CHAPTER 1

PROGRAMMING CONCEPTS

CHAPTER OBJECTIVES

In this Chapter, you will learn about:

Computers play a large role in the modern world. No doubt you realize how crucial they have become to running any business today; they have also become one of the sources of entertainment in our lives. You probably use computers for your everyday tasks as well, such as sending e-mail, paying bills, shopping, reading the latest news on the Internet, or even playing games.

A computer is a sophisticated device. However, it is important to remember that it is still only a device and cannot think on its own. In order to be useful, a computer needs instructions to follow. Facilities such as programming languages allow programmers to provide computers with a list of instructions called programs. These programs tell a computer what actions to perform. As a result, programming languages and computer programs play an important role in today's technology.

LAB 1.1

THE NATURE OF A COMPUTER PROGRAM AND PROGRAMMING LANGUAGES

LAB OBJECTIVES

After this Lab, you will be able to:

- ✔ Understand the Nature of Computer Programs and Programming Languages
- ✔ Understand the Differences between Interpreted and Compiled Languages

A computer needs instructions to follow because it cannot think on its own. For instance, when playing a game of solitaire you must choose which card to move. Each time a card is moved, a set of instructions has been executed to carry out the move. These instructions compose only a small part of the solitaire program. This program comprises many more instructions that allow a user to perform actions, such as beginning or ending a game, selecting a card's color, and so forth. Therefore, a computer program comprises instructions that direct the actions of the computer. In essence, a program plays the role of guide for a computer. It tells the computer what steps in what order should be taken to complete a certain task successfully.

Computer programs are created with the help of programming languages. A programming language is a set of instructions consisting of rules, syntax, numerical and logical operators, and utility functions. Programmers can use programming languages to create a computer program. There are many different programming

languages available today. However, all programming languages can be divided into three major groups: machine languages, assembly languages, and high-level languages.

Words such as statement *or* command *are often used when talking about instructions issued by a program to a computer. These terms are interchangeable.*

MACHINE LANGUAGES

Machine language is the native language of a particular computer because it is defined by the hardware of the computer. Each instruction or command is a collection of zeros and ones. As a result, machine language is the hardest language for a person to understand, but it is the only language understood by the computer. All other programming languages must be translated into machine language. Consider the following example of the commands issued in the machine language.

■ *FOR EXAMPLE*

Consider the mathematical notation $X = X + 1$. In programming, this notation reads *the value of the variable is incremented by one*. In the following example, you are incrementing the value of the variable by 1 using machine language specific to an Intel processor.

```
1010 0001 1110 0110 0000 0001
0000 0011 0000 0110 0000 0001 0000 0000
1010 0011 1110 0110 0000 0001
```

ASSEMBLY LANGUAGES

Assembly language uses English-like abbreviations to represent operations performed on the data. A computer cannot understand assembly language directly. A program written in assembly language must be translated into machine language with the help of the special program called an *assembler*. Consider the following example of the commands issued in assembly language.

■ *FOR EXAMPLE*

In this example, you are increasing the value of the variable by 1 as well. This example is also specific to an Intel processor.

```
MOV AX, [01E6]
ADD AX, 0001
MOV [01E6], AX
```

HIGH-LEVEL LANGUAGES

A high-level language uses English-like instructions and common mathematical notations. High-level languages allow programmers to perform complicated calculations with a single instruction. However, it is easier to read and understand than machine and assembly languages, and it is not as time-consuming to create a program in high-level language as it is in machine or assembly language.

■ *FOR EXAMPLE*

```
variable := variable + 1;
```

This example shows the simple mathematical operation of addition. This instruction can be easily understood by anyone without programming experience and with basic mathematical knowledge.

DIFFERENCES BETWEEN INTERPRETED AND COMPILED LANGUAGES

High-level languages can be divided into two groups: interpreted and compiled. Interpreted languages are translated into machine language with the help of another program called an *interpreter.* The interpreter translates each statement in the program into machine language and executes it immediately before the next statement is examined.

A compiled language is translated into machine language with the help of the program called a *compiler.* Compilers translate English-like statements into machine language. However, all of the statements must be translated before a program can be executed. The compiled version of the program is sometimes referred to as an *executable.*

An interpreted program must be translated into machine language every time it is run. A compiled program is translated into machine language only once when it is compiled. The compiled version of the program can then be executed as many times as needed.

