In this chapter, you will learn how programs use If...Then, If...Then...Else, and If...Then...ElseIf statements to make decisions. You will learn how to compare values using relational operators and build complex comparisons using logical operators. You will be introduced to the Select Case statement, radio buttons (which allow the user to select one choice from many possible choices), and check boxes (which allow the user to make on/off or yes/no types of selections). You will learn more about message boxes, which display messages to the user, class-level variables, and the process of input validation.

4.1 The Decision Structure

**CONCEPT:** The decision structure allows a program's logic to have more than one path of execution.

In the programs you have written so far, statements execute sequentially. This means that statements are executed one after the other, in the order in which they appear.
You might think of sequentially executed statements as the steps you take as you walk down a road. To complete the journey, you must start at the beginning and take each step, one after the other, until you reach your destination. This is illustrated in Figure 4-1.

**Figure 4-1 Sequence instruction**

Private Sub btnCalcTotal_Click(ByVal sender...)  ' Calculate the total of two values
    Dim intValue1 As Integer
    Dim intValue2 As Integer
    Dim intTotal As Integer
    intValue1 = CInt(txtValue1.Text)
    intValue2 = CInt(txtValue2.Text)
    intTotal = intValue1 + intValue2
    lblTotal.Text = intTotal.ToString()
End Sub

This type of code is called a **sequence structure** because the statements are executed in sequence, without branching in another direction. Programs often need more than one path of execution because many algorithms require a program to execute some statements only under certain circumstances. This can be accomplished with a **decision structure**.

### Decision Structures in Flowcharts and Pseudocode

In a decision structure’s simplest form, an action is taken only when a condition, also known as a **Boolean expression**, equals True. If the condition equals False, the action is not performed. Figure 4-2 shows a flowchart segment for a decision structure. The diamond symbol represents a yes/no question, or a true/false condition. If the answer to the question is yes (or if the condition is true), the program flow follows one path. If the answer to the question is no (or the condition is false), the program flow follows another path.

In the flowchart, the action **Wear a coat** is performed only when it is cold outside. If it is not cold outside, the action is skipped. The action is **conditionally executed** because it is only performed when a certain condition (**cold outside**) exists. Figure 4-3 shows a more elaborate flowchart, where three actions are taken, only when it is cold outside.

Decision structures can also be expressed as pseudocode. For example, the decision structure shown in Figure 4-2 can be expressed as

```
If it is cold outside Then
    Wear a coat.
End If
```

The **End If** statement marks the end of the decision structure in pseudocode. The statements appearing between **If...Then** and **End If** are executed only when the condition that is being tested is true. In our example, “it is cold outside” is the condition. The decision structure shown in Figure 4-3, which conditionally executes three actions, can be expressed as

```
If it is cold outside Then
    Wear a coat.
    Wear a toboggan.
    Wear gloves.
End If
```
4.2 The If...Then Statement

**CONCEPT:** The **If...Then** statement causes other statements to execute only under a certain condition.

One way to code a decision structure in Visual Basic is with the **If...Then** statement. Here is the general form of the **If...Then** statement.

```
If condition Then
  statement
  (more statements may follow)
End If
```

The **If...Then** statement is really very simple: if the **condition** is true, the statement or statements that appear between the **If** and the **End If** are executed. Otherwise, the statements are skipped.

**TIP:** A condition is often referred to by programmers as a Boolean expression. The word “Boolean” is taken from the mathematician George Boole, who invented a system of mathematics known as Boolean algebra.

---

**Figure 4-2** Simple decision structure flowchart

```
Is it cold outside?
Yes
Wear a coat.
No
```

**Figure 4-3** Three-action decision structure flowchart

```
Is it cold outside?
Yes
Wear a coat.
No
Wear a toboggan.
No
Wear gloves.
```
Chapter 4  Making Decisions and Working with Strings

Using Relational Operators to Form Conditions

Typically, the condition tested by an `If...Then` statement is formed with a relational operator. A relational operator determines whether a specific relationship exists between two values. For example, the greater than operator (`>`), determines whether one value is greater than another. The equal to operator (`=`) determines whether two values are equal. Table 4-1 lists the Visual Basic relational operators.

Table 4-1  Visual Basic relational operators

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

All relational operators are binary, which means they use two operands. Here is an example of an expression using the greater than operator:

\[ \text{length} > \text{width} \]

This expression is called a relational expression. A relational expression consists of a relational operator and its operands. This one is used to determine whether `length` is greater than `width`. The following expression determines whether `length` is less than `width`:

\[ \text{length} < \text{width} \]

Table 4-2 shows examples of several relational expressions that compare the variables `x` and `y`.

Table 4-2  Relational expressions

<table>
<thead>
<tr>
<th>Relational Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x &gt; y</code></td>
<td>Is ( x ) greater than ( y )?</td>
</tr>
<tr>
<td><code>x &lt; y</code></td>
<td>Is ( x ) less than ( y )?</td>
</tr>
<tr>
<td><code>x &gt;= y</code></td>
<td>Is ( x ) greater than or equal to ( y )?</td>
</tr>
<tr>
<td><code>x &lt;= y</code></td>
<td>Is ( x ) less than or equal to ( y )?</td>
</tr>
<tr>
<td><code>x = y</code></td>
<td>Is ( x ) equal to ( y )?</td>
</tr>
<tr>
<td><code>x &lt;&gt; y</code></td>
<td>Is ( x ) not equal to ( y )?</td>
</tr>
</tbody>
</table>

TIP: A relational expression is a specific type of Boolean expression.

Relational expressions can only be evaluated as true or false. If `x` is greater than `y`, the expression `x > y` is true, while the expression `x < y` is false.

The `=` operator, when used in a relational expression, determines whether the operand on its left is equal to the operand on its right. If both operands have the same value, the
expression is true. Assuming that \( a \) is 4, the expression \( a = 4 \) is true and the expression \( a = 2 \) is false.

There are two operators that can test more than one relationship at the same time. The >= operator determines whether the operand on its left is greater than or equal to the operand on the right. Assuming that \( a \) is 4, \( b \) is 6, and \( c \) is 4, the expressions \( b \geq a \) and \( a \geq c \) are true, and \( a \geq 5 \) is false. When using this operator, the > symbol must precede the = symbol, with no space between them.

The <= operator determines whether the left operand is less than or equal to the right operand. Once again, assuming that \( a \) is 4, \( b \) is 6, and \( c \) is 4, both \( a \leq c \) and \( b \leq 10 \) are true, but \( b \leq a \) is false. When using this operator, the < symbol must precede the = symbol, with no space between them.

