: | 111111111111111111111111
| 50: | 222222222222222222222222
| 51: | 333333333333333333333333
| 52: | 444444444444444444444444
| 53: | 555555555555555555555555
| 54: | 666666666666666666666666
| 55: | 777777777777777777777777
| 56: | 888888888888888888888888
| 57: | 999999999999999999999999

Fig. 23-1. A portion of the printout of PRINTERTEST1 illustrates the utility of the program for testing
a printer's quality.
CODE LISTING FOR PRINTER TEST1

(*
PrinterTest1, this program prints the complete set of printable Macintosh characters. A single FOR...loop is used for each series of characters. Output is directed to the printer to provide the ability to inspect the printer's character-quality.
*)

(*
Note: This program differs from the PrinterTest2 program, which uses a series of Procedures. PrinterTest1 uses a continuous series of statements as though it is a single Procedure.
*)

PROGRAM PrinterTest1;
uses PasPrinter;

CONST
  LineLen = 20;  (* Set the lengths of lines to print 20 chars. *)

VAR
Ch : Char;
I,J : Integer;

BEGIN
GotoXY(15,10);  // Display the START message at column 15, row 10.
WriteLn('-----> START THE TESTS <-----');  // Print the column headings.

WriteLn(Printer,'Mac Mac');
WriteLn(Printer,'NUMBER CHARACTER');
WriteLn(Printer,'------------------------');
WriteLn(Printer);

FOR I := 33 to 47 DO
BEGIN
  Write(Printer,I,': ');  // Assign numbers 48 through 57 to "I"
  FOR J := 1 to LineLen DO
    Write(Printer,Chr(I));
  WriteLn(Printer);
END;

WriteLn(Printer);
FOR I := 48 TO 57 DO  // Print the Mcintosh character for "I"
BEGIN
  Write(Printer,I,': ');  // Use a CRLF to separate the complex statements. }
  FOR J := 1 TO LineLen DO
    Write(Printer,Chr(I));
  WriteLn(Printer);
END;

WriteLn(Printer);
FOR I := 65 TO 90 DO
BEGIN
  Write(Printer,I,': ');  // Assign numbers 48 through 57 to "I"
  FOR J := 1 TO LineLen DO
    Write(Printer,Chr(I));
  WriteLn(Printer);
END;

WriteLn(Printer);
FOR I := 97 TO 122 DO
BEGIN
  Write(Printer,I,': ');  // Assign numbers 48 through 57 to "I"
  FOR J := 1 TO LineLen DO
    Write(Printer,Chr(I));
  WriteLn(Printer);
END;

194
WriteLn(Printer) ;
FOR I := 58 TO 64 DO
BEGIN
  Write(Printer,I,': ') ;
  FOR J := 1 TO LineLen DO
    Write(Printer,Chr(I)) ;
  WriteLn(Printer) ;
END ;

WriteLn(Printer) ;
FOR I := 91 TO 96 DO
BEGIN
  Write(Printer,I,': ') ;
  FOR J := 1 TO LineLen DO
    Write(Printer,Chr(I)) ;
  WriteLn(Printer) ;
END ;

WriteLn(Printer) ;
FOR I := 123 TO 126 DO
BEGIN
  Write(Printer,I,': ') ;
  FOR J := 1 TO LineLen DO
    Write(Printer,Chr(I)) ;
  WriteLn(Printer) ;
END ;

WriteLn(Printer) ;
WriteLn(Printer) ;
Write(Printer,'=----> THE TESTS ') ;
WriteLn(Printer,'ARE FINISHED <----') ;

GotoXY(10,14) ;
Write('Press the Return key to go to the desktop... ') ;
Read(Ch) ;

WriteLn(Printer,Chr(12)) ;  { Send a form feed signal.  }
WriteLn(Chr(7)) ;            { Sound the Macintosh's bell.  }
END.  { of PrinterTest1 }

CODE LISTING FOR PRINTERTEST2
(*
  PrinterTest2 ... this program prints the complete set of
  printable Macintosh characters. Procedures are used for
  each series of characters. Output is directed to the
  printer to provide the ability to inspect the printer's
  character-quality for potentially defective pins.
*)
PROGRAM PrinterTest2 ;
uses PasPrinter ;

CONST
  LineLen = 20 ;          { Set the lengths of lines to print 20 chars. }  

VAR
  I,J : Integer ;

PROCEDURE SignOn ;

BEGIN
  GotoXY(15,10) ; { Display the START message at column 15, row 10. }
  WriteLn(Output,'=----> S T A R T T H E T E S T S <-----=');
  { Print the column headings. }
  WriteLn(Printer,'Mac Mac ' ) ;
  WriteLn(Printer,' NUMBER CHARACTER' ) ;
  WriteLn(Printer,'--------------------- ' ) ;
  WriteLn(Printer) ;
END ; { of SignOn }

PROCEDURE Next1 ;

BEGIN
  FOR I := 48 TO 57 DO    { Assign numbers 48 through 57 to "I" }    BEGIN
    Write(Printer,I,' : ') ; { Print the number assigned to "I" }    BEGIN
      FOR J := 1 TO LineLen DO    { Print a CRLF to separate the statements. }
        Write(Printer,Chr(I)) ;
        WriteLn(Printer) ;
      END ;
      WriteLn(Printer) ;
  END ; { of Next1 }

PROCEDURE Next2 ;

BEGIN
  FOR I := 65 TO 90 DO    { Print the number assigned to "I" }    BEGIN
    Write(Printer,I,' : ') ;
    FOR J := 1 TO LineLen DO
      Write(Printer,Chr(I)) ;
      WriteLn(Printer) ;
  END ; { of Next2 }
PROCEDURE Next3;

BEGIN
  FOR I := 97 TO 122 DO
    BEGIN
      Write(Printer,I,'': '');
      FOR J := 1 TO LineLen DO
        Write(Printer,Chr(I));
      END;
      WriteLn(Printer);
    END;
  END;   { of Next3 }

PROCEDURE Next4;

BEGIN
  FOR I := 33 TO 47 DO
    BEGIN
      Write(Printer,I,'': '');
      FOR J := 1 TO LineLen DO
        Write(Printer,Chr(I));
      END;
      WriteLn(Printer);
    END;   { of Next4 }

PROCEDURE Next5;

BEGIN
  FOR I := 58 TO 64 DO
    BEGIN
      Write(Printer,I,'': '');
      FOR J := 1 TO LineLen DO
        Write(Printer,Chr(I));
      END;
      WriteLn(Printer);
    END;   { of Next5 }
PROCEDURE Next6;
BEGIN
  FOR I := 91 TO 96 DO BEGIN
    Write(Printer,I,':'');
    FOR J := 1 TO LineLen DO
      Write(Printer,Chr(I));
    END;
    WriteLn(Printer);
  END;
  WriteLn(Printer);
END { of Next6 }

PROCEDURE Next7;
BEGIN
  FOR I := 123 TO 126 DO BEGIN
    Write(Printer,I,':'');
    FOR J := 1 TO LineLen DO
      Write(Printer,Chr(I));
    END;
  END; { of Next7 }

BEGIN { main part of PrinterTest2 }
  SignOn; { Call the PROCEDURES sequentially. }
  Next1;
  Next2;
  Next3;
  Next4;
  Next5;
  Next6;
  Next7;

  GotoXY(10,10); { Overwrite the START message at the screen. }
  WriteLn(Output,'----> THE TESTS ARE FINISHED <----')
  WriteLn(Printer);
  WriteLn(Printer,'----> THE TESTS ARE FINISHED <----');
  WriteLn(Printer,Chr(12)); { Send a form feed signal. }
  WriteLn(Output,Chr(7)); { Sound the Macintosh's bell. }
END. { of PrinterTest2 }
Chapter 24

COPYIT: Copying and Renaming ASCII Files

Here is COPYIT, a short, versatile utility that enables you to make copies of a text file. You can copy the file to another file on the same disk by giving it a new name (as long as it follows the Macintosh file-naming conventions), copy the file to another disk, and give the copy the same name as the original or assign another file name to it.

When you run COPYIT, the screen clears (you've noticed I like to begin this way, although it isn't essential to the program’s performance), and you are instructed by a message at column five, row five on the monitor’s screen: “Type the name of the file to be copied.” You do so, and press the Return key. The next instruction, at row eight is, “Type the name of the file to be created.” Press the Return key, and the program calls the input file by name, Reset(Fi,InName), and opens the output file with the call to ReWrite(Fo,OutName).

The Eof(Fi), End Of File, function is invoked and, while the end of the file is not sensed, the program reads the input file (Fi) a line at a time, and writes it to the designated destination (Fo) a line at a time. When the End Of File marker is read, the files are closed, the Macintosh's bell is sounded, and the program's run is at its end.

CODE LISTING FOR COPYIT

(******************************************************************************) (* Enter the name of a text file; make a copy of *) (* it under another name on the default drive, *) (* or if desired, copy it onto another drive. *) (******************************************************************************)
PROGRAM CopyIt;

VAR
  Ch : Char;
  InName : String[63]; { Name of the source file to copy. }
  OutName : String[63]; { Name of the copy to be made. }
  Fi : Text;  { Symbol assigned to the input file. }
  Fo : Text;  { Symbol assigned to the output file. }
  Stuff : String[80]; { The length of "stuff" in one line. }

BEGIN { CopyIt }
  GotoXY(5,5);
  Write('Type the name of the file to be copied: ');
  ReadLn(InName);

  GotoXY(5,8);
  Write('Type the name of the file to be created: ');
  ReadLn(OutName);

  Reset(Fi,InName);
  Rewrite(Fo,OutName);

  WHILE NOT Eof(Fi) DO { While not at the end of the input file ... } BEGIN
    GotoXY(5,11);
    WriteLn('The two files are now being processed...');
    ReadLn(Fi,Stuff); { Read a line of the input file ... }
    WriteLn(Fo,Stuff) { Copy that line to the output file. }
  END;

  Close(Fo); { Close the output file. }
  Close(Fi); { Close the input file. }

  GotoXY(5,14);
  WriteLn(OutName,' has been created and its Icon');
  GotoXY(5,15);
  WriteLn('contains the contents of ',InName);
  Write(Chr(7)); { Signal that the files are closed; beep. }

  GotoXY(5,19);
  Write('Press Return to end this program... ');
  Read(Ch);
END. { of CopyIt }
Chapter 25

PRINTIT1 & PRINTIT2: Utility Programs to Make Hard Copy Printouts of Your ASCII Files

People always seem to need a utility program that allows them to make a hard copy printout of a file. If you do any programming at all, you should have such a program on hand, a program that is accurate, relatively compact, and fast running. This chapter offers two variations of a short, and very-effective program designed only to make printouts of ASCII text files such as the source-code files that are created with the Turbo Pascal editor.

The files are named PRINTIT1 and PRINTIT2 but, of course, you may give them other names within the naming conventions of the Macintosh operating system. The differences will be explored as we "walk through" a run of the compiled source code for each of the programs.

A "WALK THROUGH" PRINTIT1

After the screen is cleared, the following instruction is displayed on the monitor's screen:

Type the name of the text file you want to print:

In accordance with filename conventions, if the file you want to print is on a drive other than the one on which you have placed the executable version of PRINTIT1, (assuming you have more than one drive) you can precede the name of the file you want to print with the "other" disk drive's identifying name and a colon. Otherwise,
PRINTIT1 expects to find the file in the directory of the default disk, which is the same disk on which PRINTIT1 is located.

The file name you enter is assigned the string variable called NameIt. The PRINTIT variable is assigned the value of NameIt, which it carries to the end of the program. If the file cannot be found, a two-line error message is displayed:

```
Sorry ... but I can't find <NameIt>.
Check the name and the drive and try again.
```

The program is exited to the desktop; so the program must be started all over again. (You will see how this "fatal error" is averted with the addition of more comprehensive error-handling code in PRINTIT2.)

If the file’s name is in the disk’s directory, the program passes the test for an I/O error. It then goes directly to the BEGIN . . . END loop that follows the ELSE directive. A form feed is sent to the printer and, until the program reaches the end of the file being read, each line that is read is then sent to the printer.

When the program reads the end-of-file marker, the file is closed. Then the screen displays:

```
>>>>>> That's it for now! <<<<<<<<
```

A carriage return and line feed, WriteLn(Printer), are sent to ensure that the printer’s buffer has been emptied. The program’s run and the printout have been completed.

**A "WALK THROUGH" PRINTIT2**

The PRINTIT2 version has a few additional lines of code you may find worthwhile. Note that IOerr has been declared as a VAR of the Boolean type. The {$I-} switch appears at the start of the program’s operation. Error handling is now within a REPEAT . . . UNTIL loop. The effect is important.

In PRINTIT1, if an I/O error occurs, such as would be caused by the entry of an invalid or inaccessible file name, the program is exited the first time after the error message is displayed. With the I/O error-handling code of PRINTIT2, the same error message as is generated with PRINTIT1 is displayed when an I/O error occurs. However, the program doesn’t quit at that point. Instead, the program goes back to display the initial request:

```
Type the name of the text file you want to print:
```

There are two other relatively minor distinctions in the PRINTIT2 code and operation when compared with PRINTIT1. One, when printing begins, the program displays the message:

```
>>>>>> Now Printing <NameIt> <<<<<<<<
```
And, two, the form feed is sent to the printer after the sign off message is displayed, instead of at the start of the actual printout. Some of us prefer it that way.