LAB 1.1 EXERCISES

1.1.1 UNDERSTAND THE NATURE OF COMPUTER PROGRAMS AND PROGRAMMING LANGUAGES

a) What is a program?

For the next two questions, consider this scenario: You have been hired to work for the ABC Company. One of your responsibilities is to produce a daily report that contains complicated calculations.

b) Without using a computer program to fulfill this responsibility, what potential problems do you foresee in generating this report every day?

c) Based on your observations in question b, how do you think a computer program would make that task easier?

d) What is a programming language?

For the next question, consider the following code:

```
0010 0000 1110 0110 0000 0001
0000 0011 0000 0110 1000 0000
1010 0001 1111 0110 0000 0001
```

e) What type of programming language is this code written in?

For the next question, consider the following code:

```
MOV AX, [01E9]
ADD AX, 0010
MOV [01E6], AX
```

f) What type of programming language is this code written in?

For the next question, consider the following code:

```
variable := 2 * variable - 10;
```

g) What type of programming language is this code written in?

1.1.2 UNDERSTAND THE DIFFERENCES BETWEEN INTERPRETED AND COMPILED LANGUAGES

a) What is an interpreted language?

b) What is a compiled language?

c) Which do you think will run quicker, an interpreted or a compiled program?

LAB 1.1 EXERCISE ANSWERS

This section gives you some suggested answers to the questions in Lab 1.1, with discussion related to how those answers resulted. The most important thing to realize is whether your answer works. You should figure out the implications of the answers here and what the effects are from any different answers you may come up with.

1.1.1 ANSWERS

a) What is a program?

Answer: A computer program comprises instructions that direct the actions of the computer.

b) Without using a computer program to fulfill this responsibility, what potential problems do you foresee in generating this report every day?

Answer: Programs help us with repetitive, time-consuming, and error-prone tasks. If you do not have a program that helps you create this report, it might take you a whole day to collect the needed information for the report and perform the needed calculations. As a result, you will not be able to concentrate on your other responsibilities. In addition, sooner or later you will probably make mistakes while creating the report.

c) Based on your observations in question b, how do you think a computer program would make that task easier?

Answer: Using a program guarantees fast retrieval of needed information and accurate results, assuming that the program does not contain any errors. Furthermore, once a program is created, the same set of steps is repeated on a daily basis. Consequently, a

well-written program is not susceptible to human frailties such as typographical errors or the accidental exclusion of a formula.

d) What is a programming language?

Answer: A programming language is a set of instructions consisting of rules, syntax, numerical and logical operators, and utility functions.

e) What type of programming language is this code an example of?

Answer: This is an example of a machine language.

Machine language is understood directly by the computer. Each statement in machine language is represented by a string of zeros and ones.

This example illustrates the nonintuitive nature of machine language. However, a computer can read these instructions directly and execute them instantly. You can see that creating a program in a machine language can be a slow and tedious process. To facilitate program creation, programmers use higher-level languages that are closer to human language.

f) What type of programming language is this code an example of?

Answer: This is an example of an assembly language.

Assembly language uses mnemonic symbols to represent the binary code of machine language. Each assembly instruction is directly translated into a machine language instruction. You may notice that assembly language is slightly easier to understand than machine language.

g) What type of programming language is this code an example of?

Answer: This is an example of a high-level language.

Programs created in high-level languages are portable. They can be moved from one computer to another because a high-level programming language is not machine-specific. High-level languages must be translated into machine language with the help of an interpreter or a compiler.

1.1.2 ANSWERS

a) What is an interpreted language?

Answer: An interpreted language is translated into machine language with the help of another program called an interpreter. The interpreter translates statement in the program into machine language and executes it immediately before the next statement is examined.

b) What is a compiled language?

Answer: A compiled language is translated into machine language with the help of the program called a compiler. Compilers translate English-like statements into machine language.

c) Which do you think will run quicker, an interpreted or a compiled program?

Answer: Generally, interpreted programs run slower than compiled programs.

As you observed earlier, an interpreted program must be translated into machine language every time it is run. A compiled program is translated into machine language only once when it is compiled, and then it can be executed as many times as needed. As a result, an interpreted program runs slower than a compiled program.