The <> operator is the not equal operator. It determines whether the operand on its left is not equal to the operand on its right, which is the opposite of the = operator. As before, assuming \( a \) is 4, \( b \) is 6, and \( c \) is 4, both \( a \neq b \) and \( b \neq c \) are true because \( a \) is not equal to \( b \) and \( b \) is not equal to \( c \). However, \( a \neq c \) is false because \( a \) is equal to \( c \). Values compared by a relational expression need not be exactly the same type. Suppose we compare a variable of type Single to an integer constant, as in the following:

\[
sngTemperature > 40
\]

In this example, the integer 40 is temporarily converted to a Single so the comparison can take place. You do not have to worry about doing this conversion. It is carried out automatically by the Visual Basic compiler. Similarly, we might want to compare a Double to a Single, as in the following:

\[
dblTemperature < sngBoilingPoint
\]

The value of \( sngBoilingPoint \) is automatically converted to type Double so the values can be compared.

**Putting It All Together**

The following If...Then statement uses the > operator to determine whether \( decSales \) is greater than 50000. If that condition is true, the Boolean variable \( blnGetsBonus \) is set to \( True \).

```vba
If decSales > 50000 Then
    blnGetsBonus = True
End If
```

The following example conditionally executes multiple statements.

```vba
If decSales > 50000 Then
    blnGetsBonus = True
    decCommissionRate = 0.12
    intDaysOff = intDaysOff + 1
End If
```

Here are some specific rules to remember about the If...Then statement:

- The words If and Then must appear on the same line.
- Nothing other than a comment can appear after the Then keyword, on the same line.
- The End If statement must be on a line by itself. Only a comment may follow it on the same line.

Tutorial 4-1 examines an application that uses the If...Then statement.
Tutorial 4-1:
Examining an application that uses the If...Then statement

Step 1: Open the Test Score Average 1 project from the student sample programs folder named Chap4\Test Score Average 1.

Step 2: Run the application. The form appears, as shown in Figure 4-4.

Step 3: Enter the following test scores in the three text boxes: 80, 90, 75.

Step 4: Click the Calculate Average button. The average test score is displayed.

Step 5: Click the Clear button, and then enter the following test scores in the three text boxes: 100, 97, 99.

Step 6: Click the Calculate Average button. This time, in addition to the average test score being displayed, a congratulatory message appears. The form appears, as shown in Figure 4-5.

Figure 4-4 Test Score Average form

Figure 4-5 Average and message displayed

Step 7: Click the Exit button to terminate the application.

Step 8: Open the Code window and find the btnCalculate_Click event procedure. The code is as follows:

```vbnet
Private Sub btnCalculate_Click(ByVal eventSender As System.Object, ByVal eventArgs As System.EventArgs) Handles btnCalculate.Click
    ' This procedure calculates and displays the average test score. If the score is high, it displays a congratulatory message.
    Dim sngScore1 As Single
    Dim sngScore2 As Single
    Dim sngScore3 As Single
    Dim sngAverage As Single
    ' Copy the scores into the variables
    sngScore1 = CSng(txtScore1.Text)
    sngScore2 = CSng(txtScore2.Text)
    sngScore3 = CSng(txtScore3.Text)
```

GADDIS_CH04 12/16/08 6:47 PM Page 198
4.2 The If...Then Statement

Programming Style and the If...Then Statement

In each If...Then statement we have looked at, conditionally executed statements are indented. This is not a syntax requirement, but a programming style convention. For example, compare the following statements:

```vbnet
If decSales > 50000 Then
    blnGetsBonus = True
    decCommissionRate = 0.12
    intDaysOff = intDaysOff + 1
End If

If decSales > 50000 Then
    blnGetsBonus = True
    decCommissionRate = 0.12
    intDaysOff = intDaysOff + 1
End If
```

Both If...Then statements produce the same result. The first example, however, is more readable than the second because the conditionally executed statements are indented.

**NOTE:** Visual Basic automatically indents conditionally executed statements when you type an If...Then statement. If this feature has been turned off, you can turn it on by clicking Tools on the menu bar, then clicking Options. In the Options dialog box, perform the following:

- Click the Show all settings check box. Then, click Text Editor in the left pane, then click Basic, then click Tabs. Make sure Smart is selected in the dialog box under Indenting.
- In the left pane, click VB Specific. Make sure Automatic Insertion of end constructs and Pretty listing (reformatting) of code are both checked.

Using Relational Operators with Math Operators

It is possible to use a relational operator and math operators in the same expression. Here is an example:

```vbnet
If intX + intY > 20 Then
    lblMessage.Text = "It is true!"
End If
```
When a relational operator appears in the same expression as one or more math operators, the math operators always execute first. In this statement, the + operator adds intX and intY. The result is compared to 20 using the > operator. Here is another example:

If intX + intY > intA - intB Then
  lblMessage.Text = "It is true!"
End If

In this statement, the result of intX + intY is compared, using the > operator, to the result of intA - intB.

Most programmers prefer to use parentheses to clarify the order of operations. Relying on operator precedence rules is risky because the rules are hard to remember. Here is a preferred way to write the foregoing If statement:

If (intX + intY) > (intA - intB) Then
  lblMessage.Text = "It is true!"
End If

Using Function Calls with Relational Operators

It is possible to compare the return value of a function call with another value, using a relational operator. Here is an example:

If CInt(txtInput.Text) < 100 Then
  lblMessage.Text = "It is true!"
End If

This If...Then statement calls the CInt function to get the numeric value of txtInput.Text. The function's return value is compared to 100 by the < operator. If the result of CInt(txtInput.Text) is less than 100, the assignment statement is executed.

Using Boolean Variables as Flags

A flag is a Boolean variable that signals when some condition exists in the program. When the flag is set to False, it indicates the condition does not yet exist. When the flag is set to True, it means the condition does exist. Look at the following code, which uses a Boolean variable named blnQuotaMet.

If blnQuotaMet Then
  lblMessage.Text = "You have met your sales quota"
End If

The preceding statement assigns the string "You have met your sales quota" to lblMessage.Text if the Boolean variable equals True. If blnQuotaMet is False, the assignment statement is not executed. It is not necessary to use the = operator to compare the variable to True. The statement is equivalent to the following:

If blnQuotaMet = True Then
  lblMessage.Text = "You have met your sales quota"
End If

Checkpoint

4.1 Assuming x is 5, y is 6, and z is 8, indicate whether each of the following relational expressions equals True or False:

a. x = 5     T   F
b. 7 <= (x + 2)     T   F
c. z < 4     T   F
d. (2 + x) <> y     T   F
e. z <> 4     T   F
f. x >= 6     T   F
g. x <= (y * 2)     T   F
4.2 In the following If statement, assume that blnIsInvalid is a Boolean variable. Exactly what condition is being tested?
   If blnIsInvalid Then
     ' Do something
   End If

4.3 Do both of the following If...Then statements perform the same operation?
   If decSales > 10000 Then
     decCommissionRate = 0.15
   End If
   If decSales > 10000 Then
     decCommissionRate = 0.15
   End If

4.4 Of the two If...Then statements shown in Checkpoint 4.3, which is preferred, and why?

The If...Then...Else Statement

CONCEPT: The If...Then...Else statement executes one group of statements if the condition (or Boolean expression) is true and another group of statements if the condition is false.

