Writing Programs that Make Decisions

After studying this chapter, you will be able to:

- Use comparison and logical Boolean operators to make decisions in a program
- Compare numeric and String variables
- Write decision statements in Visual Basic, including an If statement, an If Then Else statement, nested If statements, and the Select Case statement
- Use decision statements to make multiple comparisons by using AND logic and OR logic
You should complete the exercises and labs in this chapter only after you have finished Chapter 4 of your book, Programming Logic and Design, Eighth Edition. In this chapter, you practice using Visual Basic's comparison and logical operators to write expressions. You also learn the Visual Basic syntax for decision statements, including the If statement, the If Then Else statement, nested If statements, and Select Case statements. Finally, you learn to write Visual Basic statements that make multiple comparisons.

**Boolean Operators**

You use Boolean operators in expressions when you want to compare values. When you use a Boolean operator in an expression, the expression evaluates to either True or False. In Visual Basic, you can subdivide the Boolean operators into two groups: comparison operators and logical operators. We begin the discussion with the comparison operators.

**Comparison Operators**

When you write Visual Basic programs, you will often want to compare the values stored in variables. For example, you may want to know if one value is greater than another, less than another, or equal to another value. The terms greater than, less than, and equal to each refer to relationships between two values. As with all Boolean operators, a comparison operator allows you to ask a question that results in a True or False answer. The logical path your program takes will depend on the answer to that question. Table 4-1 lists the comparison operators used in Visual Basic.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal to</td>
</tr>
</tbody>
</table>

Table 4-1 Comparison operators

To see how to use comparison operators, suppose you declare two variables: an Integer named Number1 that you initialize with the value 10 and another Integer variable named Number2 that you initialize with the value 15. The following code shows the declaration statements for these variables:

```vbnet
Dim Number1 As Integer = 10
Dim Number2 As Integer = 15
```
Boolean Operators

The following code samples illustrate how comparison operators are used in expressions:

- $\text{Number1} < \text{Number2}$ evaluates to True because 10 is less than 15.
- $\text{Number1} \leq \text{Number2}$ evaluates to True because 10 is less than or equal to 15.
- $\text{Number1} > \text{Number2}$ evaluates to False because 10 is not greater than 15.
- $\text{Number1} \geq \text{Number2}$ evaluates to False because 10 is not greater than or equal to 15.
- $\text{Number1} = \text{Number2}$ evaluates to False because 10 is not equal to 15.
- $\text{Number1} \neq \text{Number2}$ evaluates to True because 10 is not equal to 15.

Logical Operators

You can use another type of Boolean operator, logical operators, when you need to ask more than one question but you want to receive only one answer. For example, in a program, you may want to ask if a number is between the values 1 and 10. This actually involves two questions. You need to ask if the number is greater than 1 and if the number is less than 10. Here, you are asking two questions, but you want only one answer—either yes (True) or no (False).

Logical operators are useful in decision statements because, like comparison expressions, they evaluate to True or False, thereby permitting decision making in your programs.

Table 4-2 lists the logical operators used in Visual Basic.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>And</td>
<td>AND</td>
<td>All expressions must evaluate to True for the entire expression to be True.</td>
</tr>
<tr>
<td>Or</td>
<td>OR</td>
<td>Only one expression must evaluate to True for the entire expression to be True.</td>
</tr>
<tr>
<td>Not</td>
<td>NOT</td>
<td>This operator reverses the value of the expression; if the expression evaluates to False, then reverse it so that the expression evaluates to True.</td>
</tr>
</tbody>
</table>

Table 4-2 Logical operators

To see how to use the logical operators, suppose you declare two variables: an Integer named Number1 that you initialize with the value 10; and another Integer variable named Number2 that you initialize with the value 15 as in the previous example.

The following code samples illustrate how you can use the logical operators along with the comparison operators in expressions:

- $\text{Number1} > \text{Number2}$ Or $\text{Number1} = 10$ evaluates to True because the first expression evaluates to False, 10 is not greater than 15, and the second expression evaluates to True, 10 is equal to 10. Only one expression needs to be True using OR logic for the entire expression to be True.
Writing Programs that Make Decisions

- Number1 > Number2 And Number1 = 10 evaluates to False because the first expression is False, 10 is not greater than 15, and the second expression is True, 10 is equal to 10. Using AND logic, both expressions must be True for the entire expression to be True. This expression would actually result in a short-circuit evaluation. This means that when the first expression, Number1 > Number2, evaluates to False, the second expression, Number1 = 10, does not need to be evaluated because AND logic dictates that both expressions must be True for the entire expression to be True.