LAB 1.1 SELF-REVIEW QUESTIONS

In order to test your progress, you should be able to answer the following questions.

1) What group of programming languages is easiest for the computer to understand?

- **a)** _____ The machine languages
- **b)** _____ The high-level languages
- **c)** _____ The assembly languages

2) Programs created in the machine languages are which of the following?

- **a)** _____ Portable
- **b)** _____ Machine-specific

3) Which of the following is true of interpreted programs?

- **a)** _____ All statements are translated and only then executed.
- **b)** _____ Each statement is translated and executed before the next statement.

4) Before a program written in a high-level language can be executed, which of the following must take place?

- **a)** _____ A program must be interpreted.
- **b)** _____ A program must be compiled.
- **c)** _____ A program can be executed immediately.

5) Which of the following is true of the interpreter?

- **a)** _____ It translates instructions written in assembly language into machine language.
- **b)** _____ It translates machine language into a high-level language.
- **c)** _____ It translates a high-level language into machine language.

Answers appear in Appendix A, Section 1.1.

LAB 1.2

GOOD PROGRAMMING PRACTICES

LAB OBJECTIVES

After this Lab, you will be able to:

- ✔ Understand the Nature of Good Programming Practices
- ✔ Understand Formatting Guidelines

In the previous section of this chapter you encountered the terms *computer program* and *programming language*. You will recall that a program is a set of instructions, and a programming language is a tool that allows programmers to provide computers with these instructions. However, the process of creating a computer program is not as simple as just writing down instructions. Sometimes it can become a tedious and complicated task. Before a computer can be provided with these instructions, a programmer needs to know what instructions must be specified. In essence, the process of creating a program is akin to the process of applied problem solving.

Consider this mathematical word problem:

> The 1980s speed record for human-powered vehicles was set on a measured 200-meter run by a sleek machine called Vector. Pedaling back-to-back, its two drivers averaged 69.92 miles per hour.
>
> This awkward mix of units is the way data appeared in an article reporting the event. Determine the speed of the vehicle in meters per second.[1]

This word problem involves conversion from miles per hour into meters per second. However, it contains information that has nothing to do with its solution, such as the name of the vehicle and the number of people needed to operate it. In order to achieve correct results, you must be able to filter out needed informa-

[1]From *Physics (with InfoTrac and Revised CD-ROM) Algebra/Trig,* 2nd edition, by E. Hecht. © 1998. Reprinted with permission of Brooks/Cole, a division of Thomas Learning. Fax 800-730-2215.

tion and discard the rest. Next, you need to know what formulas must be used for actual conversion.

This is a relatively straightforward example of a problem-solving process that can be used for academic purposes. However, in the business world, problem descriptions are often incomplete or ambiguous. They are also harder to solve. These problems require the ability to ask questions that help clarify the problem and an ability to organize the problem into logical parts. By breaking down the problem, you will be able to focus better on possible solutions and more easily manage each part. Once each part is fully understood, the solution to the overall problem will readily develop.

This technique of breaking the problem into smaller parts and solving each part is called a *top-down approach* to problem solving. When writing a program, you can also approach your task in a top-down manner. However, to solve the problem efficiently, you need to approach it in a structured manner.

STRUCTURED PROGRAMMING

Structured programming embodies a disciplined approach to writing clear code that is easy to understand, test, maintain, and modify. A program can be organized into modules called *subroutines*. These subroutines focus on a particular part of the overall problem that the program addresses. Subroutines are easier to understand and manage because they are only components of the overall program. Together, all of the subroutines compose the overall program.

Structured programming also embodies the following three attributes: sequence, selection, and iteration. These attributes, or structures, describe how statements in the program are executed. Furthermore, a program can contain any of these structures or a combination of them.

SEQUENCE

Sequence refers to the linear execution of code. In other words, control is passed from one statement to the next statement in consecutive order. Consider Figure 1.1.

Figure 1.1 contains rectangular symbols. The rectangular symbol in the diagram can represent not only a single statement, but a subroutine as well. The arrows represent the flow of control between statements. Control is passed from statement 1 to statement 2 and then to statement 3. Thus, these statements are executed in the sequential order.