The If...Then...Else statement is an expansion of the If...Then statement. Here is its format:

   If condition Then
     statement
     (more statements may follow)
   Else
     statement
     (more statements may follow)
   End If

As in an If...Then statement, a Boolean expression is evaluated. If the expression is true, a statement or group of statements is executed. If the expression is false, a separate group of statements is executed, as in the following.

   If sngTemperature < 40 Then
     lblMessage.Text = "A little cold, isn't it?"
   Else
     lblMessage.Text = "Nice weather we're having!"
   End If

The Else statement specifies a statement or group of statements to be executed when the expression is false. In the preceding example, when the expression sngTemperature < 40 is false, the statement appearing after Else is executed. The conditionally executed statement(s) in the Else part are indented.

The If...Then...Else statement follows only one of the two paths. If you think of the statements in a computer program as steps taken down a road, consider the If...Then...Else statement as a fork in the road. Instead of being a momentary detour, like an If...Then statement, the If...Then...Else statement causes the program execution to follow one of two exclusive paths. Figure 4-6 shows a flowchart for this type of decision structure.
The logic shown in the flowchart in Figure 4-6 can also be expressed in pseudocode:

```pseudocode
If sngTemperature < 40 Then
    lblMessage.Text = "A little cold, isn't it?"
Else
    lblMessage.Text = "Nice weather we’re having!"
End If
```

In Tutorial 4-2 you complete an application that uses the `If...Then...Else` statement.

---

**Tutorial 4-2:**

**Completing an application that uses the `If...Then...Else` statement**

**Step 1:** Open the `Test Score Average 2` project from the student sample programs folder named `Chap4\Test Score Average 2`. (This is a modification of the `Test Score Average 1` application from Tutorial 4-1.)

**Step 2:** Double-click the `Calculate Average` button. The `Code` window will open and show the `btnCalculate_Click` event procedure. Type the following code:

```csharp
Private Sub btnCalculate_Click(ByVal eventSender As System.Object, ByVal eventArgs As System.EventArgs) Handles btnCalculate.Click
    ' This procedure calculates and displays the average test score. If the score is high, it displays a congratulatory message.
    Const sngGOOD_SCORE As Single = 95
    Dim sngScore1 As Single
    Dim sngScore2 As Single
    Dim sngScore3 As Single
    Dim sngAverage As Single
```
4.3 The If...Then...Else Statement

`' Copy the scores into the variables
Try
  sngScore1 = CSng(txtScore1.Text)
sngScore2 = CSng(txtScore2.Text)
sngScore3 = CSng(txtScore3.Text)
Catch Ex As Exception
  MessageBox.Show("Test scores must be numeric", "Error")
  Return
End Try
' Calculate and display the average
sngAverage = (sngScore1 + sngScore2 + sngScore3) / 3
lblAverage.Text = sngAverage.ToString("n")
' If the score is high, give the student praise.
' Otherwise, give some encouragement.
If sngAverage > sngGOOD_SCORE Then
  lblMessage.Text = "Congratulations! Great Job!"
Else
  lblMessage.Text = "Keep trying!"
End If
End Sub

Step 3: Save the project.

Step 4: Run the application and input the following test scores in the three text boxes: 80, 90, 75.

Step 5: Click the Calculate Average button. As shown in Figure 4-7, the average test score is displayed, and the message Keep trying! appears.

Figure 4-7 Test Score Average form with message displayed

Step 6: Click the Clear button, and then enter the following test scores in the three text boxes: 100, 97, 99.

Step 7: Click the Calculate Average button. This time, the message Congratulations! Great job! appears.

Step 8: Click the Exit button to terminate the application.
**Checkpoint**

4.5 Look at each of the following code segments. What value will the `If...Then...Else` statements store in the variable `intY`?

a. `intX = 0`
   
   ```
   If intX < 1 Then
     intY = 99
   Else
     intY = 0
   End If
   ```

b. `intX = 100`
   
   ```
   If intX <= 1 Then
     intY = 99
   Else
     intY = 0
   End If
   ```

c. `intX = 0`
   
   ```
   If intX <> 1 Then
     intY = 99
   Else
     intY = 0
   End If
   ```

---

**The `If...Then...ElseIf` Statement**

**CONCEPT:** The `If...Then...ElseIf` statement is like a chain of `If...Then...Else` statements. They perform their tests, one after the other, until one of them is found to be true.

We make certain mental decisions by using sets of different but related rules. For example, we might decide which type of coat or jacket to wear by consulting the following rules:

- If it is very cold, wear a heavy coat
- Else, if it is chilly, wear a light jacket
- Else, if it is windy, wear a windbreaker
- Else, if it is hot, wear no jacket

The purpose of these rules is to decide on one type of outer garment to wear. If it is cold, the first rule dictates that a heavy coat must be worn. All the other rules are then ignored. If the first rule does not apply (if it isn’t cold) the second rule is consulted. If that rule does not apply, the third rule is consulted, and so on.

**TIP:** When logic rules are not expressed correctly by a program, the result is called a **logic error**. The Visual Studio Debugger can help you to identify logic errors by letting you walk through the program code, one line at a time.

The way these rules are connected is very important. If they were consulted individually, we might go out of the house wearing the wrong jacket or, possibly, more than one jacket. For instance, if it is windy, the third rule says to wear a windbreaker. What if it is both windy and very cold? Will we wear a windbreaker? A heavy coat? Both? Because of the order in which the rules are consulted, the first rule will determine that a heavy coat is needed. The remaining rules will not be consulted, and we will go outside wearing the most appropriate garment.

This type of decision making is also common in programming. In Visual Basic, it is accomplished with the `If...Then...ElseIf` statement. Here is its general format:
4.4 The If...Then...ElseIf Statement

If condition Then
    statement
    (more statements may follow)
ElseIf condition Then
    statement
    (more statements may follow)
    (put as many ElseIf statements as necessary)
Else
    statement
    (more statements may follow)
End If

This construction is like a chain of If...Then...Else statements. The Else part of one statement is linked to the If part of another. The chain of If...Then...Else statements becomes one long statement. In Tutorial 4-3, you complete an application that uses the If...Then...ElseIf statement.