**CODE LISTING FOR PRINTIT1**

(*********************************************************************)
(* Enter the name of a source file and print a copy of the "source." *)
(* Compare this with PrintIt2. *)
(*********************************************************************)

PROGRAM PrintIt1;
uses PasPrinter;

VAR
   Ch : Char;
   InName : String[63];  ( Name of the source file to print. )
   Fi : Text;           ( Symbol assigned to the input file. )
   Stuff : String[80];  ( The length of "stuff" in one line. )

BEGIN { PrintIt1 }
   GotoXY(5,5);
   Write('Type the name of the file to be printed: ');
   ReadLn(InName);

   (* Bare bones error control...disable IO error checking. *)
   Reset(Fi,InName);

   WHILE NOT Eof(Fi) DO { While not at the end of the input file ... } 
      BEGIN
         GotoXY(5,11);
         WriteLn('=----> ', InName, ' Is Now Being Printed <-----=');
         ReadLn(Fi,Stuff);   ( Read a line of the input file ... )
         WriteLn(Printer,Stuff)  ( Send that line to the printer. )
      END;

   Close(Fi);   ( Close the input file. )
   Write(Chr(7)); ( Signal that the files are closed. )

   GotoXY(5,13);
   Write('Press Return to go to the desktop... ');
   Read(Ch);
   WriteLn(Printer,Chr(12));  ( Eject the paper with a form feed. )
END.  { of PrintIt1 }
CODE LISTING FOR PRINTIT2

(* PrintIt2 reads a text file and then prints it. *)
(* This is a relatively sophisticated version of *)
(* PrintItl. Make sure the printer is on and *)
(* ready to go. *)
(* **************************************************)

PROGRAM PrintIt2;
uses PasPrinter;

VAR
  Ch       : Char;
  InName   : String[63];  (* Name of the file to print. *)
  IOerr    : Boolean;
  Fi       : Text;       (* Symbol assigned to the file. *)
  Stuff    : String[80];

BEGIN  (*PrintIt2*)
  {$I-} { Turn off the IO error handling. We’ll do it with code. }
  REPEAT
    GotoXY(5,10);        (* Display the request for a name. *)
    Write('Type the name of the text file you want to print: ');
    ReadLn(InName);     (* Wait for the name to be typed in. *)
    ReSet(Fi,InName);
    IOerr := (IOresult <> 0);

    IF IOerr THEN       (* If there’s an error in the name ... *)
      BEGIN
        Write(Chr(7));    (* ... sound the computer’s bell. *)
        GotoXY(10,12);    (* Advise the user there’s an error. *)
        WriteLn('Sorry...but I can’t find ’,InName,‘.‘);
        GotoXY(10,14);
        WriteLn('Check the name and try again.’);
        GotoXY(10,17);
        Write('Press Return to try again...or, <Q> to quit... ’);
        Read(Ch);
        IF (Ch) IN [ ‘Q’, ‘q’] THEN EXIT;
        ClearScreen;
      END;
  UNTIL NOT IOerr;    (* Restore the default condition. *)
  {$I+}  (* There’s no error. Continue the program. *)
  ClearScreen;        (* Clear the screen of old stuff. *)
  GotoXY(20,10);      (* Advise the user, things are okay now. *)
  WriteLn('=----> Now Printing: ’,InName,‘ <----=');
WHILE NOT EOF(Fi) DO  { While we are still in the file, }
    BEGIN
        ReadLn(Fi,Stuff) ;  { read and print it, then }
        WriteLn(Printer,Stuff)  { ... read the next line. }
        END ;  { Read and print to End Of File. }
    Close(Fi) ;  { Close the file that was opened for printing. }
    WriteLn(Printer,Chr(12)) ;  { Finally, do a form feed }
    END ;
END.  { PrintIt2 }
SHOWIT is a brief and interactive program that enables you to name a text file and, assuming it exists on the disk, display its contents on the Macintosh screen. It's a handy utility program that enables you to view the contents of a text file to remind yourself of its contents and purpose. The source code is easily entered. Compilation takes just a few seconds. When you run the executable SHOWIT program, by double-clicking on its icon, the Macintosh screen clears and this instruction is displayed:

Enter the name of the file to be displayed:

In response, you key in the name of the file to be displayed and press the Return key. If, for any reason that file cannot be found—perhaps an error in typing the name, or the file is not on the disk—the Macintosh's screen displays an error message and an instruction:

The file name is not correct.
Press <Q> to Quit, or the Return key to continue:

If you press the Return key, the original instruction reappears, allowing you to re-enter the name of the file you want to view. If you press the uppercase or lowercase Q, however, the program quits and you are returned to the desktop of the Macintosh.

Let's assume all is in order. You've correctly entered the name of a file that exists on the disk. The program continues. The EOF, End Of File, function is invoked and, WHILE the end of the file you named is not sensed, the program reads that input file,
(Fi), a line at a time and sends it, (Stuff), to the video screen a line at a time. When
the marker for the end of the file is sensed, the files are closed, and the program’s
run is at its END.

The screen displays a closing message:

====> THIS IS THE END OF THE FILE: <InName> ====>

And the instruction to:

*** Press Return to go back to the desktop ***

CODE LISTING FOR SHOWIT

{ Name a text file and display its contents on the screen. }

PROGRAM ShowIt;

VAR
  Ch   : Char;
  Fi   : Text;
  InName : String[63]; { InName is the name of the file }
  IOerr : Boolean;
  Stuff : String[80]; { to be displayed on the screen. }

BEGIN
  { Turn off the automatic "abort on IO error." }
  {$I-}

REPEAT
  WriteLn;
  WriteLn;
  Write('Enter the name of the file to be displayed: ') ;
  { Wait for the name of the file to be typed. }
  ReadLn(InName);
  Reset(Fi,InName);

  IOerr := (IOresult <> 0);
  IF IOerr THEN  { If there is an IO error, activate this loop. }
  BEGIN
    WriteLn;
    WriteLn('The file name is not correct.') ;
    WriteLn;
    { Here are the user’s options. }
    Write('Press <Q> to Quit, or the Return key to continue: ') ;
    Read(Ch);

    IF (Ch = 'Q') OR (Ch = 'q') THEN
      { <Q> or <q> halts the program; returns to the desktop. }

208
HALT;
END; // of the IO error loop.

UNTIL NOT IOerr;
   // Restore the automatic "abort on IO error.
   (${I+}
   WriteLn;
   WriteLn;

WHILE NOT EOF(Fi) DO
   // The WHILE-DO loop continues until End-Of-File is reached.
   BEGIN // Start the WHILE...DO loop.
      ReadLn(Fi,Stuff);
      WriteLn(Stuff);
   END; // of the WHILE...DO loop.

Close(Fi); // Close the input file, the one being read.
WriteLn;
WriteLn;
WriteLn('=----> THIS IS THE END OF THE FILE: ', InName,' <----=');
WriteLn;

Write(' *** Press Return to go back to the desktop *** ');
Read(Ch);
END. // of ShowIt}
Your Macintosh computer and its printer can become an electric (or electronic) typewriter with TYPEIT. This program is included for three reasons: 1) it provides additional instruction in writing brief programs that become relatively powerful utilities, 2) printing hard copy of keyed text is a valid use for the computer, and 3) programs that fulfill the same tasks are commercially marketed. These lead us to the assumption that TYPEIT is worth entering, compiling, and adding to your store of handy utilities.

The program, when executed, contains an ample set of instructions for the user. The source code incorporates additional comments, which make further tutorial material unnecessary at this point.

**CODE LISTING FOR TYPEIT**

```pascal
{ This program displays and prints 4 lines of } 
{ text, such as might be used in addressing } 
{ envelopes or letterheads. } 

PROGRAM TypeIt ; 
uses PasPrinter ; 

VAR 
   Ch : Char ; 
```
BEGIN ( TypeIt )
GotoXY(1,5);
WriteLn('*** BE SURE TO TURN ON THE PRINTER... NOW ***');
WriteLn;
WriteLn;
WriteLn('Type up to four lines of text. (Be sure to')
WriteLn('press Return at the end of each line.)');
WriteLn;
WriteLn('After you type the fourth line and press')
WriteLn('the Return key, the text will be displayed.');
WriteLn;
WriteLn('You will receive new instructions to start printing...');
WriteLn;

Write('Type the text for Line #1: '); ReadLn(Line1);
Write('Type the text for Line #2: '); ReadLn(Line2);
Write('Type the text for Line #3: '); ReadLn(Line3);
Write('Type the text for Line #4: '); ReadLn(Line4);

ClearScreen;
WriteLn;
WriteLn('You entered the following text:')
WriteLn;
WriteLn(Linel);
WriteLn(Line2);
WriteLn(Line3);
WriteLn(Line4);

Write('Press Return to start printing... ');
Read(Ch);
WriteLn(Printer, ' ',Lin1) ;
WriteLn(Printer, ' ',Lin2) ;
WriteLn(Printer, ' ',Lin3) ;
WriteLn(Printer, ' ',Lin4) ;

END.   { of TypeIt }
Chapter 28

EDITIT1, EDITIT2 & EDITIT3: Three Text Editor Programs

The previous chapter’s program, TYPEIT, used envelope addressing as an example of an application for converting your Macintosh to an electronic typewriter. The previous program was limited specifically to four lines of text, each 40 characters long, which should be more than adequate for typing the information that typically includes a person’s name, title or company name, and mailing address. Well, if you are going to address envelopes with your computer, it seems reasonable to use the Macintosh to type the letter, memorandum, or other typewritten material that goes into the envelope. And that’s exactly where the EDITIT series of applications programs fits in.

The first program, EDITIT1 is quite fundamental, as far as text editing goes. The second one, EDITIT2 adds a useful attribute to the first. The features of the third one, EDITIT3, are enhanced to make this program well suited to the task. This chapter describes the unique features and provides the complete source code for each of the three programs.

EDITIT1

These characteristics apply to all three programs in this chapter. The programs will send text to the printer; therefore, we must call the PasPrinter unit right after the first line of source code, which gives the PROGRAM name. In each program, the VAR declares that the variable “Text” is a string of 70 characters, maximum. The flow of the code is smooth, progressing from the top down; hence, I chose not to use separate procedures. If you prefer separate procedures, however, there’s nothing to stop you from rewriting the code to use as many procedures as you see fit, and a separate main
block of code to call the procedures in their proper order. Rewriting the code will give you good practice in learning how to revise or modify source code.

When you run the program, it signs on by displaying a set of instructions for the user: turn on the printer, adjust the paper, and so on. You can print up to 70 characters per line. You should be aware that you can type as many characters as the screen will hold, but the printer will not print more than 70 of them. Because the variable Text was declared to be a string of length 70, the surplus beyond 70 characters are not stored in the buffer; they are just dropped.

If, when you are instructed to enter the text, you press the Return key without entering any characters, the buffer believes it has received a blank line, and it sends a line feed to the printer. In all three programs, if you want to skip a line or change line spacing—double, triple, or quadruple—press the Return key an appropriate number of times without entering any printable characters. (The Space bar, for example, sends a character that isn’t printable, and isn’t visible at the printer.)

EDITIT1 redisplays the characters that have been stored in the buffer, sends them to the Console or the video monitor of the Macintosh. At the same time, it sends them to the printer. To control the top margin of the sheet on which you are doing the printing, you must set the top of the form in the printer at the position where you want the first line of typing to appear. While doing the typing, you will have to visually monitor the paper’s position in the printer to be sure you end your typing before reaching the bottom of the form.

You can end the program by typing three uppercase or lowercase letters <Z>. The IF . . . THEN loop tests the text each time you make entries. IF it reads three letters <Z> in succession, no spacing between them, THEN it Halts the program, returning the Macintosh to the desktop.

CODE LISTING FOR EDITIT1

```pascal
( This program displays and prints text entered at the keyboard. )
( It is a basic non-formatting text editor. )

PROGRAM EditIt1;
uses PasPrinter;

VAR
Text : String[70]; { Text-String length is 70 characters. }

BEGIN  { EditIt1 }
GotoXY(15,1);
WriteLn(' ***BE SURE TO TURN ON THE PRINTER...NOW ***');
GotoXY(15,3);
WriteLn('*** ADJUST THE PRINTER’S PAPER...AND START ***');

GotoXY(5,5);
Write('Type up to 70-characters on a single line.');
WriteLn(' Then press the Return key.');
```
GotoXY(5,6);
  Write('After you press the Return key, ');
  WriteLn('the text will be printed.');

GotoXY(5,8);
  Write('Pressing the Return key without entering text is ');
  WriteLn('the same as sending');
  GotoXY(5,9);
  WriteLn('a blank line to the printer.');
  GotoXY(5,10);
  WriteLn('Therefore, it sends a single line feed to the printer.');

GotoXY(5,12);
  Write('To end the program, type "ZZZ" or "zzz" when asked ');
  WriteLn('to enter the text. ');
  GotoXY(5,13);
  WriteLn('Then press the Return key.');

REPEAT { Repeat the request for keyboard entries until... }
  WriteLn;
  WriteLn;
  WriteLn('Enter up to 65 characters of text:');
  WriteLn;
  Write(' ');
  ReadLn(Text); { Indent 10 characters. }
  ReadLn(Text); { Read the keyboard entries. }

  { If you enter three upper- or lower-case Z's, the program ends. }
  IF (Text = 'ZZZ') OR (Text = 'zzz') THEN
    HALT;
  ClearScreen;
  WriteLn;
  WriteLn('You entered the following text:');
  WriteLn;
  WriteLn(',' ,Text); { Display the text with indent. }
  WriteLn( Printer, ',' ,Text); { Print the text, indented 10. }
  UNTIL (Text = 'ZZZ') OR (Text = 'zzz'); { Three Z's also end the loop. }

END. { of EditIt1 }

EDITIT2

This version's code is a bit different, but it produces the same results. The user interface has been improved. (Note, too, there is only one WriteLn, at the very end of the program. In all other cases in this program's code, it uses the Write function, depending on the GotoXY( ) function for positioning the cursor.)

An improvement in the user interface is given with the display of a line of 70 hyphens starting at column five, row eight, [GotoXY(5,8)] that appears one line below the cur-
sor’s position for the entry of the 70 text characters. Also, every 10th hyphen is numbered from 1 to 70. The purpose of this display is to guide your typing, letting you know when you are getting close to or reaching the maximum limit of characters that will be stored in that particular string of text.

As before, typing three letters <Z> in succession ends the program and returns you to the Macintosh’s desktop.

**CODE LISTING FOR EDITIT2**

```
{ EditIt2 displays and prints text entered at the keyboard. }
{ It is a basic non-formatting text editor. }

PROGRAM EditIt2 ;
uses PasPrinter ;

VAR
  Text : String[70] ;  { Text-String length is 70 characters. }

BEGIN  { EditIt2 }
  ClearScreen ;

  { Display the instructions for the user. }
  GotoXY(15,1) ;
  Write(' ***BE SURE TO TURN ON THE PRINTER...NOW ***') ;
  GotoXY(15,3) ;
  Write(' ***ADJUST THE PRINTER’S PAPER...AND START ***') ;

  GotoXY(5,5) ;
  Write('Type up to 70-characters on a single line.') ;
  Write(' Then press the Return key.') ;

  GotoXY(5,6) ;
  Write('After you press the Return key, ') ;
  WriteLn('the text will be printed.') ;

  GotoXY(5,8) ;
  Write('Pressing the Return key without entering text is ') ;
  Write('the same as sending') ;
  GotoXY(5,9) ;
  Write('a blank line to the printer. Therefore, it has ') ;
  Write('the effect of making') ;
  GotoXY(5,10) ;
  Write('the printer skip a line; do a double-space .') ;

  GotoXY(5,12) ;
  Write('To end the program: ') ;
  GotoXY(5,13) ;
  Write('Type "ZZZ" or "zzz" when instructed ') ;
```
Write('to enter the text.');
GotoXY(5,14);
Write('Then press the Return key.');

GotoXY(5,17);
Write('NOW...PRESS THE RETURN KEY TO GET STARTED...');
ReadLn;

REPEAT { Repeat the request for keyboard entries until... }
ClearScreen;
GotoXY(5,5);
Write('Enter up to 70 characters of text:');

{ Draw a line of 70 hyphens as a guide to entering 70 characters. }
GotoXY(5,8);
Write('-----------------------------------');
Write(' - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ');

{ Place numbers at every 10th hyphen as visual character counters. }
GotoXY(5,9);
Write('1 2 3 4');
Write('5 6 7');
GotoXY(5,10);
Write('0 0 0 0');
Write('0 0 0');

GotoXY(5,7); { Position the cursor above the guide line. }
ReadLn(Text); { Read the keyboard entries. }

{ If you enter three upper- or lower-case Z's, the program ends. }
IF (Text = 'ZZZ') OR (Text = 'zzz') THEN
  HALT;

WriteLn(Printer,' ',Text); { Print the text, indented 10. }
UNTIL (Text = 'ZZZ') OR (Text = 'zzz'); { Three Z's also end the loop. }
END. { of EditIt2 }

EDITIT3

This program enhances the user interface given for EDITIT2. In addition to the features offered by EDITIT2, it counts the number of lines that have been typed, including those cases where no characters may have been typed but a line has been skipped by pressing the Return key. Because this is still using one of the available lines on the page, the program counts the line as "used."