SELECTION

Selection refers to the decision-making process. For example, when I am trying to choose between different activities for this weekend, I start with the knowledge

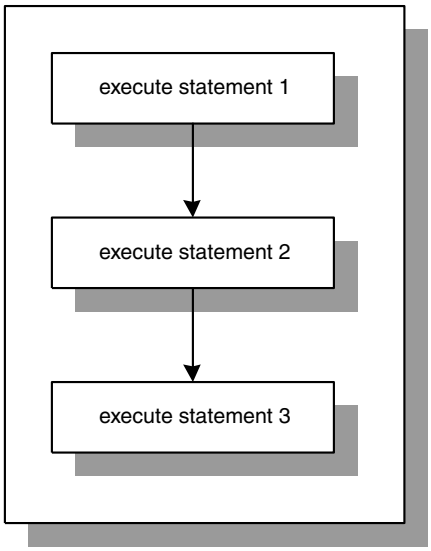

Figure 1.1 ■ Sequence Structure

that on Friday night I want to go to the movies, Saturday night I want to go dancing, and Sunday I want to spend a quiet evening at home. In order for me to choose one of the activities, I need to know what day of the week it is. The logic for my decision of the weekend activities can be illustrated as follows:

```
IF TODAY IS 'FRIDAY'
   I AM GOING TO SEE A MOVIE
IF TODAY IS 'SATURDAY'
   I AM GOING DANCING
IF TODAY IS 'SUNDAY'
   I AM SPENDING A QUIET EVENING AT HOME
```

The test conditions "TODAY IS . . ." can evaluate either to TRUE or FALSE based on the day of the week. If today happens to be Friday, the first test condition "TODAY is 'FRIDAY'" becomes TRUE, and the other test conditions become FALSE. In this case, I am going to see a movie, and the other activities can be discarded.

Figure 1.2 illustrates the general flow of control of the selection structure.

Figure 1.2 contains a diamond shape called the decision symbol. This indicates that a decision must be made or a certain test condition must be evaluated. This test condition evaluates to TRUE (Yes) or FALSE (No). If the test condition yields TRUE, statement 1 is executed. If the test condition yields FALSE, statement 2 is executed. It is important for you to remember that a rectangle can represent a set of statements or a subroutine.

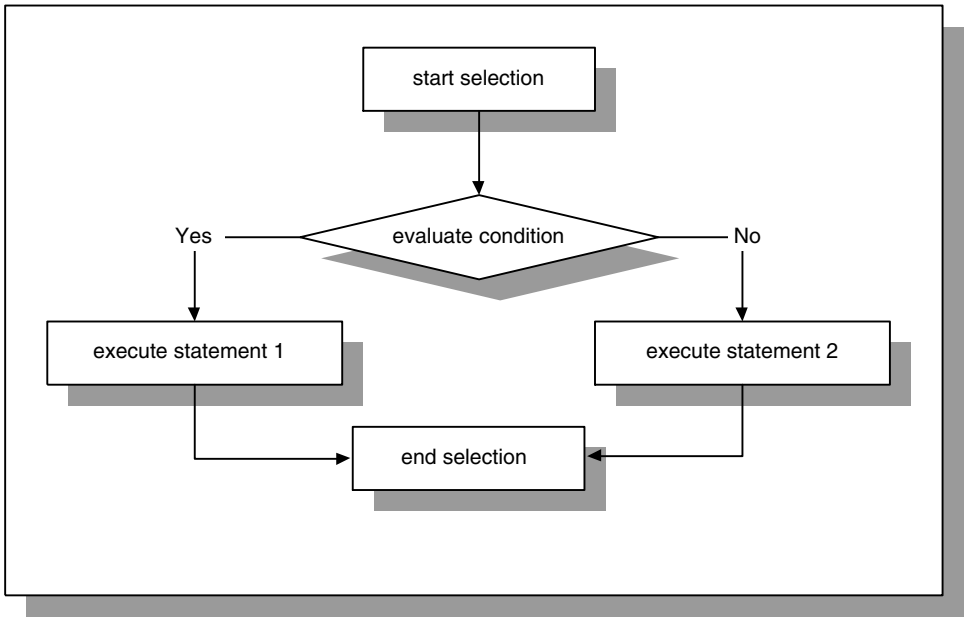

Figure 1.2 ■ Selection Structure

ITERATION

Iteration refers to an action that needs to be repeated a finite number of times. The number of times this action is repeated is based on some terminating factor. Consider the following example. You are reading a chapter from this book. Each chapter has a finite number of pages. In order to finish the chapter, you need to read through all of the pages. This is indicated as follows:

```
WHILE THERE ARE MORE PAGES IN THE CHAPTER TO READ
   READ THE CURRENT PAGE
   GO TO THE NEXT PAGE
```

The terminating factor in this example is the number of pages in the chapter. As soon as the last page in the chapter is read, the iteration is complete.