Tutorial 4-3:
Completing an application that uses the If...Then...ElseIf statement

In this tutorial, you will begin with the program from Tutorial 4-2 and add controls and program code that display the student’s letter grade (A, B, C, D, F).

Step 1: Open the Test Score Average 2 project you modified in Tutorial 4-2.

Step 2: Drag the form’s border downward about one-half inch, and drag the lblMessage control and the three button controls downward on the form to make space for a new row of controls.

Step 3: Drag the lower border of the group box downward about one-half inch to make room for a label that will display the student’s letter grade.

Step 4: Inside the group box add the new Label controls shown in Figure 4-8. When you add the label on the left, set its Text property to Grade: When you add the label on the right, set its Name property to lblGrade, its AutoSize property to False, and set its BorderStyle property to Fixed3D.

Figure 4-8 Adding the Grade label inside the group box
Step 5: Double-click the Calculate Average button. The Code window will open and show the btnCalculate_Click event procedure. Add the following code shown in bold to the procedure:

```vbnet
Private Sub btnCalculate_Click(ByVal eventSender As System.Object, ByVal eventArgs As System.EventArgs) Handles btnCalculate.Click
    ' This procedure calculates and displays the average test score. If the score is high, it displays a congratulatory message.
    Const sngGOOD_SCORE As Single = 95
    Dim sngScore1 As Single
    Dim sngScore2 As Single
    Dim sngScore3 As Single
    Dim sngAverage As Single

    ' Copy the scores into the variables
    Try
        sngScore1 = CSng(txtScore1.Text)
        sngScore2 = CSng(txtScore2.Text)
        sngScore3 = CSng(txtScore3.Text)
    Catch Ex As Exception
        MessageBox.Show("Test scores must be numeric","Error")
        Return
    End Try

    ' Calculate and display the average
    sngAverage = (sngScore1 + sngScore2 + sngScore3) / 3
    lblAverage.Text = sngAverage.ToString("n")

    ' Calculate and display the letter grade.
    If sngAverage < 60 Then
        lblGrade.Text = "F"
    ElseIf sngAverage < 70 Then
        lblGrade.Text = "D"
    ElseIf sngAverage < 80 Then
        lblGrade.Text = "C"
    ElseIf sngAverage < 90 Then
        lblGrade.Text = "B"
    ElseIf sngAverage <= 100 Then
        lblGrade.Text = "A"
    End If

    ' If the score is high, give the student praise. Otherwise, give some encouragement.
    If sngAverage > sngGOOD_SCORE Then
        lblMessage.Text = "Congratulations! Great Job!"
    Else
        lblMessage.Text = "Keep trying!"
    End If
End Sub
```

The If...Then...ElseIf statement has a number of notable characteristics. Let's analyze how it works in the Test Score application. First, the relational expression average < 60 is tested:

```vbnet
If sngAverage < 60 Then
    lblGrade.Text = "F"
```

If sngAverage is less than 60, F is assigned to lblGrade.Text, and the rest of the ElseIf statements are ignored. If sngAverage is not less than 60, the next ElseIf statement executes:

```
If sngAverage < 60 Then
    lblGrade.Text = "F"
ElseIf sngAverage < 70 Then
    lblGrade.Text = "D"
```

The first If...Then statement filtered out all grades less than 60, so when this ElseIf statement executes, sngAverage must be 60 or greater. If sngAverage is less than 70, D is assigned to lblGrade.Text and the remaining ElseIf statements are ignored. The chain of events continues until one of the expressions is true, or the End If statement is encountered. Figure 4-9 uses a flowchart to describe the logic.

**Figure 4-9** Flowchart for determining the student’s letter grade

---

**Step 6:** Save the project, run the application, and input the following test scores in the text boxes: 80, 90, 75.
Step 7: Click the Calculate Average button. The average test score and letter grade are displayed, along with the message Keep trying! (see Figure 4-10).

Step 8: Click the Exit button to terminate the application.

Figure 4-10 Student grade displayed

The Difference between the If...Then...ElseIf Statement and a Series of If...Then Statements

The execution of each ElseIf statement in the structure depends on all the conditions tested before it being false. The conditionally executed statements following an ElseIf are executed when the conditional expression following the ElseIf is true and all previous expressions are false. To demonstrate how this interconnection works, let’s look at a version of the program that uses independent If...Then statements instead of an If...Then...ElseIf statement. Tutorial 4-4 compares an If...Then...ElseIf statement to a series of If...Then statements.

Tutorial 4-4:
Comparing an If...Then...ElseIf statement to a series of If...Then statements

Step 1: Open the Test Score Average 3 project from the student sample programs folder named Chap4\Test Score Average 3.

Step 2: Run the application. When the form appears, enter the following test scores in the three text boxes: 40, 40, 40.

Step 3: Click the Calculate Average button. Despite the low scores, the grade A is displayed. Click the Clear button.

Step 4: Experiment by entering more test scores and clicking the Calculate Average button. Notice that regardless of the scores you enter, the application always reports a grade of A.
Step 5: When finished, click the Exit button to terminate the application.

Step 6: Open the Code window and find the btnCalculate_Click event handler. Notice that instead of an If...Then...ElseIf statement, this procedure uses a series of If...Then statements to assign a letter grade. The code is as follows.

If sngAverage < 60 Then
    lblGrade.Text = "F"
End If
If sngAverage < 70 Then
    lblGrade.Text = "D"
End If
If sngAverage < 80 Then
    lblGrade.Text = "C"
End If
If sngAverage < 90 Then
    lblGrade.Text = "B"
End If
If sngAverage <= 100 Then
    lblGrade.Text = "A"
End If

In this procedure, all the If...Then statements execute because they are individual statements. When you ran the tutorial, you entered three scores of 40, which give an average of 40. Because this is less than 60, the first If...Then statement causes F to be assigned to lblGrade.Text.

If sngAverage < 60 Then
    lblGrade.Text = "F"
End If
Because the next statement is If...Then instead of ElseIf, it executes. sngAverage is also less than 70, therefore it causes D to be assigned to lblGrade.Text. D overwrites the F that was previously stored there.