- Number1 <> Number2 And Number1 = 10 evaluates to True because both expressions are True; that is, 10 is not equal to 15, and 10 is equal to 10. Using AND logic, if both expressions are True, then the entire expression is True.

- Not Number1 = Number2 evaluates to True because the expression Number1 = Number2 evaluates to False, 10 is not equal to 15. The Not operator then reverses False, which results in a True value.

Comparison and Logical Operator Precedence and Associativity

Like the arithmetic operators discussed in Chapter 2, the comparison and logical operators are evaluated according to specific rules of associativity and precedence. Table 4-3 shows the precedence and associativity of the operators discussed thus far in this book.

<table>
<thead>
<tr>
<th>Operator Name</th>
<th>Symbol</th>
<th>Order of Precedence</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parentheses</td>
<td>()</td>
<td>First</td>
<td>Left to right</td>
</tr>
<tr>
<td>Exponentiation</td>
<td>^</td>
<td>Second</td>
<td>Left to right</td>
</tr>
<tr>
<td>Unary</td>
<td>- +</td>
<td>Third</td>
<td>Right to left</td>
</tr>
<tr>
<td>Multiplication, floating point division</td>
<td>* / \</td>
<td>Fourth, Fifth</td>
<td>Left to right</td>
</tr>
<tr>
<td>Integer division</td>
<td>Mod</td>
<td>Sixth</td>
<td>Left to right</td>
</tr>
<tr>
<td>Modulus</td>
<td></td>
<td>Seventh</td>
<td>Left to right</td>
</tr>
<tr>
<td>Addition and subtraction</td>
<td>+ -</td>
<td>Eighth</td>
<td>Left to right</td>
</tr>
<tr>
<td>Comparison</td>
<td>&lt; &gt; &lt;= &gt;= &lt;&gt;</td>
<td>Ninth</td>
<td>Left to right</td>
</tr>
<tr>
<td>Negation</td>
<td>Not</td>
<td>Tenth</td>
<td>Left to right</td>
</tr>
<tr>
<td>AND</td>
<td>And</td>
<td>Eleventh</td>
<td>Left to right</td>
</tr>
<tr>
<td>OR</td>
<td>Or</td>
<td>Twelfth</td>
<td>Right to left</td>
</tr>
<tr>
<td>Assignment</td>
<td>= += -= *= /= \ = ^=</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4-3 Order of precedence and associativity

As shown in Table 4-3, the AND operator has a higher precedence than the OR operator, meaning expressions that include the AND operator are evaluated first. Also notice that the
Boolean Operators

comparison operators have higher precedence than the AND and OR operators. All of these operators have left-to-right associativity.

To see how to use the logical operators and the comparison operators in expressions, first assume that the variables Number1 and Number2 are declared and initialized as shown in the following code:

```vbnet
Dim Number1 As Integer = 10
Dim Number2 As Integer = 15
```

Next, you write the following expression in Visual Basic:

```vbnet
Number1 = 8 And Number2 = Number1 Or Number2 = 15
```

Looking at Table 4-3, you can see that the comparison operator (=) has a higher level of precedence than the AND operator (And), and the AND operator (And) has a higher level of precedence than the OR operator (Or). Also, notice that there are three (=) operators in the expression; thus, the left-to-right associativity rule applies. Figure 4-1 illustrates the order in which the operators are used.

As you can see in Figure 4-1, it takes five steps, following the rules of precedence and associativity, to determine the value of the expression.

![Figure 4-1 Evaluation of expression using relational and logical operators](image)

You can change the order of precedence by using parentheses.
As you can see in Figure 4-2, when parentheses are added, it still takes five steps, but the order of evaluation is changed, and the result is also changed.

```
Dim Number1 As Integer = 10
Dim Number2 As Integer = 15

Number1 = 8 And (Number2 = Number1 Or Number2 = 15)
```

**Figure 4-2**  Evaluation of expression using relational and logical operators with parentheses added

---

**Comparing Strings**

In Visual Basic, you use the same comparison operators when you compare String variables that you use to compare numeric data types such as Integers and Doubles.