The line count is stored in the variable LineCounter, which is declared to be of the type Integer. Within the program LineCounter is initialized or reset to the value of zero. This is done each time the program is run, just before the REPEAT...UNTIL
loop is entered. The line of code, LineCounter := LineCounter + 1 increments the value of LineCounter by 1. Thus, the first time it is recognized by the program, which is immediately after the ClearScreen function is called within the REPEAT . . . UNTIL loop, it is incremented from the initial zero to a value of one. Then, its value is displayed in column one, row five to let the user know exactly what line is now available.

There are six single-spaced lines per inch. A standard 11-inch long sheet has 66 single-spaced lines available. Therefore, if we know the number of lines allowed for the top margin, the point at which we started printing, and if we know the number of lines desired for the bottom margin, it is easy to calculate, by subtraction, the number of lines we have available for typing/printing. Don't forget to keep an eye on the line number display, which is being updated constantly. As with EDITIT2 and EDITIT3, it provides a display of hyphens and character numbers to guide text entries.

CODE LISTING FOR EDITIT3

```pascal
{ EditIt3 prints text entered at the keyboard. A basic }{ non-formatting text editor, it counts the lines. }

PROGRAM EditIt3;
uses PasPrinter;

VAR
LineCounter : Integer;
Text : String[70]; { Text-String length is 70 characters. }

BEGIN  { EditIt3 }
ClearScreen;

( Display the instructions for the user. )
GotoXY(15,1);
Write(' ***BE SURE TO TURN ON THE PRINTER...NOW ***');
GotoXY(15,3);
Write(' ***ADJUST THE PRINTER’S PAPER...AND START ***');

GotoXY(5,5);
Write('Type up to 70-characters on a single line.’);
Write(' Then press the Return key.’);

GotoXY(5,6);
Write('After you press the Return key, ’);
WriteLn('the text will be printed.’);

GotoXY(5,8);
Write('Pressing the Return key without entering text is ’);
Write('the same as sending’);
GotoXY(5,9);
Write('a blank line to the printer. Therefore, it has ’);
Write('the effect of making’);
```

220
GotoXY(5,10) ;
Write('the printer skip a line; do a double-space .') ;

GotoXY(5,12) ;
Write('To end the program: ') ;
GotoXY(5,13) ;
Write('Type "ZZZ" or "zzz" when instructed ') ;
Write('to enter the text.') ;
GotoXY(5,14) ;
Write('Then press the Return key.') ;

GotoXY(5,17) ;
Write('NOW...PRESS THE RETURN KEY TO GET STARTED...') ;
ReadLn ;

LineCounter := 0 ; { Initialize LineCounter. }

REPEAT { Repeat the request for keyboard entries until... } ClearScreen ;

{ Starting with 1, increment LineCounter by 1 each }{ time the REPEAT...UNTIL loop is repeated. }
LineCounter := LineCounter + 1 ;

GotoXY(1,2) ;
Write('You are at line number: ',LineCounter) ;

GotoXY(5,5) ;
Write('Enter up to 70 characters of text:') ;
{ Draw a line of 70 hyphens as a guide to entering 70 characters. }
GotoXY(5,8) ;
Write('--------------------') ;
Write('--------------------') ;
{ Place numbers at every 10th hyphen as visual character counters. }
GotoXY(5,9) ;
Write('1 1 2 3 4') ;
Write('5 6 7') ;
GotoXY(5,10) ;
Write('0 0 0 0 0') ;
Write('0 0 0 ') ;

GotoXY(5,7) ;{ Position the cursor above the guide line. }
ReadLn(Text) ; { Read the keyboard entries. }

{ If you enter three upper- or lower-case Z's, the program ends. }
IF (Text = 'ZZZ') OR (Text = 'zzz') THEN
BEGIN
WriteLn(Printer,Chr(12));  ( FormFeed the paper at the end. )
HALT;
END;

WriteLn(Printer,' ',Text);  ( Print the text, indented 10. )
UNTIL (Text = 'ZZZ') OR (Text = 'zzz');  ( Three Z's also end the loop. )
END.  ( of EditIt3 )
Chapter 29

START, UPDATE & LISTALL: Essential Elements of a Database Program

Turbo Pascal offers a capability for creating programs for files that handle text and data effectively. In this chapter, you will write the segments for three individual, but related, programs—Start, UpDate, and ListAll—the key elements of a program to create, maintain, display, and print database files.

Each of the three segments, or programs, declares the same labels, assignments, and values in the CONST and in the TYPE blocks of the programs. If, however, there is the slightest difference or an error is entered in writing the code, the program cannot ignore it. You will note, too, that each of the programs has only a main block; it does not make calls to individual procedures.

THE START PROGRAM

The first of the three programs is appropriately named START. The value declared for the label MaxNumRecords in the CONST block provides for a maximum number of 100 individual records. Of course, the number of records can be increased or decreased merely by changing the value of 100 assigned to the CONStant MaxNumRecords.

The TYPES can be given labels or identifiers that best suit the application of the program. Obviously, with an understanding of the simple principles involved, you can create other identifiers of your own choosing. Whatever the relevance of the labels' names might be, the structure of the program remains essentially the same as given here.

START requests the entry of the name of the data file to be operated on. After you type the name of the data file (DataFile) and press the Return key, the program asks whether or not this is a new file (Yes or No?). If we respond by pressing the <N>
key, for No, this specific program ends and you are returned to the desktop. Obviously, the START program is used only when a new file is to be named and created. If you respond <Y> for Yes, this is a new file, you are advised or alerted by the message:

READ THE FOLLOWING INSTRUCTIONS CAREFULLY

If (DataFile) exists, its contents will be erased.

If you want to create a new file with the same name, press the <Y> key.

To return to the desktop without creating a new (DataFile), press the Return key.

Pressing the Return key has the same effect as pressing the <N> key did in response to the very first query. The program is exited, and you are returned to the desktop. If we do press the <Y> key, you are informed that the file (DataFile) is now being created.

“DataFile” is assigned the identifier “ItemFile” with a capacity of 100 records, which is the value of MaxNumRecords as assigned in the CONST block. If you were to exit the program right now, an examination of the icons at the desktop would reveal the existence of a new file identified by the name you entered in response to the initial query.

CODE LISTING FOR START

(* START is the first of a group of three small *)
(* database manager sections. Together they form *)
(* a useful program for creating data files. *)

PROGRAM Start ;

CONST
MaxNumRecords = 100 ; { The max of 100 records }
{ can be changed here. }

TYPE
PersonName = String[25] ;
Addr = String[25] ;
Cit = String[20] ;
St = String[2] ;
Z = String[5] ;
Ph = String[14] ;
Data = String[128] ;
Item = RECORD
  RecordNumber : Integer ;
  Name : PersonName ;
  Address : Addr ;
  City : Cit ;
VAR
  Ch  : Char;
  DataFile : Data;
  ItemFile : FILE OF Item;
  ItemRec  : Item;
  I,Counter : Integer;
  IOerr     : Boolean;
BEGIN  ( Start )
  GotoXY(1,5);
  WriteLn('------------------ START ------------------') ;
  WriteLn ;
  WriteLn('Enter <C> to Create a new data file.') ;
  Write('Enter <Q> to Quit this program now: ') ;
  ReadLn(Ch) ;
  IF (Ch) IN ['Q','q'] THEN EXIT ;
  WriteLn ;
  WriteLn ;
  Write('Enter the name of the data file: ') ;
  ReadLn(DataFile) ;
  WriteLn ;
  WriteLn ;
  Write('Is this a new file? Y)es or N)o: ') ;
  ReadLn(Ch) ;
  WriteLn ;
  WriteLn ;
  WriteLn('READ THE FOLLOWING INSTRUCTIONS CAREFULLY!') ;
  WriteLn ;
  WriteLn('If ',DataFile,' exists, its contents will be erased.') ;
  WriteLn ;
  WriteLn('If you want to create a new file with') ;
  WriteLn('the same name, press the <Y> key.') ;
  WriteLn ;
  WriteLn('To return to the desktop without creating a new') ;
Write(DataFile,' press the Return key: ') ;  
Read(Ch) ; 

IF (Ch) IN ['Y','y'] THEN ({ Accept upper or lower case <Y>. }) 
BEGIN ({ If (Ch) is <Y>, begin this part. }) 
WriteLn ; 
WriteLn ; 
WriteLn(DataFile,' is now being created.') ; 
WriteLn ; 

ReWrite(ItemFile,DataFile) ; ({ Open the file. If it }) 
{ already exists, delete } 
{ any data that is in it. } 

WITH ItemRec DO 
BEGIN 
{ Reserve space for the data. } 
NAME := ' ' ; 
FOR I := 1 TO MaxNumRecords DO 
BEGIN 
RecordNumber := I ; 
Write(ItemFile,ItemRec) ; 
END ; 
END ; 
WriteLn ; 
END. { of Start }

START creates the file in which the records are stored, or if a file already exists 
with the same name, it is overwritten. If the file exists, overwriting it will erase, de­
lete, or otherwise make any of the data that may have been stored in its records un­
retrievable. So, obviously, if the file already exists and you do not intend to change 
its name or erase its contents, you may skip over the START program for the moment 
and go directly to the second of the three programs, UPDATE.

THE UPDATE PROGRAM

UPDATE is the maintenance program. With it you can add information to a specific 
record and delete or overwrite the information stored in any of the records you may 
have created earlier. As in START, the maximum number of records is determined 
by the value assigned to MaxNumRecords in the CONST block.

Notice that the CONST and the TYPE blocks are the same in both the START 
and UPDATE programs. There are, however, minor but important differences in the 
VAR block. The variable “Ch” has been dropped, and the variable “Counter” has been 
added. The program itself uses no PROCEDURE calls; it is an entity or single block.

The program instructs you to:

Enter the name of the data file to update:
After you do so and press the Return key, the program attempts to read the name of the data file you entered. If it fails to find the icon for the file’s name, the error message and an instruction are displayed:

I can't find the (DataFile) file!
Press Return to quit...

If you see this message, you have no choice but to press the Return key, at which point the program quits and the Macintosh returns to the desktop. On the other hand, if the data file you named does exist, it passes the test, and jumps to ELSE in the program’s code. Then the program instructs you to:

Enter the record number (0 = Stop):

At this point in the program, it seeks the record number, finds it, and begins to display the Write and ReadLn statements to which you must respond with the appropriate data. After you have entered your responses and pressed the Return key, the screen clears and you are instructed to:

Enter the record number (0 = Stop).

You can repeat the procedure as often as you wish, and type the number zero when you want to quit. When you do type a zero <0>, the file you have been working with is closed and the program is exited, returning to the desktop.

CODE LISTING FOR UPDATE

(* UpDate is the second of a group of three small *)
(* database manager sections. Together they form *)
(* a useful program for creating data files. *)

PROGRAM UpDate;

CONST
    MaxNumRecords = 100;

TYPE
    PersonName = String[20];
    Addr = String[20];
    Cit = String[20];
    St = String[20];
    Z = String[5];
    Ph = String[14];
    Data = String[128];
    Item = RECORD
RecordNumber : Integer;
Name : PersonName;
Address : Addr;
City : Cit;
State : St;
ZIP : Z;
Phone : Ph;
END;

VAR
Ch : Char;
DataFile : Data;
ItemFile : FILE OF Item;
ItemRec : Item;
I,Counter : Integer;

BEGIN
GotoXY(1,5);
WriteLn('------------------------ UPDATE ------------------------')
WriteLn;
WriteLn;
Write('Enter the name of the data file to update: ');
Read(DataFile);

{$I-}
ReSet(ItemFile,DataFile);
{$I+}

IF IOresult <> 0 THEN
BEGIN
WriteLn;
WriteLn;
WriteLn('I can' 't find the "',DataFile,'"," file!');
WriteLn;
Write('Press Return to quit... ');
ReadLn(Ch);
END
ELSE
BEGIN
WriteLn;
WriteLn;
Write('Enter the record number (0 = Stop): ');
ReadLn(Counter);
WHILE Counter IN [1..MaxNumRecords] DO
BEGIN
Seek(ItemFile,Counter - 1);

}
Read(ItemFile,ItemRec);

WITH ItemRec DO
BEGIN
   WriteLn;
   Write('Enter the name: ');
   ReadLn(Name);

   Write('Address: ');
   ReadLn(Address);

   Write('City: ');
   ReadLn(City);

   Write('State: ');
   ReadLn(State);

   Write('ZIP number: ');
   ReadLn(ZIP);

   Write('Phone number: ');
   ReadLn(Phone);

   RecordNumber := Counter;
END;

Seek(ItemFile,Counter - 1);
Write(ItemFile,ItemRec);

WriteLn;
Write('Enter the record number (0 = Stop): ');
ReadLn(Counter);

END;
END;
Close(ItemFile);
END.  { of UpDate }

THE LISTALL PROGRAM

Now that you have created a data file and entered data, you must have a way to retrieve and use that data. This is the purpose of LISTALL. As with the other two programs in this set of three, the labels and the values of the CONST and the TYPE blocks are retained. The contents of the VAR block are the same as those of the START program.