Using a Trailing Else

There is one minor problem with the test averaging applications shown so far: What if the user enters a test score greater than 100? The If...Then...ElseIf statement in the Test Score Average 2 project handles all scores through 100, but none greater. Figure 4-11 shows the form when the user enters values greater than 100.
The program does not give a letter grade because there is no code to handle a score greater than 100. Assuming that any grade over 100 is invalid, we can fix the program by placing an Else at the end of the If...Then...ElseIf statement, as follows:

```vbnet
' Calculate and display the letter grade.
If sngAverage < 60 Then
    lblGrade.Text = "F"
ElseIf sngAverage < 70 Then
    lblGrade.Text = "D"
ElseIf sngAverage < 80 Then
    lblGrade.Text = "C"
ElseIf sngAverage < 90 Then
    lblGrade.Text = "B"
ElseIf sngAverage <= 100 Then
    lblGrade.Text = "A"
Else
    lblGrade.Text = "Invalid"
End If
```

The trailing Else catches any value that falls through the cracks. It provides a default response when the If...Then or none of the ElseIf statements finds a true condition.

**TIP:** When writing an If...Then...ElseIf statement, code the structure of the statement first, identifying all the conditions to be tested. For example, the code in our example might initially be written as follows:

```vbnet
If sngAverage < 60 Then
    ElseIf sngAverage < 70 Then
    ElseIf sngAverage < 80 Then
    ElseIf sngAverage < 90 Then
    ElseIf sngAverage <= 100 Then
    Else
End If
```

This creates the framework of the statement. Next, insert the conditionally executed statements, as shown in bold in the following code:

```vbnet
If sngAverage < 60 Then
    lblGrade.Text = "F"
ElseIf sngAverage < 70 Then
    lblGrade.Text = "D"
```

The program does not give a letter grade because there is no code to handle a score greater than 100. Assuming that any grade over 100 is invalid, we can fix the program by placing an Else at the end of the If...Then...ElseIf statement, as follows:
ElseIf sngAverage < 80 Then
    lblGrade.Text = "C"
ElseIf sngAverage < 90 Then
    lblGrade.Text = "B"
ElseIf sngAverage <= 100 Then
    lblGrade.Text = "A"
Else
    lblGrade.Text = "Invalid"
End If

A good design approach is to decide which conditions must be tested first, and then decide what actions must be taken for each condition.

CheckPoint

4.6 The following If...Then... ElseIf statement has several conditions that test the variable intX. Assuming intX equals 20, how many times will the following statement compare intX before it finds a condition that is true?

If intX < 10 Then
    intY = 0
ElseIf intX < 20 Then
    intY = 1
ElseIf intX < 30 Then
    intY = 2
ElseIf intX < 40 Then
    intY = 3
Else
    intY = -1
End If

4.7 In the following If...Then... ElseIf statement, if the variable intX equals 5, how many times will the code assign a value to intY?

If intX < 10 Then
    intY = 0
ElseIf intX < 20 Then
    intY = 1
ElseIf intX < 30 Then
    intY = 2
ElseIf intX < 40 Then
    intY = 3
End If

In the following set of If...Then statements, if the variable intX equals 5, how many times will the code assign a value to intY?

If intX < 10 Then
    intY = 0
End If
If intX < 20 Then
    intY = 1
End If
If intX < 30 Then
    intY = 2
End If
If intX < 40 Then
    intY = 3
End If
4.5 Nested If Statements

**CONCEPT:** A nested If statement is an If statement in the conditionally executed code of another If statement. (In this section, we use the term If statement to refer to an If...Then, If...Then...Else, or If...Then...ElseIf statement.)

A nested If statement is an If statement that appears inside another If statement. In Tutorial 4-5, you will examine an application that uses nested If statements. The application determines whether a bank customer qualifies for a special loan. The customer must meet one of the following qualifications:

- Earn $30,000 per year or more and have worked in his or her current job for more than two years.
- Have worked at his or her current job for more than five years.

**Tutorial 4-5:**
Completing an application with a nested If statement

**Step 1:** Open the Loan Qualifier project from the student sample programs folder named Chap4Loan Qualifier.

**Step 2:** Open Form1 in the Design window. It should appear as shown in Figure 4-12.

**Figure 4-12 Loan Qualifier application**

**Step 3:** Double-click the Check Qualifications button and insert the following bold statements in the btnCheckQual_Click procedure:

```vbnet
Private Sub btnCheckQual_Click(ByVal eventSender As System.Object, ByVal eventArgs As System.EventArgs) Handles btnCheckQual.Click

    ' Retrieve and convert the salary and years
    ' from the text boxes.
    Dim sngSalary As Single
    Dim intYearsOnJob As Integer
    Try
        sngSalary = CSng(txtSalary.Text)
        intYearsOnJob = CInt(txtYearsOnJob.Text)
    End Try
```


Catch ex As Exception
    MessageBox.Show("Salary and Years on Job must be valid numbers","Error")
    Return
End Try

The statements declare two variables and assign them the contents of the salary and years on job text boxes.

Step 4: Next, add the following nested If statement to the same procedure:

' Determine whether the applicant qualifies for the special loan.
If sngSalary > 30000 Then
    If intYearsOnJob > 2 Then
        lblMessage.Text = "The applicant qualifies."
    Else
        lblMessage.Text = "The applicant does not qualify."
    End If
Else
    If intYearsOnJob > 5 Then
        lblMessage.Text = "The applicant qualifies."
    Else
        lblMessage.Text = "The applicant does not qualify."
    End If
End If

Step 5: Save and run the application. Enter 45000 for salary and 3 for years at current job. Click the Check Qualifications button. The message The applicant qualifies should appear on the form.

Step 6: Enter 15000 for salary and 3 for years at current job. Click the Check Qualifications button. The message The applicant does not qualify appears on the form.

Step 7: Experiment with other values. When you are finished, click the Exit button to terminate the application.

Examining the Nested If Statement in More Depth

In the Loan Qualifier project, the outermost If statement tests the following condition:

If sngSalary > 30000 Then

If this condition is true, the nested If statement shown in bold is executed:

If sngSalary > 30000 Then
    If intYearsOnJob > 2 Then
        lblMessage.Text = "The applicant qualifies."
    Else
        lblMessage.Text = "The applicant does not qualify."
    End If
Else
    If intYearsOnJob > 5 Then
        lblMessage.Text = "The applicant qualifies."
    Else
        lblMessage.Text = "The applicant does not qualify."
    End If
End If
However, if the condition `sngSalary > 30000` is not true, the `Else` part of the outer-most `If` statement causes its nested `If` statement, shown in bold, to execute:

```vbnet
If sngSalary > 30000 Then
    If intYearsOnJob > 2 Then
        lblMessage.Text = "The applicant qualifies."
    Else
        lblMessage.Text = "The applicant does not qualify."
    End If
Else
    If intYearsOnJob > 5 Then
        lblMessage.Text = "The applicant qualifies."
    Else
        lblMessage.Text = "The applicant does not qualify."
    End If
End If
```

Figure 4-13 shows a flowchart for these nested `If` statements.