The following code shows how to use the equality operator to compare two String variables and also to compare one String variable and one string constant:

```
Dim S1 As String = "Hello"
Dim S2 As String = "World"
' The following test evaluates to False because "Hello"
' is not the same as "World".
If S1 = S2 Then
   ' code written here executes if True
Else
   ' code written here executes if False
End If
' The following test evaluates to True because "Hello"
' is the same as "Hello".
If S1 = "Hello" Then
   ' code written here executes if True
Else
   ' code written here executes if False
```
Comparing Strings

End If
' The following test evaluates to False because "Hello"
' is not the same as "hello".
If S1 = "hello" Then
' code written here executes if True
Else
' code written here executes if False
End If

Visual Basic does not consider a lowercase h to be equal to an uppercase H because their ASCII values are different. Lowercase h has an ASCII value of 104, and uppercase H has an ASCII value of 72. A table of ASCII values can be found in Appendix A: Understanding Numbering Systems and Computer Codes in Programming Logic and Design.

The following code shows how to use the other comparison operators to compare two String variables and also to compare one String variable and one string constant:

Dim S1 As String = "Hello"
Dim S2 As String = "World"
' The following test evaluates to False because "Hello"
' is not greater than "World".
If S1 > S2 Then
' code written here executes if True
Else
' code written here executes if False
End If

' The following test evaluates to True because "Hello"
' is the same as "Hello".
If S1 <= "Hello" Then
' code written here executes if True
Else
' code written here executes if False
End If

Two String variables are equal when their contents are the same.

When you compare Strings, Visual Basic compares the ASCII values of the individual characters in the String to determine if one String is greater than, less than, or equal to another, in terms of alphabetizing the text in the Strings. As shown in the preceding code, the String variable S1, whose value is "Hello", is not greater than the String variable S2, whose value is "World", because World comes after Hello in alphabetical order.

The following code sample shows additional examples of using the comparison operators and the equality operator with two String variables:

Dim S1 As String = "whole"
Dim S2 As String = "whale"
' The next statement evaluates to True because the
' contents of S1, "whole", are greater than the
' contents of S2, "whale".

When you compare Strings, Visual Basic compares the ASCII values of the individual characters in the String to determine if one String is greater than, less than, or equal to another, in terms of alphabetizing the text in the Strings. As shown in the preceding code, the String variable S1, whose value is "Hello", is not greater than the String variable S2, whose value is "World", because World comes after Hello in alphabetical order.

The following code sample shows additional examples of using the comparison operators and the equality operator with two String variables:

Dim S1 As String = "whole"
Dim S2 As String = "whale"
' The next statement evaluates to True because the
' contents of S1, "whole", are greater than the
' contents of S2, "whale".
Decision Statements

Every decision in a program is based on whether an expression evaluates to True or False. Programmers use decision statements to change the flow of control in a program. **Flow of control** means the order in which statements are executed. Decision statements are also known as branching statements because they cause the computer to make a decision, choosing from one or more branches (or paths) in the program.

There are different types of decision statements in Visual Basic. You begin with the If statement.

The if Statement

The If statement is a single-path decision statement. As you learned in *Programming Logic and Design*, If statements are also referred to as "single alternative" or "single-sided" statements.

When you use the term single-path, it means that if an expression evaluates to True, your program executes one or more statements, but if the expression evaluates to False, your program will not execute those statements. There is only one defined path—the path taken if the expression evaluates to True. In either case, the statement following the If statement is executed.

The syntax, or set of rules, for writing an If statement in Visual Basic is as follows:

```
If expression Then
    statementA
End If
```
Decision Statements

Note that when you type the keyword If to begin an If statement, you follow it with an expression followed by the keyword Then.

When the compiler encounters an If statement, the expression is evaluated. If the expression evaluates to True, then the computer executes statementA. If the expression evaluates to False, then the computer will not execute statementA. Remember that whether the expression evaluates to True and executes statementA, or the expression evaluates to False and does not execute statementA, the statement following the End If executes next.