Figure 1.3 illustrates the general flow of control of the iteration structure.

As long as the condition evaluates to TRUE, the statements inside the iteration structure are repeated. As soon as the condition evaluates to FALSE, the flow of control is passed to the exit point of the iteration structure.

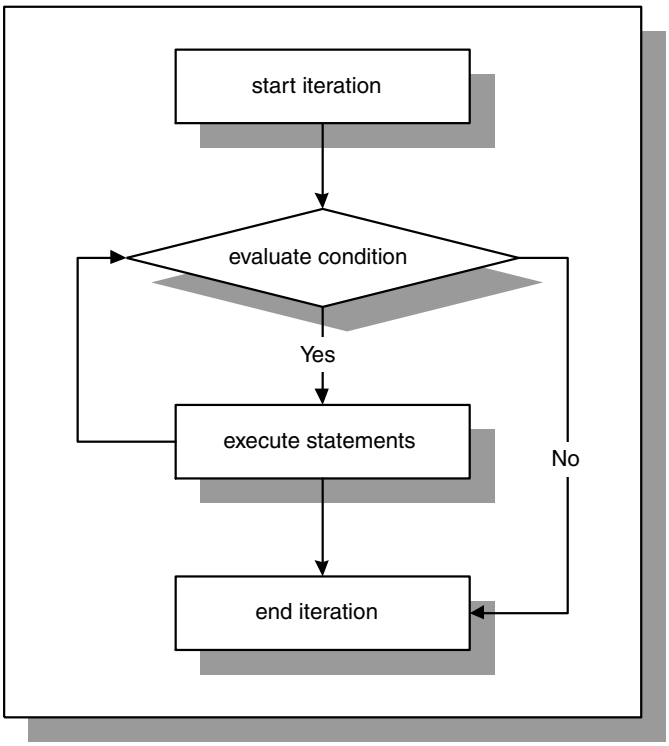

Figure 1.3 ■ Iteration Structure

DIFFERENCES BETWEEN STRUCTURED AND NONSTRUCTURED PROGRAMMING

Before structured programming became widely used, programs were simply sequential lines of code. This code was not organized into modules and did not employ many of the structures you encountered earlier in this chapter. The result was a meandering set of statements that was difficult to maintain and understand. In addition, these programs used multiple GOTO statements that allow program control to jump all over the code. Almost all programs that use GOTO statements can be rewritten using structures such as selection and iteration.

FORMATTING GUIDELINES

It was mentioned earlier that structured programming allows us to write clear code that is easy to understand, test, maintain, and modify. However, structured programming alone is not enough to create readable and manageable code. Formatting is a very important aspect of writing a program. Moreover, your formatting style should stay consistent throughout your programs.

Consider this example of a SELECT statement that has not been formatted.

FOR EXAMPLE

```
SELECT s.first_name, s.last_name, e.final_grade FROM
student s, enrollment e WHERE s.student_id = e.student_id
AND e.final_grade IS NOT NULL;
```

Even though this example contains only a very simple SELECT statement, you can see that the logic is hard to follow.

Consider the same SELECT statement with a few formatting changes.

FOR EXAMPLE

```
SELECT s.first_name, s.last_name, e.final_grade
  FROM student s, enrollment e
 WHERE s.student_id = e.student_id
   AND e.final_grade IS NOT NULL;
```

You have probably noticed that the second version of the SELECT statement is much easier to read and understand. It is important to realize that both SELECT statements are syntactically correct. They produce the same output when run.