**Logical Operators**

**CONCEPT:** Logical operators combine two or more relational expressions into a single expression.

Logical operators can combine multiple Boolean expressions into a compound expression. Each individual Boolean expression might be very simple. But then, you can combine them using logical operators (also called Boolean operators) to make complex decisions. Table 4-3 lists Visual Basic’s logical operators.
The **And Operator**

The *And* operator combines two expressions into one. Both expressions must be true for the overall expression to be true. The following *If* statement uses the *And* operator:

```vbnet
If intTemperature < 20 And intMinutes > 12 Then
    lblMessage.Text = "The temperature is in the danger zone."
End If
```

<table>
<thead>
<tr>
<th>Operator</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>And</em></td>
<td>Combines two expressions into one. Both expressions must be true for the overall expression to be true.</td>
</tr>
<tr>
<td><em>Or</em></td>
<td>Combines two expressions into one. One or both expressions must be true for the overall expression to be true. It is only necessary for one to be true, and it does not matter which.</td>
</tr>
<tr>
<td><em>Xor</em></td>
<td>Combines two expressions into one. One expression (not both) must be true for the overall expression to be true. If both expressions are true, or both expressions are false, the overall expression is false.</td>
</tr>
<tr>
<td><em>Not</em></td>
<td>Reverses the logical value of an expression: makes a true expression false and a false expression true.</td>
</tr>
</tbody>
</table>

In this statement, the two relational expressions are combined into a single expression. The assignment statement is only executed if `intTemperature` is less than 20 and `intMinutes` is greater than 12. If either relational expression is false, the entire expression is false and the assignment statement is not executed.

Table 4-4 shows a truth table for the *And* operator. The truth table lists all the possible combinations of values that two expressions may have, followed by the resulting value returned by the *And* operator connecting the two conditions. As the table shows, both Expression 1 and Expression 2 must be true for the *And* operator to return value of *True*.

<table>
<thead>
<tr>
<th>Expression 1</th>
<th>Expression 2</th>
<th>Expression 1 <em>And</em> Expression 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

**TIP:** You must provide complete expressions on each side of the *And* operator. For example, the following is not correct because the condition on the right side of the *And* operator is not a complete expression:

```vbnet
intTemperature > 0 And < 100
```

The expression must be rewritten as follows:

```vbnet
intTemperature > 0 And intTemperature < 100
```
Short Circuit-Evaluation with AndAlso

When the And operator appears in a compound Boolean expression, Visual Basic evaluates both subexpressions. Consider the following example in which the first expression compares \( \text{dblX} \) to zero and the second expression calls a Boolean function named CheckValue:

\[
\text{If } \text{dblX} > 0 \text{ AndAlso CheckValue(dblX) Then}
\]

\[
\text{lblResult.Text }= \text{"Expression is True"}
\]

\[
\text{Else}
\]

\[
\text{lblResult.Text }= \text{"Expression is False"}
\]

\[
\text{End If}
\]

When using the And operator, both subexpressions (before and after AndAlso) must be true for the compound expression to be true. Comparing a value to zero is fast and easy, but executing a function call can be time-consuming, depending on what's inside the function. In our example, Visual Basic calls the CheckValue function, regardless of the value of \( \text{dblX} \). What if we could compare \( \text{dblX} \) to zero, and return \text{False} immediately if it was less than or equal to zero? Then there would be no need to call the CheckValue function. Such behavior is called short-circuit evaluation. In Visual Basic you use the AndAlso operator to achieve short-circuit evaluation. In the following example, assuming that \( \text{dblX} \) is less than or equal to zero, CheckValue is not called and Expression is False is displayed:

\[
\text{If } \text{dblX} > 0 \text{ AndAlso CheckValue(dblX) Then}
\]

\[
\text{lblResult.Text }= \text{"Expression is True"}
\]

\[
\text{Else}
\]

\[
\text{lblResult.Text }= \text{"Expression is False"}
\]

\[
\text{End If}
\]

See the ShortCircuit application in the student sample programs folder named Chap4\ShortCircuit for an example of the AndAlso operator.

The Or Operator

The Or operator combines two expressions into one. One or both expressions must be true for the overall expression to be true. It is only necessary for one to be true, and it does not matter which. The following If statement uses the Or operator:

\[
\text{If } \text{intTemperature} < 20 \text{ Or intTemperature } > 100 \text{ Then}
\]

\[
\text{lblMessage.Text }= \text{"The temperature is in the danger zone."}
\]

\[
\text{End If}
\]

The assignment statement will be executed if \( \text{intTemperature} \) is less than 20 or \( \text{intTemperature} \) is greater than 100. If either relational test is true, the entire expression is true and the assignment statement is executed.

Table 4-5 is a truth table for the Or operator.

<table>
<thead>
<tr>
<th>Expression 1</th>
<th>Expression 2</th>
<th>Expression 1 Or Expression 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

All it takes for an Or expression to be true is for one of the subexpressions to be true. It doesn't matter if the other subexpression is true or false.
4.6 Logical Operators

**Short Circuit-Evaluation with OrElse**

When the `Or` operator appears in a compound Boolean expression, Visual Basic evaluates both expressions on the left and right side of the operator. Consider the following example, in which the first expression compares `dblX` to zero; the second calls a Boolean function named `CheckValue`:

```vbnet
If dblX = 0 Or CheckValue(dblX) Then
    lblResult.Text = "Expression is True"
End If
```

When this code executes, the expression `dblX = 0` will be tested, and then the `CheckValue` function is called. In some situations, however, it shouldn’t be necessary to call the `CheckValue` function to determine the value of the compound expression. If the expression `dblX = 0` is true, then we know that the compound expression is true, so the function call can be skipped. As previously mentioned, this type of evaluation is known as short-circuit evaluation, and it can be performed with the `OrElse` operator.

In the following example, if `dblX` equals zero, the `CheckValue` function is not called:

```vbnet
If dblX = 0 OrElse CheckValue(dblX) Then
    lblResult.Text = "Expression is True"
End If
```

See the `ShortCircuit` application in the student sample programs folder named `Chap4/ShortCircuit` for an example of the `OrElse` operator.

**The Xor Operator**

`Xor` stands for exclusive `Or`. The `Xor` operator takes two expressions as operands and creates an expression that is true when one, but not both, of the subexpressions is true. The following `If` statement uses the `Xor` operator:

```vbnet
If decTotal > 1000 Xor decAverage > 120 Then
    lblMessage.Text = "You may try again."
End If
```

The assignment statement will be executed if `decTotal` is greater than 1000 or `decAverage` is greater than 120, but not both. If both relational tests are true, or neither is true, the entire expression is false. Table 4-6 shows a truth table for the `Xor` operator.