Note that a Visual Basic statement, such as an If statement, can be either a simple statement or a block statement. A block statement is made up of multiple Visual Basic statements. Visual Basic defines a block as multiple statements placed between the If and the End If. The following example illustrates an If statement that uses the comparison operator (<) to test if the value of the variable CustomerAge is less than 65. You will see If in the fourth line and the End If in the second-to-last line:

```
Dim CustomerAge As Integer = 53
Dim Discount As Integer
Dim NumUnder_65 As Integer = 0
If CustomerAge < 65 Then
    Discount = 0
    NumUnder_65 += 1
End If
System.Console.WriteLine("Discount : " & Discount)
```

In the preceding code, the variable named CustomerAge is initialized to the value 53. Because 53 is less than 65, the expression CustomerAge < 65 evaluates to True, and the block statement executes. The block statement is made up of the two assignment statements within the If and End If statements: Discount = 0 and NumUnder_65 += 1. If the expression evaluates to False, the block statement does not execute. In either case, the next statement to execute is the output statement, System.Console.WriteLine("Discount : " & Discount).

The following code uses an If statement to test a String variable and a string constant for equality:

```
Dim DentPlan As String = "Y"
Dim GrossPay As Double
If DentPlan = "Y" Then
    GrossPay = GrossPay - 23.50
End If
```

In this example, if the value of the String variable named DentPlan and the string constant "Y" are the same value, the expression evaluates to True, and the GrossPay calculation assignment statement executes. If the expression evaluates to False, the GrossPay calculation assignment statement does not execute.

**Exercise 4-1: Understanding If Statements**

In this exercise, you use what you have learned about writing If statements. Study the following code and then answer Questions 1–4.
Lab 4.1: Understanding If Statements

In this lab, you complete a prewritten Visual Basic program for a carpenter who creates personalized house signs. The program is supposed to compute the price of any sign a customer orders, based on the following facts:
The starting price for all signs is $35.00.

The first five letters or numbers are included in the minimum charge; there is a $4 charge for each additional character.

If the sign is oak, add $20.00. No charge is added for pine.

Black or white characters are included in the minimum charge; there is an additional $15 charge for gold-leaf lettering.

1. Open the file named HouseSign.vb using Notepad or the text editor of your choice.
2. You need to declare variables for the following, and initialize them where specified:
   - A variable for the cost of the sign initialized to 0.00
   - A variable for the color of the characters initialized to "gold"
   - A variable for the wood type initialized with the value "oak"
   - A variable for the number of characters initialized with the value 8
3. Write the rest of the program using assignment statements and If statements as appropriate. The output statements are written for you.
4. Compile the program.
5. Execute the program. Your output should be: The charge for this sign is $82.

Note that you do not see a decimal point or digits after the decimal point. You learn how to include a decimal point and specify the number of places after the decimal point in Chapter 9 of this book.

The If Then Else Statement

The If Then Else statement is a dual-path or dual-alternative decision statement. That is, your program will take one of two paths as a result of evaluating an expression in an If Then Else statement.

The syntax for writing an If Then Else statement in Visual Basic is as follows:

```vb
If expression Then
    statementA
Else
    statementB
End If
```

When the compiler encounters an If Then Else statement, the expression following the keyword If is evaluated. If the expression evaluates to True, then the computer executes `statementA`. Otherwise, if the expression evaluates to False, the computer executes `statementB`. Both `statementA` and `statementB` can be simple statements or block statements. Regardless of which path is taken in a program, the statement following the If Then Else statement is the next one to execute.
The following code sample illustrates an If Then Else statement written in Visual Basic:

```vbnet
Dim HoursWorked As Integer = 45
Dim Rate As Double = 15.00
Dim GrossPay As Double

Dim Overtime As String = "Yes"
Const HOURS_IN_WEEK As Integer = 40
Const OVERTIME_RATE As Double = 1.5
If HoursWorked > HOURS_IN_WEEK Then
    Overtime = "Yes"
    GrossPay = HOURS_IN_WEEK * Rate + 
               (HoursWorked - HOURS_IN_WEEK) * 
               OVERTIME_RATE * Rate
Else
    Overtime = "No"
    GrossPay = HoursWorked * Rate
End If
System.Console.WriteLine("Overtime: " & Overtime)
```

In the preceding code, the value of the variable named HoursWorked is tested to see if it is greater than HOURS_IN_WEEK.

HOURS_IN_WEEK is a constant that is initialized with the value 40, and OVERTIME_RATE is a constant that is initialized with the value 1.5.