LISTALL instructs you to:

Enter the name of the data file to list:
If the file's name doesn't exist, the error message "I can't find that file" is displayed, and the program is exited. If the file's name is correct and does exist as you entered it, the program asks:

Printout or Screen? (S or P):

If you should type any character other than <P> or <S>, the query will be repeated UNTIL you do enter one of the two characters; either in uppercase or lowercase. <S> causes a rapid retrieval and display of all the records in the file. <P> does the same rapid retrieval, but sends the information to the screen and to the printer, which should be on and ready. If the printer is not on or ready, the record appears only at the video display. When the printer is turned on, connected, and ready, however, the printout is made. The program then continues, closes the data file, and displays the message:

***** DONE *****

CODE LISTING FOR LISTALL

(* ListAll is the third of a group of three small *)
(* database manager sections. Together they form *)
(* a useful program for creating data files. *)

PROGRAM ListAll;
uses PasPrinter; { This program must call "uses PasPrinter". }

CONST
   MaxNumRecords = 100 ; { The max of 100 records }
      { can be changed here. }

TYPE
   PersonName = String[25] ;
   Addr = String[25] ;
   Cit = String[20] ;
   St = String[2] ;
   Z = String[5] ;
   Ph = String[14] ;
   Data = String[128] ;
   Item = RECORD
      RecordNumber : Integer ;
      Name : PersonName ;
      Address : Addr ;
      City : Cit ;
      State : St ;
      ZIP : Z ;
      Phone : Ph ;
   END ;

VAR
   Ch : Char ;
DataFile : Data;
ItemFile : FILE OF Item;
ItemRec : Item;
I,Counter : Integer;
IOerr : Boolean;

BEGIN { ListAll }
  GotoXY(1,5);
  WriteLn('----------------- DISPLAY & PRINT -----------------');
  WriteLn;

  {$I-} { See ... Procedure UpDate ... for comments. }
REPEAT
  WriteLn;
  Write('Enter the name of the data file to list: ');
  ReadLn(DataFile);

  ReSet(ItemFile,DataFile);
  IOerr := (IOresult <> 0);
  IF IOerr THEN
    BEGIN
      WriteLn;
      WriteLn;
      WriteLn('I can’t find ',DataFile);
      WriteLn;
      Write('Press <Q> to Quit. Press Return to continue: ');
      Read(Ch);
      WriteLn;
      IF (Ch) IN ['Q', 'q'] THEN EXIT;
    END;
  UNTIL NOT IOerr;

  {$I+}
  { Continue if there’s no error in the datafile’s name. }
BEGIN
  REPEAT
    WriteLn;
    WriteLn;
    Write('Printout or Screen? (S or P): ');
    ReadLn(Ch);
  UNTIL (Ch) IN ['S', 's', 'P', 'p']; { Accept only <S> or <P>. }
FOR I := 1 TO MaxNumRecords DO
  BEGIN
    Read(ItemFile,ItemRec);
  END;
WITH ItemRec DO
BEGIN
  IF Name <> '' THEN
  BEGIN
    BEGIN
      { Send the record to the video display. }
      WriteLn;
      WriteLn(' * Record Number: ','RecordNumber:3,'');
      WriteLn(' ');
      WriteLn(' * Person''s Name: ',Name:25);
      WriteLn(' * Address: ',Address:25);
      WriteLn(' * City: ',City:20);
      WriteLn(' * State: ',State:2);
      WriteLn(' * ZIP: ',ZIP:5);
      WriteLn(' * Phone Number: ',Phone:14);
      WriteLn;
    END;
    IF (Ch) IN ['P','p'] THEN
    BEGIN
      BEGIN
        { Send the record to the printer. }
        WriteLn(Printer);
        WriteLn(Printer,' * Record Number: ','RecordNumber:3,'');
        WriteLn(Printer, ' ');
        WriteLn(Printer,' * Person''s Name: ',Name:25);
        WriteLn(Printer,' * Address: ',Address:25);
        WriteLn(Printer,' * City: ',City:20);
        WriteLn(Printer,' * State: ',State:2);
        WriteLn(Printer,' * ZIP: ',ZIP:5);
        WriteLn(Printer,' * Phone Number: ',Phone:14);
        WriteLn(Printer);
      END;
    END;
  END;
END;
Close(ItemFile); { Close the file that was opened. }
WriteLn;
WriteLn(' ***** DONE *****');
WriteLn;
WriteLn;
Write('Press Return to go back to the desktop... ');
Read(Ch);
END; { of ListAll }

232
Chapter 30

DATABASE1: Database Program with Enhanced I/O Error Handling

This chapter is based on the lessons learned from Chapter 29. Therefore, it follows naturally, that it would be valuable to read Chapter 29 first. The three separate programs presented in Chapter 29, START, UPDATE, and LISTALL, are converted to procedures, thereby integrating them into a single program.

The procedures are called from the main block of a new program now named DATABASE1. This program contains more comprehensive I/O error handling than the error handling in the code listings for START, UPDATE, and LISTALL in Chapter 29. An important improvement, too, is the fact that the user can move from one program (now procedures) to the other without first being required to return to the desktop. The error handling is essentially transparent to the user—until a human error, a typo, is made or an incorrect data file name is introduced.

As you look at the listings, note that the labels, declarations, assignments, and values are essentially the same in the CONST and the TYPE blocks. The VAR block has been expanded to include all the variables that are used in START, UPDATE, and LISTALL, plus a new one, which declares the variable IOerr to be of the type Boolean.

The program begins with a REPEAT . . . UNTIL loop that repeats the sequential calls to the Procedures and built-in Functions until you deliberately enter the character <Q> to quit the program. Start is the first procedure called. Except for the fact it cannot stand alone, PROCEDURE Start is identical in content to PROGRAM Start.

The completion of PROCEDURE Start opens a call to PROCEDURE Update. The PROCEDURE Update is significantly different from the Program UPDATE. The Procedure Update asks for the same entries as does Program UPDATE; however, note
that a REPEAT ... UNTIL segment encloses the first statement in which the name of the data file is requested, and the error message (if appropriate) is signaled. The difference is apparent in the program's operation. Whereas an I/O error in the Program UpDate would, in effect, cause the program to abort, making it necessary to restart the Program UPDATE, the error-handling method in the Procedure UpDate prevents such an abort or, as it is woefully but popularly called by programmers, a "fatal error."

Thus, in case of an I/O error, such as caused by a typo or entering a nonexistent or misspelled file name, the REPEAT ... UNTIL segment of code returns the program to the Write statement that immediately follows the REPEAT instruction. If you do not succeed in identifying the name of an appropriate data file, you can deliberately and gracefully exit the DataBasel program by following the displayed instructions, such as:

Press Return to quit...

If the name of the data file you have entered passes the test made inside the REPEAT ... UNTIL segment, the program continues with code that is identical with the section of code following the ELSE clause listed in the Program UPDATE. When you have finished updating, you press <0> instead of a record number. The Procedure UPDATE is exited, and Procedure ListAll is called.

As with the Procedure UpDate, a REPEAT ... UNTIL segment encloses the I/O error handling and error message statements. In this way, as before, a fatal error is eliminated, and the program persistently repeats the instructions until you make a valid response or deliberately press <Q> to quit. If you do press <Q>, the program gracefully ends, and you are returned to the desktop. The options to display or print the contents of a data file are the same in both the LISTALL procedure and the program.

When you successfully complete and deliberately quit the REPEAT ... UNTIL segment of code given at the end of the main block of code, the data file you have been working with is closed, and a final message tells you:

****** ALL DONE ******

CODE LISTING FOR DATABASE1

(* DataBasel converts three separate procedures *)
(* Start, UpDate, and ListAll *)
(* into a single integrated database manager. *)

PROGRAM DataBasel ;
uses PasPrinter ;

CONST
    MaxNumRecords = 100 ;  { The max of 100 records can be changed here. }

TYPE
    ItemName = String[20] ;
    InStk    = String[20] ;

234
Supp = String[20] ;
Data = String[20] ;

VAR
Item = RECORD
   Name : ItemName ;
   RecordNumber : Integer ;
   InStock : InStk ;
   Supplier : Supp ;
END ;

VAR
Ch : Char ;
DataFile : Data ;
I,Counter : Integer ;
IOerr : Boolean ;
ItemFile : FILE OF Item ;
ItemRec : Item ;

PROCEDURE Start ;
BEGIN  { Start }
   { Position the opening message on the screen at column 1, row 5. }
   GotoXY(1,5) ;
   WriteLn('------------------ START ------------------ ')
   WriteLn
   Write('Enter the name of the data file: ') ;
   ReadLn(DataFile) ;
   WriteLn
   Write('Is this a new file? Y)es or N)o: ') ;
   ReadLn(Ch) ;
   IF (Ch) IN ['Y', 'y'] THEN  { Accept upper or lower case <Y>. }
      BEGIN  { If (Ch) is a <Y>, begin this part. }
         WriteLn
         WriteLn
         WriteLn(DataFile,' is now being created.' )
         WriteLn
         Read(Ch) ;
      END ;
   END ;
END ;
Rewrite(ItemFile,DataFile);  { Open the file. Delete any data in it. }

WITH ItemRec DO
BEGIN
  NAME := '';  { Reserve space for the data. }
  FOR I := 1 TO MaxNumRecords DO
    BEGIN
      RecordNumber := I;
      Write(ItemFile,ItemRec)
    END;
  END;
  WriteLn;
  RecordNumber := I;
  Write(ItemFile,ItemRec)
END;

Close(ItemFile);  { Close the file. }
WriteLn;
WriteLn(' ****** DONE ******');

END
ELSE
END;  { of Start }

PROCEDURE Update;
BEGIN  { Update }
  ClearScreen;
  GotoXY(1,5);
  WriteLn('---------------------- UPDATE ----------------------');
  {$I-}  { Turn off automatic abort in case of I/O error. }
  REPEAT
    WriteLn;
    Write('Enter the name of the data file to update: ');  
    ReadLn(DataFile);
    ReSet(ItemFile,DataFile);
    IOerr := (I0result <> 0);  { Assign a True value to IOerr. }
    IF IOerr THEN
      { IF IOerr is True ... display a message. }
      BEGIN
        WriteLn;
        WriteLn;
        WriteLn('I can''t find that file!');
        WriteLn;
        Write('Press <Q> to Quit, or any other key to continue: ');  
        ReadLn(Ch);
        WriteLn;
        IF (Ch) IN ['Q','q'] THEN
          HALT
      END;
  END;
UNTIL NOT IOerr;
{ Continue if there's no error in the data file's name. }
BEGIN
  WriteLn;
  WriteLn;
  Write('Enter the record number (0 = Stop): ');
  ReadLn(Counter);
  WHILE Counter IN [1..MaxNumRecords] DO
    { While the count is >= 1 and <= 100 ... }
    BEGIN
      Seek(ItemFile,Counter - 1);
      Read(ItemFile,ItemRec);
      WITH ItemRec DO
        BEGIN
          { Collect data for the file's records. }
          Write('Enter the item's name: ');
          ReadLn(Name);
          Write('Enter the qty in stock: ');
          ReadLn(InStock);
          Write('Enter the supplier's ID: ');
          ReadLn(Supplier);
          RecordNumber := Counter;
        END;
      GotoXY(1,7);
      WriteLn;
      { Press <0> to quit the data collection. }
      Write('Enter the item number (0 = Quit): ');
      ReadLn(Counter);
    END;
  Seek(ItemFile,Counter - 1);
  Write(ItemFile,ItemRec);
  GotoXY(1,7);
  WriteLn;
BEGIN
  { ListAll }
  ClearScreen;
  GotoXY(1,5);
  WriteLn('--------------- DISPLAY & PRINT ---------------');
  WriteLn;
  { See ... Procedure UpDate ... for comments. }
  REPEAT
    WriteLn;
    Write('Enter the name of the data file to list: ');
    Read(DataFile);
ReSet(ItemFile,DataFile);
IOerr := (IOresult <> 0);
IF IOerr THEN
BEGIN
WriteLn;
WriteLn;
WriteLn('I can''t find that file!');
WriteLn;
Write('Press <Q> to quit, or any other key to continue: ');
Read(Ch);
WriteLn;
IF (Ch) IN ['Q','q'] THEN
HALT
END;
UNTIL NOT IOerr;
($I+)
{ Continue if there's no error in the datafile's name. }
BEGIN
REPEAT
WriteLn;
Write('Printout or Screen? (S or P): ');
ReadLn(Ch);
UNTIL (Ch) IN ['S','s','P','p']; { Accept only <S> or <P>. }
FOR I := 1 TO MaxNumRecords DO
BEGIN
Read(ItemFile,ItemRec);
WITH ItemRec DO
BEGIN
IF Name <> '' THEN
{ If the record has a name entered... }
BEGIN
{ Send the record to the video display. }
WriteLn;
WriteLn(' * Record Number: ',RecordNumber:3,' ');
WriteLn(' * Item Name: ',Name:20);
WriteLn(' * In stock: ',InStock:20);
WriteLn(' * Supplier: ',Supplier:20);
WriteLn;
END;
IF (Ch) IN ['P','p'] THEN
{ If the printer has been selected... }
BEGIN
{ And if the record has a name entered... }
IF Name <> '' THEN
BEGIN
{ Send the record to the printer. }
WriteLn(Printer);
WriteLn(Printer,' * Record Number: ',RecordNumber:3,' ');
}
WriteLn(Printer,' * Item Name: ',Name:20) ;
WriteLn(Printer,' * In stock: ',InStock:20) ;
WriteLn(Printer,' * Supplier: ',Supplier:20) ;
WriteLn(Printer)
END ;
END ;
END ;
Close(ItemFile) ;   { Close the file that was opened. }
WriteLn ;
WriteLn('       ***** DONE *****') ;
END
END ;
{ ListAll }

BEGIN   { main block of DataBasel }
REPEAT
  Start ;            { Call the Procedure. }
  UpDate ;          { Call the Procedure. }
  ListAll ;         { Call the Procedure. }
  WriteLn ;
  WriteLn('Press <Q> to Quit. Any other key to continue: ') ;
  Read(Ch) ;
  WriteLn('       ***** ALL DONE *****') ;    { That's it. }
  Close(ItemFile) ;   { Close the file. }
  WriteLn(Chr(7))    { Ring the Macintosh bell. }
END.    { This is the "real" END of the DataBasel program }
Chapter 31

DATABASE2: Enancements

to DATABASE1

A database manager program provides many opportunities for customization, practice with writing code, and/or play. I combine all three in this version of the DATABASE1 program presented in Chapter 30. The name I chose is DATABASE2: however, you can rename it to suit your own preferences, of course. I will point out the most significant differences in code between DATABASE1 and DATABASE2. The listing of the code for DATABASE2 is given at the end of this chapter.

How have I "customized" the program? The START, UPDATE, and LISTALL programs in Chapter 29, and the conversions to procedures in the DATABASE1 program in Chapter 30 concerned themselves with a very limited number of fields for an inventory program. But DATABASE2 has wider applications in that it is a file of records of names, street addresses, cities, states, zip codes, and phone numbers. True, some of us prefer to use our pocket or briefcase-size directories; however, the point of this chapter's exercise is to present you with the "fleshed-out skeleton" of a database program whose personal usefulness as an application program depends on your own ingenuity in modifying the code. Now, you have the clues, tips, and guidance needed to customize DATABASE2 to fit your specific requirements.

