In this chapter, you begin to learn how to write structured Visual Basic programs. As you will see, creating a flowchart and writing pseudocode before you actually write the program ensures that you fully understand the program's intended design. You begin by looking at a structured flowchart and pseudocode from your text, Programming Logic and Design, Eighth Edition. You should do the exercises and labs in this chapter only after you have finished Chapters 2 and 3 of that book.

Using Flowcharts and Pseudocode to Write a Visual Basic Program

In the first three chapters of Programming Logic and Design, you studied flowcharts and pseudocode for the Number-Doubling program. Figure 3-1 shows the functional, structured flowchart and pseudocode for this program.

By studying the flowchart and pseudocode, you can see that this program makes use of the sequence and loop structures introduced to you in Programming Logic and Design. The
Using Flowcharts and Pseudocode to Write a Visual Basic Program

remainder of this section walks you through the Visual Basic code for this program. The explanations assume that you are simply reading along, but if you want, you can type the code as it is presented. The goal of this section is to help you get a feel for how flowcharts and pseudocode can serve as a guide as you write Visual Basic programs. You must learn more about Visual Basic before you can expect to write this program by yourself.

In Figure 3-1, the first line of the pseudocode is the word start. How do you translate this pseudocode command into the Visual Basic code that will start the Number-Doubling program? In Chapter 1 of this book, you learned that, to start a Visual Basic program, you first create a module. So, to start the Number-Doubling program, you will first create a module named NumberDouble. You should also include one procedure in the module named Main(), which is always the first procedure that executes in a Visual Basic program. Thus, the code that follows starts the Number-Doubling program by including the statements that turn on the Explicit and Strict options, creating a module named NumberDouble and including the Main() procedure:

```vbnet
Option Explicit On
Option Strict On
Module NumberDouble
    Sub Main()
        End Sub
    End Module
End Module
```

Next, you see that two variables, originalNumber and calculatedAnswer, are declared as data type num. The Visual Basic code that follows adds the variable declarations, with the declarations shown in bold.

```vbnet
Option Explicit On
Option Strict On
Module NumberDouble
    Sub Main()
        Dim OriginalNumber As Integer
        Dim CalculatedAnswer As Integer
        End Sub
    End Module
```

If you are typing the code as it is presented here, save the program in a file that has an appropriate name, such as NumberDouble.vb. The complete program is also saved in a file named NumberDouble.vb and is included in the student files for this chapter.

The next line of the pseudocode instructs you to input the originalNumber. In other words, you need to write the input statement that primes the loop. You learned about priming read statements in Chapter 3 of Programming Logic and Design. In Chapter 2 of this book, you learned how to use interactive input statements in programs to allow the user to input data. You also learned to prompt the user by explaining what the program expects to receive as input. The following example includes the code that implements the priming read by displaying a dialog box where users can input the number they want doubled. The next statement converts the input String to an Integer.
The code in boldface has been added to the NumberDouble module in the Main() procedure. The String variable named OriginalNumberString is added to hold the input entered into the input dialog box. If you were writing this code yourself, you would start by writing the code for the NumberDouble module, and then edit it to add the boldface code shown here:

```vbnet
Option Explicit On
Option Strict On
Module NumberDouble
  Sub Main()
    Dim OriginalNumber As Integer
    Dim OriginalNumberString As String
    Dim CalculatedAnswer As Integer
    OriginalNumberString = InputBox$("Enter number to double : ")
    OriginalNumber = Convert.ToInt32(OriginalNumberString)
    While OriginalNumber <> 0
      CalculatedAnswer = OriginalNumber * 2
      Debug.Print CalculatedAnswer
      OriginalNumberString = InputBox$("Enter number to double : " & _
                                "or 0 to end: ")
      OriginalNumber = Convert.ToInt32(OriginalNumberString)
    End While
  End Sub
End Module
```

You have not learned enough about while loops to write this code yourself, but you can observe how it is done in this example. You will learn more about loops in Chapter 5 of this book.

According to the pseudocode, the body of the loop is made up of three sequential statements. The first statement calculates the originalNumber multiplied by 2; the second statement prints the calculatedAnswer; and the third statement retrieves the next originalNumber.
from the user. In Visual Basic, you actually need to add an additional, fourth statement in the body of the while loop. This fourth statement converts the input String to an Integer.

In the following example, the code that makes up the body of the loop is in bold.

```vbnet
Option Explicit On
Option Strict On
Module NumberDouble
    Sub Main()
        Dim OriginalNumber As Integer
        Dim OriginalNumberString As String
        Dim CalculatedAnswer As Integer
        OriginalNumberString = InputBox$( _
            "Enter number to double " & _
            "or 0 to end: ")
        OriginalNumber = Convert.ToInt32(OriginalNumberString)
        While OriginalNumber <> 0
            CalculatedAnswer = OriginalNumber * 2
            System.Console.WriteLine(OriginalNumber & _
                " doubled is " & CalculatedAnswer)
            OriginalNumberString = InputBox$( _
                "Enter number to double " & _
                "or 0 to end: ")
            OriginalNumber = Convert.ToInt32(OriginalNumberString)
        End While
        End Sub
    End Module
```

The last line of the pseudocode instructs you to end the program. In Visual Basic, the End Sub statement signifies the end of the Main() procedure, and the End Module statement ends the NumberDouble module.

At this point, the program is ready to be compiled. Assuming there are no syntax errors, it should execute as planned. Figure 3-2 displays the input and output of the Number-Doubling program.

Notice that the underscore character (_) is used in the output statement to continue a single statement from one line to the next.

Notice that an End While statement marks the end of code that executes as part of a loop.
Although you have not learned everything you need to know to write this program yourself, you can see from this example that writing the program in Visual Basic is easier if you start with a well-designed, functional, structured flowchart or pseudocode.

Lab 3-1: Using Flowcharts and Pseudocode to Write a Visual Basic Program

In this lab, you use the pseudocode in Figure 3-3 to add code to a partially created Visual Basic program. When completed, college admissions officers should be able to use the Visual Basic program to determine whether to accept or reject a student, based on his or her test score and class rank.