You use the greater than comparison operator (>) to make the comparison. If the expression HoursWorked > HOURS_IN_WEEK evaluates to True, then the statement that assigns the string constant "Yes" to the variable named Overtime executes, followed by another statement that calculates the employee's gross pay, including overtime pay, and assigns the calculated value to the variable named GrossPay.

If the expression HoursWorked > HOURS_IN_WEEK evaluates to False, then a different path is followed, and the second block statement following the keyword Else executes. This block statement contains one statement that assigns the string constant "No" to the variable named Overtime, and another statement that calculates the employee's gross pay with no overtime, and assigns the calculated value to the variable named GrossPay.

Regardless of which path is taken in this code, the next statement to execute is the output statement System.Console.WriteLine("Overtime: " & Overtime) immediately followed by the output statement System.Console.WriteLine("Gross Pay: $" & GrossPay).

Exercise 4-2: Understanding If Then Else Statements

In this exercise, you use what you have learned about writing If Then Else statements. This example program was written to calculate customer charges for a telephone company. The
telephone company charges 25 cents per minute for calls outside of the customer's area code that last over 10 minutes. All other calls are 10 cents per minute. Study the following code and then answer Questions 1-4.

' Telephone.vb - This program determines telephone call charges.
Option Explicit On
Option Strict On
Module Telephone
    Sub Main()
        ' Work done in the housekeeping() procedure
        Dim CustAC As Integer
        Dim CustNumber As Integer
        Dim CalledAC As Integer
        Dim CalledNumber As Integer
        Dim CallMinutes As Integer
        Dim CallCharge As Double
        Dim MAX_MINS As Integer = 10
        Const CHARGE_1 As Double = 0.25
        Const CHARGE_2 As Double = 0.10
        ' Work done in the detailLoop() procedure
        CustAC = 847
        CustNumber = 5551234
        CalledAC = 630
        CalledNumber = 5557890
        CallMinutes = 50
        If CalledAC <> CustAC And CallMinutes > MAX_MINS Then
            CallCharge = CallMinutes * CHARGE_1
        Else
            CallCharge = CallMinutes * CHARGE_2
        End If
        ' Work done in the endOfJob() procedure
        System.Console.WriteLine("Customer Number: " & _
            CustAC & ":" & CustNumber)
        System.Console.WriteLine("Called Number: " & _
            CalledAC & ":" & CalledNumber)
        System.Console.WriteLine("The charge for this call is $" & CallCharge)
    End Sub
End Module

1. What is the exact output when this program executes?

2. What is the exact output if the value of CallMinutes is changed to 20?
3. What is the exact output if the expression in the If statement is changed to CallMinutes >= MAX_MINS?

4. What is the exact output if the variable named CalledAC is assigned the value 847 rather than the value 630?

Lab 4-2: Understanding If Then else Statements

In this lab, you complete a prewritten Visual Basic program that computes the largest and smallest of three integer values. The three values are -50, 53, and 78.

1. Open the file named LargeSmall.vb using Notepad or the text editor of your choice.
2. Two variables named Largest and Smallest are declared for you. Use these variables to store the largest and smallest of the three integer values. You must decide what other variables you will need and initialize them if appropriate.
3. Write the rest of the program using assignment statements, If statements, or If Then Else statements as appropriate. There are comments in the code that tell you where you should write your statements. The output statement is written for you.
4. Compile the program.
5. Execute the program. Your output should be:
   The largest value is 78
   The smallest value is -50

Nested If Statements

You can nest If statements to create a multipath decision statement. When you nest If statements, you include an If statement within another If statement. This is helpful in programs in which you want to provide more than two possible paths.

The syntax for writing a nested If statement in Visual Basic is as follows:

If expressionA Then
    statementA
Else If expressionB Then
    statementB
Else
    statementC
End If
This is called a nested If statement because the second If statement is a part of the first If statement. This is easier to see if the example is changed as follows:

\[
\text{If } expressionA \text{ Then} \\
\quad \text{statementA} \\
\text{Else} \\
\quad \text{If } expressionB \text{ Then} \\
\quad \quad \text{statementB} \\
\quad \text{Else} \\
\quad \quad \text{statementC} \\
\quad \text{End If} \\
\text{End If}
\]

Now you will see how a nested If statement works. As shown in Figure 4-3, if \(expressionA\) evaluates to True, then the computer executes \(statementA\). If \(expressionA\) evaluates to False, then the computer will evaluate \(expressionB\). If \(expressionB\) evaluates to True, then the computer will execute \(statementB\). If both \(expressionA\) and \(expressionB\) evaluate to False, then the computer will execute \(statementC\). Regardless of which path is taken in this code, the statement following the If Then Else statement is the next one to execute.