A CLOSE LOOK AT THE DATABASE2 PROGRAM

The DATABASE2 program creates, updates, and lists (displays and prints) up to 100 records (this number can be changed) in a data file to which you assign an unambiguous name. The listing can be sent to the screen or to the screen and to the printer simultaneously.
The primary operational limitation in the DATABASE1 program is that we must always go sequentially through the three procedures, even if we merely want to update a file or list the contents of a data file. DATABASE2 overcomes that limitation by presenting a menu in the first procedure, Start, from which you can choose <C> to create a new data file of records, <U> to update an existing file or any of its records, <L> to list the records in an existing data file, or <Q> to quit the program and return to the desktop without operating on a data file.

The maximum number of records is, as before, declared as a CONstant with the value of 100. The TYPE declarations have been significantly changed to establish an array of strings of varying length, each with a name for a field that closely resembles the nature of the string. Lengths are also assigned to each of the strings. These lengths can be changed to fit specific types of strings. For example, I assigned a length of 25 characters to the string for field identified as PersonName, while I assigned a length of two to the string for St, the state, and five to the Z, the zip code, and 14 to the string for Ph, phone number.

The labels assigned in the TYPE block are made part of the array Item. They are assigned plain English names, which they carry throughout the program. The names and the assignments in the VAR block are the same as in the DATABASE1 program.

Note that PROCEDURE ListAll and PROCEDURE UpDate must be declared at the start of the program as "Forward." This is because both are called from within the first procedure, Start. The Procedure Start is called from the main block of code (at the end of the code's listing). It clears the screen and presents a menu with four choices: <C> to create a new data file, <U> to update an existing file, <L> to list an existing data file, and <Q> to quit the program immediately.

The response is read from the keyboard and assigned to the character (Ch), and is then tested by a series of IF . . . THEN statements. If the character read at the keyboard passes the tests (that is, if it has been assigned the value of one of the four allowable characters), appropriate action is taken as designated by the BEGIN . . . END groups that follow each IF . . . THEN statement.

If <U> is the response, Procedure UpDate is called and the action then is very much the same as in the DATABASE1 program, complete with I/O error handling. <L> calls the Procedure ListAll. <C>, of course, falls through each of the IF . . . THEN tests, keeps the program in the Procedure UpDate, and activates the WriteLn('Enter the name of the data file: ') statement. Option <Q> brings the program to Turbo Pascal's built-in Halt function, which ends the program and returns us to the desktop. Data files are opened and closed within each of the procedures. Note the brevity of the main BEGIN . . . END block. Its purpose, under the rules of Pascal, is to get the whole thing started.

CODE LISTING FOR DATABASE2

(* DataBase2 embellishes DataBase1 and the *)
(* PROGRAMS Start, UpDate, and ListAll. *)

PROGRAM DataBase2 ;
uses PasPrinter ;
CONST
MaxNumRecords = 100;  { The max of 100 records }
{ can be changed here. }

TYPE
PersonName = String[25];
Addr    = String[25];
Cit     = String[20];
St      = String[2];
Z       = String[5];
Ph      = String[14];
Data    = String[128];
Item    = RECORD
  RecordNumber : Integer;
  Name         : PersonName;
  Address      : Addr;
  City         : Cit;
  State        : St;
  ZIP          : Z;
  Phone        : Ph;
END;

VAR
  Ch         : Char;
  DataFile   : Data;
  ItemFile   : FILE OF Item;
  ItemRec    : Item;
  I,Counter  : Integer;
  IOerr      : Boolean;

Procedure ListAll; Forward;
Procedure UpDate; Forward;

PROCEDURE Start;
BEGIN   { Start }
REPEAT
  ClearScreen;
  { Position the opening on the screen at column 1, row 1. }
  GotoXY(1,1);
  WriteLn('THIS IS "DATABASE2' ... THE DATABASE MANAGER');
  GotoXY(1,3);
  WriteLn('------------- START -------------');
  WriteLn;
  WriteLn('Enter <C> to Create a new data file.');
  WriteLn('Enter <U> to Update an existing data file.');
  WriteLn('Enter <L> to List an existing data file.');
Write('Enter <Q> to Quit this program now: ') ;
ReadLn(Ch) ;

IF (Ch) IN ['U','u'] THEN
BEGIN
    UpDate ;
    ListAll ;
END ;

IF (Ch) IN ['L','l'] THEN ListAll ;

IF (Ch) IN ['Q','q'] THEN EXIT ;

WriteLn ;
WriteLn ;
Write('Enter the name of the data file: ') ;
ReadLn(DataFile) ;

WriteLn ;
WriteLn ;
Write('Is this a new file? Y)es or N)o: ') ;
ReadLn(Ch) ;

IF (Ch) IN ['Y','y'] THEN  { Accept upper or lower case <Y>. }
BEGIN  { If (Ch) is a <Y>, begin this part. }
    WriteLn ;
    WriteLn ;
    WriteLn('READ THE FOLLOWING INSTRUCTIONS CAREFULLY!') ;
    WriteLn ;
    WriteLn('If ',DataFile,' exists, its contents will be erased.') ;
    WriteLn ;
    WriteLn('If you want to create a new file with') ;
    WriteLn('the same name, press the <Y> key.') ;
    WriteLn ;
    WriteLn('To return to the menu without creating a new') ;
    Write(DataFile,' press the Return key: ') ;
    Read(Ch) ;

    IF (Ch) IN ['Y','y'] THEN  { Accept upper or lower case <Y>. }
        BEGIN  { If (Ch) is <Y>, begin this part. }
            WriteLn ;
            WriteLn ;
            WriteLn(DataFile,' is now being created.') ;
            WriteLn ;
            WriteLn ;
ReWrite(ItemFile,DataFile); { Open the file. If it
  { already exists, delete
  { any data that is in it. }
WITH ItemRec
DO
BEGIN
  NAME := ''; { Reserve space for the data. }
  FOR I := 1 TO MaxNumRecords DO
    BEGIN
      RecordNumber := I ;
      Write(ItemFile,ItemRec);
    END;
  END;

  Close(ItemFile); { Close the file. }
END;
UNTIL (Ch) IN ['Q','q'] ;
IF (Ch) IN ['Q','q'] THEN EXIT ; { of Start }

PROCEDURE UpDate ;
BEGIN
  ClearScreen ;
  GotoXY(1,3);
  WriteLn('---------------------- UPDATE -------------------------') ;
  { Turn off the automatic abort in case of I/O error. }
  REPEAT
    WriteLn ;
    Write('Enter the name of a data file to update: ') ;
    ReadLn(DataFile) ;
    ReSet(ItemFile,DataFile) ;
    IOerr := (IOresult <> 0) ; { Assign a True value to IOerr. }
    IF IOerr THEN { IF IOerr is True ... display a message. }
      BEGIN
        WriteLn ;
        WriteLn ;
        WriteLn('I can’’t find ',DataFile) ;
        WriteLn ;
        Write('Press <Q> to Quit, or Return to continue: ') ;
        ReadLn(Ch) ;
        WriteLn ;
        IF (Ch) IN ['Q','q'] THEN EXIT ;
      END ;
END;
UNTIL NOT IOerr;
{S+}
( Continue if there's no error in the data file's name. )
BEGIN
WriteLn;
WriteLn;

Write('Enter the record number (or enter 0 = Stop): '); ReadLn(Counter);

WHILE Counter IN [1..MaxNumRecords] DO
{ While the count is >= 1 and <= 100 ... }
BEGIN
Seek(ItemFile,Counter - 1);
Read(ItemFile,ItemRec);

WITH ItemRec DO
BEGIN { Collect data for the file's records. }
WriteLn;

Write('Enter the name: '); ReadLn(Name);
Write('Address: '); ReadLn(Address);
Write('City: '); ReadLn(City);
Write('State: '); ReadLn(State);
Write('ZIP number: '); ReadLn(ZIP);
Write('Phone number: '); ReadLn(Phone);

RecordNumber := Counter;
END;

Seek(ItemFile,Counter - 1);
Write(ItemFile,ItemRec);
ClearScreen;
GotoXY(1,7);
WriteLn;

{ Press <O> to quit the data collection. }
PROCEDURE Listall;

BEGIN  ( ListAll )
ClearScreen;
GotoXY(1,5);
WriteLn('------------------------ DISPLAY & PRINT ------------------------');
WriteLn;

{ See ... Procedure UpDate ... for comments. }
REPEAT
  WriteLn;
  Write('Enter the name of the data file to list: ');
  ReadLn(DataFile);
  ReSet(ItemFile,DataFile);
  IOerr := (IOresult <> 0);
  IF IOerr THEN
    BEGIN
      WriteLn;
      WriteLn;
      WriteLn('I can''t find that file!');
      WriteLn;
      Write('Press <Q> to Quit, or any other key to continue: ');
      ReadLn(Ch);
      WriteLn;
      IF (Ch) IN ['Q', 'q'] THEN EXIT;
    END;
  END;
  UNTIL NOT IOerr;

{ Continue if there's no error in the datafile's name. }
BEGIN
  REPEAT
    WriteLn;
    WriteLn;
    Write('Printout or Screen? (S or P): ');
    ReadLn(Ch);
    WriteLn;
    IF (Ch) IN ['S', 's'] THEN Printout;
    IF (Ch) IN ['P', 'p'] THEN Screen;
    IF (Ch) IN ['Q', 'q'] THEN EXIT;
  END;
END;

PROCEDURE UpDate;

BEGIN  ( UpDate )
ReadLn(Counter);
END;

PROCEDURE Quit;

BEGIN  ( Quit )
Close(ItemFile);
END  ( of UpDate )

ReadLn(Ch);

UNTIL (Ch) IN ['S','s','P','p'] ; ( Accept only <S> or <P>. )

FOR I := 1 TO MaxNumRecords DO
BEGIN
  Read(ItemFile,ItemRec);
  WITH ItemRec DO
  BEGIN
    IF Name <> ' ' THEN
    { If the record has a Name entry... }
    BEGIN
      { Send the record to the video display. }
      WriteLn;
      WriteLn(' * Record Number: ',RecordNumber:3,' '); 
      WriteLn('-----------------------------------------') ;
      WriteLn(' * Person''s Name: ',Name:25) ;
      WriteLn(' * Address: ',Address:25) ;
      WriteLn(' * City: ',City:20) ;
      WriteLn(' * State: ',State:2) ;
      WriteLn(' * ZIP: ',ZIP:S) ;
      WriteLn(' * Phone Number: ',Phone:14) ;
      WriteLn;
    END ;

    IF (Ch) IN ['P','p'] THEN
    { If the printer has been selected... }
    BEGIN
      IF Name <> ' ' THEN
      BEGIN
        { Send the record to the printer. }
        WriteLn(Printer) ;
        WriteLn(Printer,' * Record Number: ',RecordNumber:3,' ') ;
        WriteLn(Printer,-----------------------------------------') ;
        WriteLn(Printer,' * Person''s Name: ',Name:25) ;
        WriteLn(Printer,' * Address: ',Address:25) ;
        WriteLn(Printer,' * City: ',City:20) ;
        WriteLn(Printer,' * State: ',State:2) ;
        WriteLn(Printer,' * ZIP: ',ZIP:S) ;
        WriteLn(Printer,' * Phone Number: ',Phone:14) ;
        WriteLn(Printer)
      END ;
    END ;
  END ;
END ;
Close(ItemFile);  { Close the file that was opened. }
WriteLn;
WriteLn('****** DONE ******');
WriteLn;
WriteLn;
Write('Press Return to go back to the main menu...  ');
Read(Ch);
END;
Start;
END;  { of ListAll }

BEGIN  { the main part of DataBase2 is very short. }
Start;  { Call the Start Procedure. }
WriteLn(Chr(7));  { Sound the Macintosh's bell. }
END.  { of DataBase2 ... the "real end. " }
Appendix A:

Turbo Pascal
Expressions and Rules

Turbo expressions specify the rules in Turbo Pascal for computing values used in algorithmic constructions.

**OPERATORS AND RULES OF PRECEDENCE**

1) Unary operators: @, not.
2) Multiplying operators: the asterisk character (*), the division character (/), and these words: div, mod, and, shl, shr.
3) Adding operators: the plus sign (+), the minus sign (−), or, xor.
4) Relational operators, which are expressed by the symbols: =, < >, <, >, < =, > =, in.

Operators of equal precedence are recognized from the left to the right sequence in which they appear. Enclosing expressions and the operators within parentheses changes the actual order of computation. The parentheses give the expression they enclose an independence and a precedence over other operators within the same statement.

If the operands of a multiplying and an adding operator are of the Integer type, the result is of the Integer type. If, however, one or both of the operands is of the type Real, the result is of the Real type.

**UNARY MINUS**

The unary minus denotes a negation of its operand, which may be of the Real or Integer type.
THE NOT OPERATOR

The NOT operator inverses or negates the logical value (1 becomes 0, or 0 becomes 1) of its Boolean operand; for example, NOT True = False, and NOT False = True. Turbo Pascal also allows the NOT operator to be applied to an Integer operand. In this case, bitwise, negation occurs, as in these examples:

\[
\begin{align*}
\text{NOT } 0 & \quad = -1 \\
\text{NOT } -15 & \quad = 14 \\
\text{NOT } \$2345 & \quad = \$DCBA
\end{align*}
\]

The NOT operator is a unary operator. The results generated by Boolean type operands are governed by normal Boolean logic. For example, the result of a Boolean operation involving values A and B is true only if both A and B are true.

### MULTIPLYING OPERATORS

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Operand Type</th>
<th>Result Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>multiplication</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>Real, Integer</td>
<td>Real</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>Integer</td>
<td>Real</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>Real, Integer</td>
<td>Real</td>
</tr>
<tr>
<td>div</td>
<td>integer division</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>mod</td>
<td>modulus</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>and</td>
<td>arithmetic and</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>and</td>
<td>logical and</td>
<td>Boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>shl</td>
<td>shift left</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>shr</td>
<td>shift right</td>
<td>Integer</td>
<td>Integer</td>
</tr>
</tbody>
</table>

### ADDITION OPERATORS

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Operand Type</th>
<th>Result Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>+</td>
<td>addition</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>+</td>
<td>addition</td>
<td>Real, Integer</td>
<td>Real</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>Real, Integer</td>
<td>Real</td>
</tr>
<tr>
<td>or</td>
<td>arithmetic or</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>or</td>
<td>logical or</td>
<td>Boolean</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

252
<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Operand Type</th>
<th>Result Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>xor</td>
<td>arithmetic or</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>xor</td>
<td>logical or</td>
<td>Boolean</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

**RELATIONAL OPERATORS**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Result Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>equal to</td>
<td>Boolean</td>
</tr>
<tr>
<td>&lt; &gt;</td>
<td>not equal to</td>
<td>Boolean</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>Boolean</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>Boolean</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>Boolean</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

**PROGRAM HEADINGS**

Pascal programs must have program headings and program blocks. The blocks are divided into declarations and statements.