Usually, the logic depicted in the program is more complex than that of the SELECT statement. Therefore, proper formatting of the code is extremely important for two major reasons. First, a well-formatted program will facilitate any changes made later by the program's author. In other words, even the author will understand the logic of the program more easily if he or she needs to modify the program later. Second, any person who has to maintain the program can more easily follow the logical structure of the program.

In order for the program to be readable and understandable, there are two main guidelines to follow. First, the format of the program must illustrate the logical structure of the program. You can reveal the logical structure of the program by using indentation in your code. Consider the example of the selection structure used earlier in this chapter.

FOR EXAMPLE

```
IF TODAY IS 'FRIDAY'
   I AM GOING TO SEE A MOVIE
IF TODAY IS 'SATURDAY'
   I AM GOING DANCING
IF TODAY IS 'SUNDAY'
   I AM SPENDING A QUIET EVENING AT HOME
```

You have probably noticed that each statement following the IF clause is indented. As a result, it is easier to understand what activity is taken based on the day of the week. You could take this example and format it differently.

■ *FOR EXAMPLE*

```
IF TODAY IS 'FRIDAY' I AM GOING TO SEE A MOVIE
IF TODAY IS 'SATURDAY' I AM GOING DANCING
IF TODAY IS 'SUNDAY' I AM SPENDING A QUIET EVENING AT HOME
```


This example also shows a formatted version of the selection structure. However, this formatting style does not reveal the logical structure of the selection as well as the previous example. As a matter of fact, this example looks like an extremely short story rather than a program.

Second, *your program should contain comments*. Comments will help you explain what you are trying to accomplish. However, you should be careful because too many comments can make your code confusing.

You can use the code format used in this book's examples as you write your programs. It is not the only good format available, but it will be a good example of formatting technique, which will help you to develop your own style. However, regardless of your style, you should follow these guidelines when creating a program.

LAB 1.2 EXERCISES

1.2.1 UNDERSTAND THE NATURE OF GOOD PROGRAMMING PRACTICES

a) What is a top-down approach?

b) What is structured programming?

c) Create the following selection structure: Determine which season each month of the year belongs to.

d) Create the following iteration structure: For every day of the week display its name.

e) Create the following structure: For every day that falls within the business week, display its name. For every day that falls on the weekend, display "The weekend is here, and it is here to

stay!!!" *Hint:* You will need to use iteration and selection structures. The selection structure must be placed inside the iteration structure.

1.2.2 UNDERSTAND FORMATTING GUIDELINES

a) What is the reason for formatting your code?

b) What are two main guidelines of good formatting?

LAB 1.2 EXERCISE ANSWERS

This section gives you some suggested answers to the questions in Lab 1.2, with discussion related to how those answers resulted. The most important thing to realize is whether your answer works. You should figure out the implications of the answers here and what the effects are from any different answers you may come up with.

1.2.1 ANSWERS

a) What is a top-down approach?

Answer: The technique of breaking a problem into parts and solving each part is called a top-down approach to problem solving. By breaking down the problem, it is easier to focus on possible solutions and manage each part. Once each part is fully understood, the solution to the overall problem can be readily developed.

b) What is structured programming?

Answer: Structured programming embodies a disciplined approach to writing clear code that is easy to understand, test, maintain, and modify. A program can be organized into modules called subroutines. These subroutines focus on a particular part of the overall problem that the program addresses. Subroutines are easier to understand and manage because they are only components of the overall program. Together, all of the subroutines compose the overall program.

c) Create the following selection structure: Determine which season each month of the year belongs to.

Answer: Your selection structure should look similar to the following:

```
IF MONTH IN ('DECEMBER', 'JANUARY', 'FEBRUARY')
   IT IS WINTER
IF MONTH IN ('MARCH', 'APRIL', 'MAY')
   IT IS SPRING
IF MONTH IN ('JUNE', 'JULY', 'AUGUST')
   IT IS SUMMER
IF MONTH IN ('SEPTEMBER', 'OCTOBER', 'NOVEMBER')
   IT IS FALL
```

The test conditions of this selection structure use the operator IN. This operator allows you to construct the list of valid months for every season. It is important to understand the use of the parentheses. In this case, it is not done for the sake of a syntax rule. This use of parentheses allows us to define clearly the list of values for a specific month, hence helping us to outline the logic of the structure.