![Diagram of nested If statement](image)

**Figure 4-3** Evaluation of a nested If statement

The Visual Basic code sample that follows illustrates a nested If statement:

```vbnet
If EmpDept <= 3 Then
    SupervisorName = "Dillon"
Else If EmpDept <= 7 Then
    SupervisorName = "Escher"
Else
    SupervisorName = "Fontana"
End If
System.Console.WriteLine("Supervisor: " & SupervisorName)
```
When you read the preceding code, you can assume that a department number is never less than 1. If the value of the variable named EmpDept is less than or equal to the value 3 (in the range of values from 1 to 3), then the value “Dillon” is assigned to the variable named SupervisorName. If the value of EmpDept is not less than or equal to 3, but it is less than or equal to 7 (in the range of values from 4 to 7), then the value "Escher" is assigned to the variable named SupervisorName. If the value of EmpDept is not in the range of values from 1 to 7, then the value "Fontana" is assigned to the variable named SupervisorName. As you can see, there are three possible paths this program could take when the nested If statement is encountered. Regardless of which path the program takes, the next statement to execute is the output statement System.Console.WriteLine("Supervisor: " & SupervisorName).

Exercise 4-3: Understanding Nested If Statements

In this exercise, you use what you have learned about writing nested If statements. This example program was written for the Woof Wash dog-grooming business to calculate a total charge for services rendered. Woof Wash charges $12 for a bath, $10 for a cut, and $7 to trim nails. Study the following code and then answer Questions 1–3.

' WoofWash.vb - This program determines if a doggy service is provided and prints the charge.
Option Explicit On
Option Strict On
Module WoofWash
Sub Main()
  ' Work done in the housekeeping() procedure
  Dim Service As String
  Const SERVICE_1 As String = "bath"
  Const SERVICE_2 As String = "cut"
  Const SERVICE_3 As String = "trim nails"
  Dim Charge As Double
  Const BATH_CHARGE As Double = 12.00
  Const CUT_CHARGE As Double = 10.00
  Const NAIL_CHARGE As Double = 7.00
  Service = InputBox$("Enter service: ")
  ' Work done in the detailLoop() procedure
  If Service = SERVICE_1 Then
    Charge = BATH_CHARGE
  Else If Service = SERVICE_2 Then
    Charge = CUT_CHARGE
  Else If Service = SERVICE_3 Then
    Charge = NAIL_CHARGE
  Else
    Charge = 0.00
  End If
  ' Work done in the endOfJob() procedure
  If Charge > 0.00 Then
    System.Console.WriteLine(_
"The charge for a doggy " & _
Service & " is $" & Charge & ".")
End Sub
End Module

Decision Statements

Else
    System.Console.WriteLine(_
        "We do not perform the " & _
        Service & " service.")
End If
End Sub
End Module

1. What is the exact output when this program executes if the user enters "bath"?

2. What is the exact output when this program executes if the user enters "shave"?

3. What is the exact output when this program executes if the user enters "BATH"?

Lab 4-3: Understanding Nested If Statements

In this lab, you complete a prewritten Visual Basic program that calculates an employee's productivity bonus and prints the employee's name and bonus. Bonuses are calculated based on an employee's productivity score as shown in Table 4-4. A productivity score is calculated by first dividing an employee's transactions dollar value by the number of transactions and then dividing the result by the number of shifts worked.

<table>
<thead>
<tr>
<th>Productivity Score</th>
<th>Bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= 30</td>
<td>$50</td>
</tr>
<tr>
<td>31 - 69</td>
<td>$75</td>
</tr>
<tr>
<td>70 - 199</td>
<td>$100</td>
</tr>
<tr>
<td>&gt;= 200</td>
<td>$200</td>
</tr>
</tbody>
</table>

Table 4-4 Employee productivity scores and bonuses

1. Open the file named EmployeeBonus.vb using Notepad or the text editor of your choice.
2. Variables have been declared for you, and the input statements and output statements have been written. Read them over carefully before you proceed to the next step.