In Turbo Pascal, the program heading is optional. It is good practice to include the name of the program in the heading, ending with a semicolon, which indicates the end of the statement. For example, if the program’s name is TopScore, the program heading would be written as:

```
Program TopScore;
```

The so-called standard Pascal requires a listing of the parameters through which the program communicates with the environment. The parameters are surrounded by parentheses, and each parameter is separated from the next by a comma, as in this example:

```
Program TopScore(Input,Output);
```

In Turbo, “Input” is assumed to be the keyboard. “Output” is assumed to be the video display, or console. If these assumptions are the desired defaults, the words “Input” and “Output” may be omitted from the program heading.

**BLOCKS**

Immediately following the program heading is a program “block” divided into several parts:

1) Label declaration
2) Constant definition
3) Type definition
4) Variable declaration
5) Procedures and functions
A label declaration is a label-name followed by a colon. It enables the program to branch directly to a statement preceded by the label. The labels must be declared, and each must be separated from other labels in a list by a comma. A semicolon terminates the list of labels. Standard Pascal limits labels to numbers with a maximum of four digits. In Turbo Pascal, numbers and digits can be used as labels. "Label" is a reserved word.

A constant definition allows synonyms to be used as identifiers for constant values. "CONST," a reserved word, heads this list of definitions in the program block. It is followed by a list of constant assignments, each of which is separated from the other by a semicolon. Each constant name-assignment is separated from its assigned value by an equal sign. Turbo Pascal predefines the following values, which can be referenced within a program without having to be listed as a constant:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type and Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pi</td>
<td>Real (3.1415926536E+00).</td>
</tr>
<tr>
<td>False</td>
<td>Boolean (the Truth value is false).</td>
</tr>
<tr>
<td>True</td>
<td>Boolean (the Truth value is true).</td>
</tr>
<tr>
<td>Maxint</td>
<td>Integer (32767).</td>
</tr>
</tbody>
</table>

The reserved word "Type" heads the type section of the program block. It is followed by one or more type assignments separated by semicolons. An equal sign is used to separate each type identifier from its defined type; as in:

```
Quantity = Integer ;
Place = (office,lobby,foyer,hall,elevator) ;
Series = array[1..15] of Real ;
```

Variables must be declared before they can be used. The compiler reports as an error any use of a variable that has not been declared. The reserved word "var" heads the list of variables, each of which is separated from its identifier by a colon. Each line ends with a semicolon. Identifiers of the same type can be listed in sequence and separated from the other by commas. An example is shown below:

```
VAR
    Outward, Inward, Distance : Real ;
    I, N, Q : Integer ;
    Okay, NotOkay : Boolean ;
    Buffer : array[0..127] of Byte ;
```

The user-declared procedure consists of a procedure heading followed by a block which consists of a declaration and a statement. The procedure heading contains the reserved word "procedure" followed by the name of the procedure, which may then be followed by a formal parameter list; the whole heading is terminated by a semicolon. The statements to be performed by the procedure follow the heading and, if used, the parameter list.
“Function” is a reserved word and defines a part of a program that computes and returns a value.

Every Pascal program has a statement part, which is always the last part of any block. The statement part consists of a series of statements followed by a semicolon. A compound statement or series of statements starts with the reserved word BEGIN and is followed by the list of individual statements. The reserved word END followed by a semicolon terminates the statement part. At the very end of a program, a period follows the final use of the reserved word END.
Appendix B:

Turbo Pascal
and Standard Pascal

The interest in Turbo Pascal as a language/compiler for serious applications programs is growing rapidly. Turbo Pascal offers a number of extensions to Pascal as specified by the ANSI/IEEE standard 770X3.97-1983 and often referred to as "standard Pascal." Several important differences are described below. In Turbo Pascal:

- An identifier may contain underscore characters after the first character.
- Integer constants may be written in hexadecimal notation prefixed by a dollar sign, $.
- Identifiers may be used as labels.
- String constants are compatible with the Turbo Pascal string-types. They may contain control characters and other nonprinting characters.
- Label, constant, type, variable, procedure, and function declarations may be used any number of times and in any order in a block.
- The $\oplus$ symbol is an operator, never treated identically with the $\wedge$ (caret) symbol.
- Comments must begin and end with the same set of symbols, either a matching pair of braces (curly brackets) or left- and right-going complementary sets of parentheses and asterisks.
- It is not necessary for each possible value of the tag-type in a variant-part to appear once, as is the requirement in ANSI Pascal.
- In ANSI Pascal, the component-type of a file-type may not be a structured-type having a component of a file-type. This is not enforced in Turbo Pascal.
- Buffer-variables are not associated with file-variables in Turbo Pascal; writing
the ^ (caret) symbol after a file-variable flags an error code during compilation.
- It is not necessary that the statement-part of a function contain at least one assignment to the function identifier, as is required in ANSI Pascal.
- A field that is the selector of a variant-part may not be an actual variable parameter in ANSI Pascal. Turbo Pascal does not enforce this requirement.
- Turbo Pascal procedures and functions do not allow procedural and functional parameters.
- In ANSI Pascal, the standard procedures Reset and Rewrite take only one parameter, a file variable. In Turbo Pascal, however, Reset and Rewrite must also have a second parameter, a string-type expression that names an external file.
- Get and Put are defined as standard procedures to read from and write to files in ANSI Pascal. Get and Put are not defined in Turbo Pascal.
- In ANSI Pascal, the standard procedures Read and Write are defined in terms of Get and Put and references to buffer-variables. In Turbo Pascal, Read and Write perform the same tasks but they are automatic operations.
- The syntax New(p,c1, . . . ,cn), which creates a dynamic variable with a specific active variant in ANSI Pascal, is not allowed in Turbo Pascal.
- In ANSI Pascal, the syntax Dispose(q,k1, . . . ,kn) removes a dynamic variable with a specific active variant. This syntax is not allowed in Turbo Pascal.
- The standard procedures Pack and Unpack, which are used to pack and unpack packed variables in ANSI Pascal, are not defined in Turbo Pascal.
- In ANSI Pascal, the term i mod j always computes a positive value; it is an error if j is zero or negative. In Turbo Pascal, i mod j is computed as i - (i div j) * j; it is not an error if j is negative.
- ANSI Pascal allows a goto statement within a block to refer to a label in an enclosing block. In Turbo Pascal, this is flagged as an error.
- It is an error in ANSI Pascal if the value of the selector in a case statement is not equal to any of the case-constants. In Turbo Pascal this is not an error; however, the case statement is ignored if it does not contain an otherwise clause.
- In ANSI Pascal, statements that threaten the control-variable of a FOR statement are not allowed; this is not enforced in Turbo Pascal.
- In ANSI Pascal, a Read from a text-file with a char-type variable assigns a blank to the variable if Eoln was true before the Read. Turbo Pascal assigns a carriage-return character (ASCII 13) to the variable.
- In ANSI Pascal, a Read from a text file with an integer-type or a real-type variable terminates as soon as the next character in the file is not part of a signed-integer or a signed-number. Reading is terminated in Turbo Pascal when the next character in the file is a blank or a control character (including the end-of-line character).
- In ANSI Pascal, a Write to a text file with a packed-string-type value causes the string to be truncated if the specified field width is less than the length of the string. In Turbo Pascal, the string is always written in full, even when it is longer than the specified field width.

**RESERVED WORDS**

The following are Turbo Pascal's RESERVED words. They cannot be used as
labels and identifiers beyond the methods that are in Turbo Pascal’s formal rules of syntax (listed alphabetically across):

<table>
<thead>
<tr>
<th>and</th>
<th>array</th>
<th>begin</th>
<th>case</th>
<th>const</th>
</tr>
</thead>
<tbody>
<tr>
<td>div</td>
<td>do</td>
<td>downto</td>
<td>implementation</td>
<td>for</td>
</tr>
<tr>
<td>in</td>
<td>else</td>
<td>external</td>
<td>file</td>
<td></td>
</tr>
<tr>
<td>forward</td>
<td>function</td>
<td>goto</td>
<td>if</td>
<td>otherwise</td>
</tr>
<tr>
<td>packed</td>
<td>inline</td>
<td>in</td>
<td>label</td>
<td>interface</td>
</tr>
<tr>
<td>mod</td>
<td>nil</td>
<td>not</td>
<td>of</td>
<td>or</td>
</tr>
<tr>
<td>then</td>
<td>to</td>
<td>program</td>
<td>record</td>
<td>procedure</td>
</tr>
<tr>
<td>repeat</td>
<td>set</td>
<td>shl</td>
<td>shr</td>
<td>string</td>
</tr>
<tr>
<td>type</td>
<td>unit</td>
<td>until</td>
<td>uses</td>
<td>var</td>
</tr>
<tr>
<td>while</td>
<td>with</td>
<td>xor</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Here is a list of RESERVED words in standard Pascal:

<table>
<thead>
<tr>
<th>and</th>
<th>array</th>
<th>begin</th>
<th>case</th>
<th>const</th>
<th>div</th>
</tr>
</thead>
<tbody>
<tr>
<td>do</td>
<td>downto</td>
<td>else</td>
<td>end</td>
<td>file</td>
<td>for</td>
</tr>
<tr>
<td>function</td>
<td>goto</td>
<td>if</td>
<td>in</td>
<td>label</td>
<td>mod</td>
</tr>
<tr>
<td>all</td>
<td>not</td>
<td>of</td>
<td>or</td>
<td>packed</td>
<td>procedure</td>
</tr>
<tr>
<td>program</td>
<td>record</td>
<td>repeat</td>
<td>set</td>
<td>then</td>
<td>to</td>
</tr>
<tr>
<td>type</td>
<td>until</td>
<td>var</td>
<td>while</td>
<td>with</td>
<td></td>
</tr>
</tbody>
</table>

**SPECIAL SYMBOLS**

It is important to know the following rules, which apply to the use of characters from the ASCII character set in writing Turbo Pascal code:

- The English alphabet, uppercase and lowercase letters A through Z.
- The Arabic numerals 0 through 9.
- The Hex digits are restricted to Arabic numerals 0 through 9, and uppercase and lowercase letters A through Z.
- All ASCII control characters (ASCII 0 through 31), and the space character (ASCII 32).

**FILE-HANDLING PROCEDURES AND FUNCTIONS**

Turbo Pascal implements the following file-handling procedures and functions, which are not available in ANSI Pascal:

<table>
<thead>
<tr>
<th>close</th>
<th>erase</th>
<th>FilePos</th>
<th>FileSize</th>
<th>IOResult</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rename</td>
<td>SeekEof</td>
<td>Seek</td>
<td>SeekEoln</td>
<td></td>
</tr>
</tbody>
</table>

In Turbo Pascal, string-type variables may be input and output with the Read, ReadLn, Write, and WriteLn standard procedures.

Turbo Pascal assigns two standard devices, Console and Printer, and supports user-defined devices.
Turbo Pascal for the Macintosh implements these standard (built-in) procedures and functions:

<table>
<thead>
<tr>
<th>Exit</th>
<th>Halt</th>
<th>New</th>
<th>Dispose</th>
</tr>
</thead>
<tbody>
<tr>
<td>MemAvail</td>
<td>MaxAvail</td>
<td>Chr</td>
<td>Ord</td>
</tr>
<tr>
<td>Ord4</td>
<td>Pointer</td>
<td>Trunc</td>
<td>Round</td>
</tr>
<tr>
<td>Float</td>
<td>Abs</td>
<td>Sqr</td>
<td>Int</td>
</tr>
<tr>
<td>Sqrt</td>
<td>Sin</td>
<td>Cos</td>
<td>Exp</td>
</tr>
<tr>
<td>Ln</td>
<td>ArcTan</td>
<td>Succ</td>
<td>Pred</td>
</tr>
<tr>
<td>Odd</td>
<td>Length</td>
<td>Pos</td>
<td>Concat</td>
</tr>
<tr>
<td>Copy</td>
<td>Delete</td>
<td>Insert</td>
<td>ClearScreen</td>
</tr>
<tr>
<td>ClearEOL</td>
<td>GotoXY</td>
<td>InsertLine</td>
<td>DeleteLine</td>
</tr>
<tr>
<td>ReadChar</td>
<td>SizeOf</td>
<td>MoveLeft</td>
<td>MoveRight</td>
</tr>
<tr>
<td>FillChar</td>
<td>ScanEQ</td>
<td>KeyPressed</td>
<td>ScanNE</td>
</tr>
<tr>
<td>Hi</td>
<td>Lo</td>
<td>HiWord</td>
<td>LoWord</td>
</tr>
<tr>
<td>Swap</td>
<td>SwapWord</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

260
Appendix C:

Identifiers, Reserved Words, Constants, Variables, and Expressions

Identifiers are the names assigned to items such as constants and variables. In standard Pascal, an identifier must start with a letter. It may be followed by additional letters and digits, within the limitations of the specific compiler used by the programmer. As with several other compilers, Turbo Pascal has a capability that can be very helpful in writing or inventing identifier-labels that are easily read by the unaided human eye. With Turbo Pascal we are able to use the underline character (_) anywhere a letter can be used. For example, here’s an identifier—not an unusual one, but deliberately long to emphasize the point—as it might appear in standard Pascal:

\[\text{Fetchandfindtherightdigit}\]

Of course, this is rather hard to read and will slow you down a bit. With Turbo Pascal, you can write the same identifier as:

\[\text{Fetch\_and\_find\_the\_right\_digit}\]

The improvement is obvious. Many programmers choose to write long identifiers in a combination of uppercase and lowercase letters. Using the same set of words as in our previous example, this is illustrated here:

\[\text{FetchAndFindTheRightDigit}\]
Or, in Turbo Pascal the same identifier might be written as:

```
Fetch_And_Find_The_Right_Digit
```

Turbo Pascal identifiers are not case sensitive. Uppercase and lowercase letters can be intermixed within a word or a sentence without confusing the compiler. For example, as far as the compiler is concerned, these four identifiers are the same:

```
NICEGOING
NiceGoing
nicegoing
nIcEgOiNg
```

The maximum length for an identifier name in Turbo Pascal for the Macintosh is 63 characters, counting letters, numbers, and underscores. This is the maximum length that the Macintosh operating system allows. An identifier that is 63 characters long has to be ranked among the rarest occurrences in the nature and science of Pascal programming.

**RESERVED WORDS**

All versions and variations of Pascal use special identifiers called *reserved* or *key* words. These special identifiers are part of the syntax of the Pascal programming language. This means they cannot be used for any purpose within the source code of a program other than the one for which they are intended. They cannot be used to name programs, constants, or data types.

You have encountered many reserved words in the programs in this book. For example, the word `PROGRAM` is used at the start of every program's source code; not just in this book, but in the universe of Pascal syntax. `PROGRAM` cannot be used as an identifier anywhere else in a Pascal program. It can be used as part of a string of characters that is to be sent to the console for viewing or to a printer, for example. In such cases, `PROGRAM` and all the string's characters must be surrounded by single quote marks.