Now, consider the following fragment of the selection structure:

```
IF MONTH IS 'DECEMBER'
   IT IS WINTER
IF MONTH IS 'JANUARY'
   IT IS WINTER
IF MONTH IS 'FEBRUARY'
   IT IS WINTER
…
```

This selection structure results in the same outcome, yet it is much longer. As a result it does not look well structured, even though it has been formatted properly.

d) Create the following iteration structure: For every day of the week, display its name.

Answer: Your selection structure should look similar to the following:

```
WHILE THERE ARE MORE DAYS IN THE WEEK
   DISPLAY THE NAME OF THE CURRENT DAY
   GO TO THE NEXT DAY
```

Assume that you are starting your week on Monday—there are six days left. Next, you will display the name of the current day of the week, which is Monday for the first iteration. Then, you move to the next day. The next day is Tuesday, and there are five more days in the week. So, you will display the name of the current day—Tuesday—and move to the next day, and so forth. Once the name of the seventh day (Sunday) has been displayed, the iteration structure has completed.

e) Create the following structure: For every day that falls within the business week, display its name. For every day that falls on the weekend, display "The weekend is here, and it is here to stay!!!"

Answer:Your structure should look similar to the following:

```
WHILE THERE ARE MORE DAYS IN THE WEEK
   IF DAY BETWEEN 'MONDAY' AND 'FRIDAY'
      DISPLAY THE NAME OF THE CURRENT DAY
   IF DAY IN ('SATURDAY', 'SUNDAY')
      DISPLAY 'THE WEEKEND IS HERE, AND IT IS HERE TO STAY!!!'
   GO TO THE NEXT DAY
```


This structure is a combination of two structures: iteration and selection. The iteration structure will repeat its steps for each day of the week. The selection structure will display the name of the current day or the message "The weekend is. . . ."

Assume that you are starting your week on Monday again. There are six days left. Next, control of the flow is passed to the selection structure. Because the current day happens to be Monday, and it falls within the business week, its name is displayed. Then, control of the flow is passed back to the iteration structure, and you are ready to move to the next day.

The next day is Tuesday, and there are five more days in the week. So, control is passed to the iteration structure again. Tuesday also falls within the business week, so its name is displayed as well. Next, control is passed back to the iteration structure, and you go to the next day, and so forth. Once the day falls on the weekend, the message "The weekend is . . ." is displayed.

1.2.2 ANSWERS

a) What is the reason for formatting your code?

Answer:A well-formatted program is easier to understand and maintain because format can reveal the logical structure of the program.

b) What are two main guidelines of good formatting?

Answer: First, the code of the program should be indented so that the logical structure of the program is clear. Second, the program should contain comments describing what is being accomplished.

LAB 1.2 SELF-REVIEW QUESTIONS

In order to test your progress, you should be able to answer the following questions.

1) Which one is not a feature of the structured programming?

a) _____ Iteration
b) _____ Sequence
c) _____ GOTO
d) _____ Modularity

2) Structured programming allows control of the program to jump all over the code.

a) _____ True
b) _____ False

3) Which of the following is true about sequence structure?

a) _____ It refers to the decision-making process.
b) _____ It refers to the linear execution of code.
c) _____ It refers to the repetition of code.

4) A test condition must evaluate to which of the following in order for the selection to execute?

a) _____ TRUE
b) _____ FALSE
c) _____ None of the above

5) A poorly formatted SELECT statement produces output different from a well formatted SELECT statement.

a) _____ True
b) _____ False
c) _____ None of the above

Answers appear in Appendix A, Section 1.2.

CHAPTER 1

TEST YOUR THINKING

In this chapter you learned what a program is. You also defined the concepts of the structured programming. Here are some projects that will help you test the depth of your understanding.

1) Create the following structure: Based on the value of a number, determine if it is even or odd. *Hint:* Before you decide how to define even and odd numbers, you should decide what structure must be used to achieve the desired results.
2) Create the following structure: The structure you created in the previous exercise is designed to work with a single number. Modify it so that it can work with a list of numbers.

The projects in this section are meant to have you utilize all of the skills that you have acquired throughout this chapter. The answers to these projects can be found in Appendix D and at the companion Web site to this book, located at http://www.phptr.com/rosenzweig2e. Visit the Web site periodically to share and discuss your answers.