<table>
<thead>
<tr>
<th>Expression 1</th>
<th>Expression 2</th>
<th>Expression 1 Or Expression 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
TIP: You must provide complete expressions on both sides of the Xor operator. For example, the following is not correct because the condition on the right side of the Xor operator is not a complete expression:

\[
\text{value} < 0 \text{ Xor } > 100
\]

The expression must be rewritten as follows:

\[
\text{value} < 0 \text{ Xor value} > 100
\]

**The Not Operator**

The **Not operator** takes a Boolean expression and reverses its logical value. In other words, if the expression is true, the Not operator returns *False*, and if the expression is false, it returns *True*. The following If statement uses the Not operator:

```vbnet
If Not intTemperature > 100 Then
    lblMessage.Text = "You are below the maximum temperature."
End If
```

First, the expression intTemperature > 100 is tested to be true or false. Then the Not operator is applied to that value. If the expression intTemperature > 100 is true, the Not operator returns False. If it is false, the Not operator returns True. This example is equivalent to asking *Is intTemperature not greater than 100?* Table 4-7 shows a truth table for the Not operator.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Not Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

**Checking Numeric Ranges with Logical Operators**

When your program is determining whether a number is inside a numeric range, it's best to use the And operator. For example, the following If statement checks the value in intX to determine whether it is in the range of 20 through 40:

```vbnet
If intX >= 20 And intX <= 40 Then
    lblMessage.Text = "The value is in the acceptable range."
End If
```

The expression in the If statement is true only when intX is greater than or equal to 20 and less than or equal to 40. The value in intX must be within the range of 20 through 40 for this expression to be true.

When your program is determining whether a number is outside a range, it's best to use the Or operator. The following statement determines whether intX is outside the range of 20 through 40:

```vbnet
If intX < 20 Or intX > 40 Then
    lblMessage.Text = "The value is outside the acceptable range."
End If
```

It is important not to get these logical operators confused. For example, the following expression cannot be true because no value exists that is both less than 20 and greater than 40.
4.6 Logical Operators

If intX < 20 And intX > 40 Then
    lblMessage.Text = "The value is outside the acceptable range."
End If

If You Want to Know More about Using Not, And, Or, and Xor Together

It is possible to write an expression containing more than one logical operator. For example, examine the following If statement:

```
If intX < 0 And intY > 100 Or intZ = 50 Then
    ' Perform some statement.
End If
```

Logical operators have an order of precedence. The Not operator has the highest precedence, followed by the And operator, followed by the Or operator, followed by the Xor operator. So, in the example statement, the following expression is evaluated first:

```
intX < 0 And intY > 100
```

The result of this expression is then applied to the Or operator to carry out the rest of the condition. For example, if the first expression (using And) is true, the remainder of the condition will be tested as follows:

```
True Or intZ = 50
```

If the first expression (using And) is false, however, the remainder of the condition will be tested as follows:

```
False Or intZ = 50
```

Always use parentheses in logical expressions to clarify the order of evaluation. The following If statement confirms that the And operator executes before the Or operator:

```
If (intX < 0 And intY > 100) Or intZ = 50 Then
    ' Perform some statement.
End If
```

You can use parentheses to force one expression to be tested before others. For example, look at the following If statement:

```
If intX < 0 And (intY > 100 Or intZ = 50) Then
    ' Perform some statement.
End If
```

In the statement, the expression \((\text{intY} > 100 \text{ Or intZ} = 50)\) is tested first.

If You Want to Know More about Using Math Operators with Relational and Logical Operators

It is possible to write expressions containing math, relational, and logical operators. For example, look at the following code segment:

```
intA = 5
intB = 7
intX = 100
intY = 30
If (intX > (intA * 10)) And (intY < (intB + 20)) Then
    ' Perform some statement.
End If
```

In statements containing complex conditions, math operators execute first. After the math operators, relational operators execute. Logical operators execute last. Let’s use this order to step through the evaluation of the condition shown in our sample If statement.
First, the math operators execute, causing the statement to become

\[
\text{If (intX > 50) And (intY < 27) Then}
\]

Next, the relational operators execute, causing the statement to become

\[
\text{If True And False Then}
\]

Since True And False equals False, the condition is false.

**Checkpoint**

4.8 The following truth table shows various combinations of the values True and False connected by a logical operator. Complete the table by indicating whether the result of each combination is True or False.

<table>
<thead>
<tr>
<th>Logical Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>True And False</td>
<td></td>
</tr>
<tr>
<td>True And True</td>
<td></td>
</tr>
<tr>
<td>False And True</td>
<td></td>
</tr>
<tr>
<td>False And False</td>
<td></td>
</tr>
<tr>
<td>True Or False</td>
<td></td>
</tr>
<tr>
<td>True Or True</td>
<td></td>
</tr>
<tr>
<td>False Or True</td>
<td></td>
</tr>
<tr>
<td>False Or False</td>
<td></td>
</tr>
<tr>
<td>True Xor False</td>
<td></td>
</tr>
<tr>
<td>True Xor True</td>
<td></td>
</tr>
<tr>
<td>Not True</td>
<td></td>
</tr>
<tr>
<td>Not False</td>
<td></td>
</tr>
</tbody>
</table>

**Comparing, Testing, and Working with Strings**

**CONCEPT:** Visual Basic provides various methods in the `String` class that make it easy to work with strings. This section shows you how to use relational operators to compare strings, and discusses several intrinsic functions and string methods that perform tests and manipulations on strings.

In the preceding examples, you saw how numbers can be compared using the relational operators. You can also use relational operators to compare strings. For example, look at the following code segment, in which `strName1` and `strName2` are string variables.

```vba
strName1 = "Mary"
strName2 = "Mark"
If strName1 = strName2 Then
    lblMessage.Text = "The names are the same"
Else
    lblMessage.Text = "The names are NOT the same"
End If
```

The `=` operator tests `strName1` and `strName2` to determine whether they are equal. Since the strings “Mary” and “Mark” are not equal, the `Else` part of the `If` statement will cause the message The names are NOT the same to be copied to `lblMessage.Text`.

You can compare string variables with string literals as well. The following code sample uses the <> operator to determine if `strMonth` is not equal to October:
4.7 Comparing, Testing, and Working with Strings

If strMonth <> "October" Then
    ' statement
End If

You can also use the >, <, >=, and <= operators to compare strings. Before we look at these operators, though, we must understand how characters are stored in memory.

Computers do not actually store characters, such as A, B, C, and so on, in memory. Instead, they store numeric codes that represent the characters. Visual Basic uses Unicode, which is a numbering system that represents all letters of the alphabet (lowercase and uppercase), the printable digits 0 through 9, punctuation symbols, and special characters. Each character is stored in memory as its corresponding Unicode number. When the computer is instructed to print the value on the screen, it displays the character that corresponds to the numeric code.