Reserved words for Turbo Pascal and standard Pascal are listed in Appendix B.

**CONSTANTS**

Certain words have been given or have acquired standard values that cannot be changed. For example, `Pi`, the mathematical term has one value (see Appendix B) referred to as a *constant*.

When you want to change the value of an identifier, it is declared to be a *variable*, which is the opposite of a *constant*. A CONSTANT can be one of seven types:

- integer
- char
- string
- byte
Only the first five CONStants in the above list are the standard predefined types used in Turbo Pascal.

**VARIABLES**

At times it is necessary to work with an identifier whose value is either unknown or that might be changed during the program’s operation. The process is to create a name for an identifier and declare it to be one of the types listed above. The *variable* identifiers are called VARs, simply because the values assigned to them can be made to vary (new values can be assigned to them) inside a program.

**EXPRESSIONS**

An *expression* is sometimes compared with a mathematical formula. It may consist of a number of assorted identifiers of compatible types connected by operators or symbols. The program works rapidly to evaluate the relationships of the identifiers and, on the basis of the operators, combine the values assigned to the identifiers and assign a single value to the combination.

To illustrate the point, assume you have three variable identifiers: A, B, C. You’ve declared them to be of the integer type. Within the program, each has been assigned a value, A := 5, B := 3, and C := 1.

The program is using the expression:

\[(A + B - C) \times 7\]

According to the values that have been assigned to the variables, the program evaluates the above expression as:

\[(5 + 3 - 1) \times 7\]

Of course, the result of this calculation is 49. The value of the entire expression can be assigned to another variable of the same type as A, B, and C. It can be assigned to D, for example, which then acquires the value of 49. Its value can be assigned to any one of the declared variables A, B, or C, which changes the variable’s value accordingly, as in these two examples:

\[(A + B - C) \times 7 := D\]

or

\[(A + B - C) \times 7 := B\]

In the first equation the variable D is assigned the value of 49; B retains the value of 3. In the second equation, after the calculations, B is assigned the value of 49.
Appendix D:

Predefined Procedures and Functions

Among the features that distinguish one compiler from another, though both may be intended for the "same" language, is the compiler's own standard library of predefined procedures and functions. A compiler's library is designed to simplify the task of programming, as the designer of the compiler visualizes the task. The procedures and functions of a compiler's library may be referred to as "standard" or "built-in." The compiler recognizes them by their identifier-names and takes the necessary actions. This significantly relieves the programmer of the requirement to write formulas or detailed, often-complex lines of code each time a recurring action or calculation is called by the program's code.

Turbo Pascal for the Macintosh provides a set of predefined procedures and functions that can be called by name from within the source code written by the programmer. The following lists the "built-ins" of Turbo Pascal, Version 1.0.

**PREDEFINED PROCEDURES**

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close(F)</td>
<td>Close file F.</td>
</tr>
<tr>
<td>ClearEOL</td>
<td>Clear the screen to the end of the line on which the cursor is positioned.</td>
</tr>
<tr>
<td>ClearScreen</td>
<td>Clear the entire screen.</td>
</tr>
<tr>
<td>Delete(S)</td>
<td>Delete section of string S.</td>
</tr>
<tr>
<td>DeleteLine</td>
<td>Delete the line on which the cursor is positioned.</td>
</tr>
<tr>
<td>Dispose(P)</td>
<td>Recover memory used by P.</td>
</tr>
<tr>
<td>Eof</td>
<td>End of file marker.</td>
</tr>
</tbody>
</table>
Erase(F) ................................ Delete file F from the directory.
Exit ................................ Exit immediately from the current block.
FillChar(V,L,D) .................. Fill V with data D for L bytes.
Goto(X,Y) ....................... Move the cursor to column X, row Y (1,1 is the upper left corner position).
Halt ................................ Stop the program and return to the desktop.
Insert(S,D,P) ................... Insert string D into S.
InsertLine ........................ Insert a line on the video display.
MoveLeft ........................ Copy bytes from source to destination (starting at the lowest address).
MoveRight ........................ Copy bytes from source to destination (starting at the highest address).
New(P) .......................... Create or reserve memory for P.
Read(X) .......................... Read item in from the keyboard.
ReadLn(X) ....................... Same as Read(X) above, but move to a new line at the end of the read.
Release(P) ...................... Reset the heap pointer to P.
Rename(F) ........................ Rename file F.
Reset(F,n) ...................... Open file F for input.
Rewrite(F,n) .................... Open file F for output.
Seek(F,P) ........................ Move to record P in file F.
Str(N,S) ........................ Convert number N to string S.
Val(S,N,P) ..................... Convert string S to number N (error at P).
Write(X) ........................ Write items out to the screen.
WriteLn(X) ..................... Same as Write(X), but starts a new line after the Write expression is completed.

PARTIAL LIST OF PREDEFINED FUNCTIONS
Abs(A) .......................... Absolute value of A (Real, Integer).
Annuity .......................... Used in calculations of financial data.
ArcTan(X) ........................ Arctangent of X (Real).
Chr(I) .......................... Character with the ASCII value of I (Char).
Compound ........................ Returns the compound interest.
Concat(S,..,S) .................. Concatenate the strings (String).
Copy(S,P,L) ..................... Substring of P length L (String).
CopySign(x,y) ................... Returns the value of y with the sign of x.
Cos(X) .......................... Cosine of X (Real).
Eof(F) .......................... Test file F for End-Of-File (Boolean).
Eoln(F) .......................... Test file F for End-Of-Line (Boolean).
Exp(X) .......................... Exponential of X (Real).
FilePos(F) ...................... Current record in file F (Integer).
FileSize(F) ..................... Total records in file F (Integer).
Float ........................... Convert integer-type to real-type value.
GetPrecision ................... Returns the current rounding precision.
GetRound ........................ Returns the current rounding direction.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hi(I)</td>
<td>Upper byte of I (Integer).</td>
</tr>
<tr>
<td>HiWord(X)</td>
<td>Returns high-order word of argument (X).</td>
</tr>
<tr>
<td>Int(X)</td>
<td>Integer portion of X (Real).</td>
</tr>
<tr>
<td>KeyPressed</td>
<td>Keyboard status flag (Boolean).</td>
</tr>
<tr>
<td>Length(S)</td>
<td>Length of string S (Integer).</td>
</tr>
<tr>
<td>Ln(X)</td>
<td>Natural logarithm of X (Real).</td>
</tr>
<tr>
<td>Lo(I)</td>
<td>Lower byte of I (Integer).</td>
</tr>
<tr>
<td>LoWord(X)</td>
<td>Returns the low-order word of argument (X).</td>
</tr>
<tr>
<td>MaxAvail</td>
<td>Returns the size of largest contiguous free block in the heap.</td>
</tr>
<tr>
<td>MemAvail</td>
<td>Bytes/paragraphs available (Integer).</td>
</tr>
<tr>
<td>Odd(I)</td>
<td>Odd/even test of I (Boolean).</td>
</tr>
<tr>
<td>Ord(Sc)</td>
<td>Ordinal value of scalar variable (Integer).</td>
</tr>
<tr>
<td>Ord4(X)</td>
<td>Ordinal value of ordinal or pointer value.</td>
</tr>
<tr>
<td>Pred(X)</td>
<td>Predecessor of the argument X.</td>
</tr>
<tr>
<td>Pointer(X)</td>
<td>Integer-type value to pointer-type value.</td>
</tr>
<tr>
<td>RandomX</td>
<td>Uses an algorithm for randomizing.</td>
</tr>
<tr>
<td>ReadChar</td>
<td>Wait for a key to be pressed.</td>
</tr>
<tr>
<td>Rint(x)</td>
<td>Rounds x to an integral value.</td>
</tr>
<tr>
<td>Round(x)</td>
<td>Rounded-off value of real; becomes integer.</td>
</tr>
<tr>
<td>Scalb(x)</td>
<td>Scales x by the power of two.</td>
</tr>
<tr>
<td>ScanEQ(Ch)</td>
<td>Scan a range of bytes for a (Ch) value.</td>
</tr>
<tr>
<td>ScanNE(Ch)</td>
<td>Scan a range for a non-matching (Ch) value.</td>
</tr>
<tr>
<td>SeekEof</td>
<td>Returns end-of-file status of a file.</td>
</tr>
<tr>
<td>SeekEoln</td>
<td>Returns end-of-line status of a file.</td>
</tr>
<tr>
<td>Sin(X)</td>
<td>Sine of X (Real).</td>
</tr>
<tr>
<td>SizeOf(X)</td>
<td>Size in bytes of variable X.</td>
</tr>
<tr>
<td>Sqr(A)</td>
<td>Square of A (A * A) (Real, Integer).</td>
</tr>
<tr>
<td>Sqrt(A)</td>
<td>Square root of A (Real).</td>
</tr>
<tr>
<td>Str2Num</td>
<td>Converts a decimal string argument.</td>
</tr>
<tr>
<td>Succ(Sc)</td>
<td>Successor of scalar value (same type).</td>
</tr>
<tr>
<td>Swap(I)</td>
<td>Swap upper with lower bytes of I (Integer).</td>
</tr>
<tr>
<td>SwapWord</td>
<td>Swaps high- and low-order words.</td>
</tr>
<tr>
<td>TestHalt</td>
<td>Returns true if halt flags are set.</td>
</tr>
<tr>
<td>TextException</td>
<td>Returns true if exceptions are set.</td>
</tr>
<tr>
<td>Trunc(X)</td>
<td>Truncated value of real; becomes integer.</td>
</tr>
<tr>
<td>XpwrI</td>
<td>Returns x raised to the power of i.</td>
</tr>
<tr>
<td>XpwrY</td>
<td>Returns x raised to the power of y.</td>
</tr>
</tbody>
</table>

**TURTLEGRAPHICS**

A graphics program, TURTLE, is included with Borland’s Turbo Pascal. It is based on a concept developed at the Massachusetts Institute of Technology that basically concerns a “turtle” and its ability to move a given distance and change its direction at specified points. Thus, it draws lines as it “walks” along. The images that are created can be significantly more interesting and simpler to produce than those generated by
algorithms which declare cartesian coordinates.

**TURTLECIRCLE**

A brief example of a TURTLEGRAPHICS program is given in the source code for TURTLECIRCLE, Fig. D-1. (Note the uses clause at the second line of the code. It must be included to call up the appropriate interface units.)

**CODE LISTING FOR TURTLECIRCLE**

```pascal
PROGRAM TurtleCircle ;
uses MemTypes,QuickDraw,OSIntf,ToolIntf,Turtle ;

VAR
  Angle : Integer ;

BEGIN
  PenDown ;
  SetPosition(75,20) ;
  FOR Angle := 0 to 89 DO 
    BEGIN
      TurnRight(4) ;
      Forwd(-5) ;
    END ;
  ReadLn ;
END .
```

Fig. D-1. The source code for a Turtle Graphics program that draws a circle.

When TURTLECIRCLE is executed, the Macintosh displays a simple circle, which is shown in the actual printout reproduced in Fig. D-2.

**TURTLEPATTERN**

A variation of TURTLECIRCLE produces the pattern illustrated in Fig. D-3, TURTLEPATTERN. The code for this program is given in Fig. D-4.

**CODE LISTING FOR TURTLEPATTERN**

Although comprehensive applications for graphics such as those that can be created with TURTLEGRAPHICS are beyond the intended scope of this book, I have listed 16 TURTLEGRAPHICS procedures below which are standard in Turbo Pascal.

**PREDEFINED TURTLEGRAPHICS PROCEDURES**

<table>
<thead>
<tr>
<th>Back</th>
<th>Home</th>
<th>SetHeading</th>
<th>TurtleDelay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear</td>
<td>NoWrap</td>
<td>SetPosition</td>
<td>Wrap</td>
</tr>
<tr>
<td>Forward</td>
<td>PenDown</td>
<td>TurnLeft</td>
<td>Xcor</td>
</tr>
<tr>
<td>Heading</td>
<td>PenUp</td>
<td>TurnRight</td>
<td>Ycor</td>
</tr>
</tbody>
</table>
Fig. D-2. Printout of the screen's display created when the code for TURTLECIRCLE is run.

Fig. D-3. Printout of the screen's display created when the code for TURTLEPATTERN is run.
PROGRAM TurtlePattern;
uses MemTypes,QuickDraw,OSIntf,ToolIntf,Turtle;

VAR
  Angle : Integer;

BEGIN
  PenDown;
  SetPosition(75,0);
  FOR Angle := 0 to 89 DO
    BEGIN
      TurnRight(75);
      Forwd(-100);
    END;
  ReadLn;
END.

Fig. D-4. The source code for the TURTLEPATTERN program.
Appendix E:

I/O Error Handling

If you want to use I/O error handling more frequently than in the programs given in this book or if you want to expand and increase its use, the following examples can provide some guidance.

Assume that a program queries you for the name of a file, and you type in a name, but you misspell it, make a typo, or enter the name of a file that doesn’t exist in the disk’s directory. Without I/O error handling, the program would abort on an I/O error, and return you to the Macintosh desktop. If you’ve entered data, it might not be saved, or it might be scrambled. Turbo Pascal solves this problem with specific compiler directives.

You can disable the "abort on an I/O error" message and its action in sections of code or for the entire program by turning off the I/O error-handling feature. You do this by inserting the compiler directive {`$I-`}, a switch that looks like a comment. But, very significantly, the dollar sign adjacent to the left brace and adjacent to the uppercase letter I makes the difference. Write this compiler directive into your program’s source code at the point you want to defeat the automatic abort process.

The I/O error handling with its automatic abort-on-error process can be turned back on by inserting another switch that cancels the first; type `{`$I+`}`, and press the Return key. An example is shown in Fig. E-1.

Note again, for emphasis, that it is the $ sign that makes the I- and I+ behave as compiler directives rather than as comments to be ignored by the compiler. The default value for I/O error handling is `{`$I+`}`. It is on unless you deliberately switch it off by appending a minus sign immediately following the combination $I. A plus sign,
REPEAT
  Write('Type the name of the file: ');
  ReadLn(Name_It);
  Reset(In_File,Name_It);
  IOerr := (IOresult <> 0);
  IF IOresult > 0 THEN
    WriteLn('I can't find that file. Try it again.' )
  UNTIL NOT IOerr ;

Fig. E-1. An example of code that makes use of a compiler switch to display a message and prevent the program from crashing in the event of an I/O error.

instead of a minus sign, appended to the $I returns the directive to its default status.

With the above example of I/O error checking and trapping appropriately inserted in your program's source code, if for any reason you type the name of a file that is not in the disk's directory, the program would not abort but would display:

   I can't find that file. Try it again.

IOerr must be declared as a VAR of the type Boolean. Therefore, IOerr remembers there was an I/O error, even after IOresult has been reset to 0.

With this information, try revising some of the programs that ask for filename inputs. Include the above code sample. Select an error message that uses your own wording. Several chapters do provide examples from which you can learn, and whose I/O error-checking segments you may want to extract and use in your own program code.
## Index

| A      | capital assets, 53  |
|        | CASE statements, 113, 139 |
|        | cash in float, 33     |
|        | character strings, 13 |
|        | CHECKBOOK program, 41-52, 41 |
|        | clause, 17            |
|        | COINCHANGER program, 14, 16 |
|        | collection period, 35 |
|        | comments, 5, 257      |
|        | commissions, 173      |
|        | comparisons, 34       |
|        | compiling, 3          |
|        | constant value, 12    |
|        | constants, 261, 282   |
|        | contribution analysis, 27-32 |
|        | CONTRIBUTION program, 27-32 |
|        | control codes, 113    |
|        | conversions of English to metric, 139 |
|        | copying ASCII files, 199 |
|        | COPYIT program, 199-200 |
|        | COUNTERCHARS program, 65, 70 |
|        | counting programs, 65-72 |
|        | COUNTSTROKES program, 65, 69 |
|        | COUNTWORDS program, 65, 67-69 |
|        | crash, 2              |
|        | database program, three-part, 223 |
|        | DATABASE1 program, 233-240 |
|        | DATABASE2 program, 241-250 |
|        | DAYS CREDIT program, 33-39 |
|        | debugging, 2          |
|        | delimiters, 5         |
|        | depletion allowances, 54 |
|        | DEPRECIATION program, 53-64 |
|        | devices, directing information to, 14 |
|        | displaying text files, 207 |
|        | documentation, 2      |
|        | errors, 110, 271      |
|        | EDITIT programs, 215-222 |
|        | English/metric conversions, 139 |
|        | enhancements, 7       |
|        | equidistant data points, 129 |
|        | error handling, 233   |
|        | I/O, 271              |
|        | error trapping, 2, 19, 140 |
|        | ERRORTRAP program, 19 |
|        | evaluating investments, 73 |
|        | expense accounts, 97  |
|        | expressions, 251, 261, 263 |
| B      | balance sheets, 88    |
|        | banner, 43            |
|        | blocks, 253           |
|        | Boolean operator, 252 |
|        | break-even point, 163 |
|        | BREAKEVEN program, 163-172 |
|        | buffer-variables, 257 |
|        | calculator, 187       |
|        | calls, 17             |
|        | data points, 129      |
|        | database program, three-part, 223 |
|        | DATABASE1 program, 233-240 |
|        | DATABASE2 program, 241-250 |
|        | DAYS CREDIT program, 33-39 |
|        | debugging, 2          |
|        | delimiters, 5         |
|        | depletion allowances, 54 |
|        | DEPRECIATION program, 53-64 |
|        | devices, directing information to, 14 |
|        | displaying text files, 207 |
|        | documentation, 2      |
|        | errors, 110, 271      |
|        | EDITIT programs, 215-222 |
|        | English/metric conversions, 139 |
|        | enhancements, 7       |
|        | equidistant data points, 129 |
|        | error handling, 233   |
|        | I/O, 271              |
|        | error trapping, 2, 19, 140 |
|        | ERRORTRAP program, 19 |
|        | evaluating investments, 73 |
|        | expense accounts, 97  |
|        | expressions, 251, 261, 263 |
|        | file-handling procedures and functions, 259 |
|        | file-type, 257        |
financial net worth, 87
FOR . . . DO loop, 12
four-function calculator, 187
functions, 257, 258
file-handling, 259
predefined, 265
G
GOTO statement, 258
GOTO XY function, 217
graphics, 11
graphics presentations, 23-26
graphs, plotting on, 129
H
halo effect, 153
headings, 253
HISTOGRAM program, 23-26
histograms, 23-26
I
I/O error handling, 233, 271
identifiers, 257, 261
IF . . . THEN clause, 187
ImageWriter printout, 17
integer constants, 257
interest payments, 79
INVESTAL program, 73-78
Investment evaluation, 73
L
labels, 257
late payers, classification of, 34
liquidity, 34
LISTALL program, 223, 229-232
loan table generator, 79
LOANPAYMENTS program, 79-86
M
METRICENGLISH program, 139-146
MINICALC program, 187-190
mortgage table generator, 79
moving average, calculation of, 119
multiplying operators, 252
N
naming programs, 6
near letter quality mode, 17
NETWORTH program, 87-96
NOT operator, 252
number handling, 14
O
operators, 251
P
PASSWORD program, 20
passwords, 20
personnel appraisal, 153
plant asset, 53
PLOTTER program, 129-138
predefined functions, 265
predefined procedures, 265
PRINTTEST1-3 programs, 191-198
printing, 17, 113, 201, 211
quality control of, 191
PRINTIT programs, 201-206
procedures, 9, 257, 258
file-handling, 259
predefined, 265
profits, 163
program compiling, 3
program creation, 3
program enhancement, 7
program headings, 253
programs, 5-22
APPRAISAL, 153-162
AUTOREPORT, 97
BREAKEVEN, 163-172
CHECKBOOK, 41-52
COINCHANGER, 14
CONTRIBUTION, 27-32
COPYIT, 199-200
COUNTCHARS, 65, 70
COUNTSTROKES, 65, 69
COUNTWORDS, 65, 67-69
DATABASE1, 233-240
DATABASE2, 241-250
DAYS CREDIT, 33-39
DEPRE CIATION, 53-64
EDIT IT, 215-222
ERRORTRAP, 19
HISTOGRAM, 23-26
INVESTAL, 73-78
LISTALL, 229-232
LOAD PAYMENTS, 79-86
METRIC ENGLISH, 139-146
MINICALC, 187-190
NETWORTH, 87-96
PASSWORD, 20
PLOTTER, 129-138
PRINTTEST, 191-198
PRINTIT, 201-206
REPCOMMISSION, 173-181
SALESREPORT, 181-186
SET PRINTER, 113-118
SHOWIT, 207-210
SIMPLEINTSUM, 13
S IMPLERE A LSUM, 13
SORTER, 147-152
START, 223-226
TIMETREND, 119-128
TRIPREPORT, 106-111
TURTLE, 267
TYPEIT, 211-214
UP DATE, 226-229
R
raw data, 121
real estate, depreciation of, 54
relational operators, 253
renaming ASCII files, 199
REPCOMMISSION program, 173-181
REPEAT . . . UNTIL loop, 140, 233
report format, 88
reserved words, 258, 261, 262
return on investment, 97
revenues vs. costs, 163
rules, 251
rules of precedence, 251
S
sales performance analysis, 181
sales representatives, commissions
for, 173
SALESREPORT program, 181-186
scrap value, 54
SET PRINTER program, 113-118
SHOWIT program, 207-210
SIMPLEINTSUM, 13
SIMPLEREALSUM, 13
smoothed data, 121
SORTER program, 147-152
source code, 2
special symbols, 259
START program, 223-226
statements, 258
step-function order, 23
string constants, 257
strings, 7, 257
syntax, 258
T
text editor, 215
text file, display of, 207
TIMETREND program, 119-128
time-trend analysis, 119
trapping errors, 19
trends, 34, 119
monitoring of, 42
trip expense report generator, 97
TRIPREPORT program, 97, 106-111
Turbo Pascal
expressions and rules for, 251
introduction to, 1-4
standard Pascal vs., 257-260
turnover ratio, 34
TURTLE CIRCLE program, 268
TURTLE GRAPHICS program, 267
TURTLE PATTERN program, 268
type, 257
TYPEIT program, 211-214
typing to printer, 211

274
unary minus, 251
UPDATE program, 223, 226-229
user interaction, 8
USES clause, 17
utilities, 2
utility programs, 201
values, 12
variable costs, 27
variable string, 7
variables, 12, 257, 261, 262, 263
WRITE function, 217
WRITELN function, 217
Other Bestsellers From TAB

- **TROUBLESHOOTING AND REPAIRING THE NEW PERSONAL COMPUTERS**—Art Margolis
  A treasury of time- and money-saving tips and techniques that shows personal computer owners and service technicians how to troubleshoot and repair today's new 8- and 16-bit computers (including the IBM PC/XT/AT and compatibles, the Macintosh, the Amiga, the Commodores, and other popular brands). Leading off with an invaluable Trouble-Symptoms guide that focuses on the most common microcomputer problems, Margolis examines the symptom, describes the problem, indicates which chips or circuits are most likely to be the source of the trouble, and directs you to the solution! 416 pp., 343 illus., 7" x 10".
  Paper $17.95 Hard $22.95
  Book No. 2809

- **SUPERCOMPUTERS OF TODAY AND TOMORROW: THE PARALLEL PROCESSING REVOLUTION**—Richard A. Jenkins
  A view of future technology from the birth of modern computer science to the development of today's most advanced fifth generation computers. You'll discover what technological breakthroughs are being made in parallel processing and artificial intelligence research and their expected impact on the worlds of science and industry. 208 pp., 26 illus.
  Paper $14.95 Hard $21.95
  Book No. 2622

- **MACINTOSH™ ASSEMBLY LANGUAGE PROGRAMMING**
  Delve below the surface-level capabilities of your Macintosh and discover the incredible power that assembly language can unlock. You'll learn all about the fundamentals of machine code ... gain an understanding of editors and assemblers ... and tap into the 68000's addressing modes and instruction set. It's a book no Mac owner can afford to miss! 208 pp., 31 illus. 7" x 10".
  Paper $16.95 Hard $24.95
  Book No. 2611

- **MASTERING THE 68000™ MICROPROCESSOR**—Phillip R. Robinson
  Delve into the heart of today's new 68000-based microcomputers—Apple®, Macintosh, ATARΙ®, and others. Here's a software-oriented approach to understanding and programming the Motorola family of microprocessors and support chips including the 68008, 68010, and the 68020. It takes you inside the architecture of the 68000 including registers, flags, and more! 272 pp., 121 illus.
  Paper $16.95 Hard $22.95
  Book No. 1886

- **1001 THINGS TO DO WITH YOUR MACINTOSH™**
  Here's a goldmine of exciting ideas ... more than 1000 ways you can use your Macintosh ... for applications ranging from household record-keeping to game playing and hobby use. Not only does this book include a treasure trove of ideas for getting more from your computer ... it provides you with actual programs, printouts, flowcharts, diagrams, and step-by-step instructions. 256 pp., 30 illus.
  Paper $10.95 Hard $15.95
  Book No. 1846

*Prices subject to change without notice.

Look for these and other TAB books at your local bookstore.

TAB BOOKS Inc.
P.O. Box 40
Blue Ridge Summit, PA 17214

Send for FREE TAB catalog describing over 1200 current titles in print.

OR ORDER TOLL-FREE TODAY: 1-800-233-1128
If you are intrigued with the possibilities of the programs included in *Turbo Pascal® for the Mac: Programming with Business Applications* (TAB Book No. 2927), you should definitely consider having the ready-to-run disk containing the source code in the book. This software is guaranteed free of manufacturer's defects. (If you have any problems, return the disk within 30 days, and we'll send you a new one.) Not only will you save the time and effort of typing the programs, the disk eliminates the possibility of errors that can prevent the programs from functioning. Interested? Available for any Macintosh with at least 128K at $24.95 for each disk plus $1.00 each for shipping and handling.

---

I'm interested. Send me:

_________ disk for *Turbo Pascal for the Mac: Programming with Business Applications* (6246S)

_________ TAB BOOKS catalog

Check/Money Order enclosed for $_________ ($24.95 plus $1.00 shipping and handling for each disk ordered.)

_________ VISA  _________ MasterCard

Account No.  __________________________  Expires  __________________________

Name  ________________________________________________________________

Address  ______________________________________________________________

City  __________________________  State  __________________________  Zip  ________

Signature  ______________________________________________________________

Mail To:  TAB BOOKS Inc.

        P.O. Box 40

        Blue Ridge Summit, PA 17294-0840

OR CALL TOLL-FREE TODAY:  1-800-233-1128

In Pennsylvania and Alaska call 717 794-2191

(Pa. add 6% sales tax. Orders outside U.S. must be prepaid with international money orders in U.S. dollars.)

Prices subject to change without notice.  TAB 2927
I'm interested. Send me:

_____________ disk for Turbo Pascal for the Mac: Programming with Business Applications (6246S)

_____________ TAB BOOKS catalog

Charge my credit card for ________________ ($24.95 plus $1.00 shipping and handling for each tape or disk ordered).

_____________ VISA ________________________ MasterCard ________________________

Account No. ____________________________ Expires ____________________________

Name ____________________________________________________________

Address ____________________________________________________________

City ____________________________ State ____________________________ Zip ______

Signature ____________________________________________________________

OR CALL TOLL FREE TODAY: 1-800-233-1128
IN PENNSYLVANIA AND ALASKA, CALL 717-794-2191

(Pa. add 6% sales tax. Orders outside U.S. must be prepaid with international money orders in U.S. dollars.)
Prices subject to change without notice.

TAB 2927
BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 9 BLUE RIDGE SUMMIT, PA 17214

POSTAGE WILL BE PAID BY ADDRESSEE

TAB BOOKS Inc.
P.O. Box 40
Blue Ridge Summit, PA 17214-9988

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES
Explore the amazing versatility and power of Turbo Pascal 3.0 and 3.1 on your Macintosh.

Speed on demand . . . powerful syntax . . . a full range of built-in functions . . . these are just a few of the advantages that have earned Borland International top acclaim for its Turbo Pascal compiler. For Mac users who have been anxious to try out Turbo Pascal's power, the time is now! Turbo® Pascal for the Mac: Programming with Business Applications makes the learning easy.

Mac-specific information is presented here in a manner that will be appreciated for its direct approach to specific applications. If it's profit and loss of a particular product you want to determine, turn to the chapter dedicated to that program. Other chapters describe in detail programs that will allow you to create charts and graphs from input raw data.

In fact, many of the programs included are ready to use with no modification whatsoever necessary. Each is individually described so that you can learn how the program is written, giving you the know-how to create your own Turbo Pascal programs for other applications. Think of it . . . a program to depreciate your capital assets . . . a program to evaluate your cash investments . . . programs that, at your fingertips, make it a simple matter to generate master record of sales, commissions paid out, expense reports, and more.

Anyone familiar with the Macintosh and the Turbo Pascal program's editor and compiler can quickly and easily become a proficient programmer following this guide. Turbo® Pascal for the Mac: Programming with Business Applications is a learn-by-doing guide with numerous useful examples and in-depth discussion of the program applications. The phenomenon of Borland's Turbo Pascal must be personally experienced. Its remarkable capabilities will make you look at your Mac in a whole new light!

Dr. Leon A. Wortman is president of Success Analysis Corporation, a consultant firm to high technology companies. He has written several computer books for TAB, including Turbo Pascal Programming, with Applications.

Send for FREE TAB Catalog describing over 1200 current titles in print.