NOTE: Unicode is an international encoding system that is extensive enough to represent all the characters of all the world's alphabets.

In Unicode, letters are arranged alphabetically. Because A comes before B, the numeric code for the letter A is less than the code for the letter B. In the following If statement, the relational expression "A" < "B" is true.

    If "A" < "B" Then
        ' Do something
    End If

TIP: When comparing strings, make sure they are consistent in their use of uppercase and lowercase letters. Avoid comparing "jones" to "Adams" or "BAKER", for example. The ordering of strings is affected by the choice of uppercase and lowercase letters.

When you use relational operators to compare strings, the strings are compared character-by-character. For example, look at the following code segment:

    strName1 = "Mary"
    strName2 = "Mark"
    If strName1 > strName2 Then
        lblMessage.Text = "Mary is greater than Mark"
    Else
        lblMessage.Text = "Mary is not greater than Mark"
    End If

The > operator compares each character in the strings "Mary" and "Mark", beginning with the first, or leftmost characters, as shown in Figure 4-14.

**Figure 4-14** String comparison

<table>
<thead>
<tr>
<th>M</th>
<th>a</th>
<th>r</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>a</td>
<td>r</td>
<td>k</td>
</tr>
</tbody>
</table>

Here is how the comparison takes place:

1. The M in Mary is compared with the M in Mark. Since these are the same, the next characters are compared.
2. The \textit{a} in \textit{Mary} is compared with the \textit{a} in \textit{Mark}. Since these are the same, the next characters are compared.

3. The \textit{r} in \textit{Mary} is compared with the \textit{r} in \textit{Mark}. Since these are the same, the next characters are compared.

4. The \textit{y} in \textit{Mary} is compared with the \textit{k} in \textit{Mark}. Since these are not the same, the two strings are not equal. The character \textit{y} is greater than \textit{k}, so it is determined that \textit{Mary} is greater than \textit{Mark}.

\textbf{NOTE:} If one of the strings in a relational comparison is shorter in length than the other, Visual Basic treats the shorter character as if it were padded with blank spaces. For example, suppose the strings "\textit{Hi}gh" and "\textit{Hi}" were compared. The string "\textit{Hi}" would be treated as if it were four characters in length, with the last two characters being spaces. Because the space character has a lower value than all other characters in Unicode, "\textit{Hi}" would be less than "\textit{Hi}gh".

\section*{Testing for No Input}

You can determine whether the user has entered a value into a text box by comparing the TextBox control’s Text property to the predefined constant \texttt{String.Empty} as shown here:

\begin{verbatim}
If txtInput.Text = String.Empty Then
    lblMessage.Text = “Please enter a value”
Else
    ‘ The txtInput control contains input, so
    ‘ perform an operation with it here.
End If
\end{verbatim}

The predefined constant \texttt{String.Empty} represents an \textit{empty string}, which is a string that contains no characters.

The \texttt{If} statement copies the string \textit{Please enter a value} to \texttt{lblMessage} if the \texttt{txtInput} control contains no input. The statements following \texttt{Else} are only executed if \texttt{txtInput} contains a value. You can use this technique to determine whether the user has provided input for a required field before performing operations on that field.

\textbf{NOTE:} The technique we used in the preceding \texttt{If} statement does not detect a string that contains only spaces. A space is a character, just as the letter \textit{A} is a character. If the user types only spaces into a text box, you will have to trim away the spaces to determine if any other characters were typed. Later in this section, we will discuss functions for trimming spaces from strings.

\section*{The \texttt{ToUpper} and \texttt{ToLower} Methods}

The \texttt{ToUpper} and \texttt{ToLower} methods are both members of the String class, so they may be called with any string variable or expression. The \texttt{ToUpper} method returns the uppercase equivalent of a string. Here is the method’s general format:

\begin{verbatim}
StringExpression.ToUpper()
\end{verbatim}

\texttt{StringExpression} can be any string variable or string expression. In the following example, \texttt{strLittleWord} and \texttt{strBigWord} are string variables:

\begin{verbatim}
strLittleWord = “Hello”
strBigWord = strLittleWord.ToUpper()
\end{verbatim}
After the statement executes, `bigWord` will contain “HELLO” in uppercase letters. Notice that the original string, “hello” had one uppercase letter—the initial H. The `ToUpper` method only converts lowercase characters. Characters that are already uppercase and characters that are not alphabet letters are not converted.

**TIP:** The `ToUpper` method does not modify the value of the string, but returns the string's uppercase equivalent. For example, after the statements in the previous example execute, `strLittleWord` still contains the original string "Hello".

The `ToLower` method works just like the `ToUpper` method, except it returns a lowercase version of a string. Here is the method's general format:

```
StringExpression.ToLower()
```

In the following example, `strBigTown` and `strLittleTown` are string variables:

```csharp
strBigTown = "NEW YORK"
strLittleTown = bigTown.ToLower()
```

After the statements execute, the variable `strLittleTown` contains the string "new york". The `ToLower` method only converts uppercase characters. Characters that are already lowercase, and characters that are not alphabet letters, are not converted.

**TIP:** Like `ToUpper`, the `ToLower` method does not modify the original string.

You may also use the `ToUpper` and `ToLower` methods with a control's `Text` property. In the following example `strLastName` is a string variable:

```csharp
strLastName = txtLastName.Text.ToUpper()
```

The `ToUpper` and `ToLower` methods are helpful in performing string comparisons. String comparisons in Visual Basic are case sensitive, meaning that uppercase letters are not considered the same as their lowercase counterparts. In other words, A is not the same as a. This can lead to problems when you construct `If` statements that compare strings. Tutorial 4-6 leads you through such an example.

**Tutorial 4-6:**

Examining an application that performs string comparisons

**Step 1:** Open the Secret Word 1 project from the student sample programs folder named Chap4\Secret Word 1.

**Step 2:** Run the application. The form shown in Figure 4-15 appears. This application asks you to enter the secret word, which might be similar to a password in some programs. The secret word is PROSPERO.

**Step 3:** Enter `prospero` in all lowercase letters, and click the Ok button. You will see the message Wrong! That is NOT the secret word!

**Step 4:** Enter `Prospero` with an uppercase P, followed by all lowercase letters. Click the Ok button. Once again, you see the message Wrong! That is NOT the secret word!
Chapter 4  Making Decisions and Working with Strings

Figure 4-15  Secret Word form

Step 5: Enter PROSPERO in all uppercase letters and click the Ok button. This time you see the message Congratulations! That is the secret word!

Step 6: Click the Exit button to close the application.

Step 7: Open the Code window and find the btnOk_Click event procedure. The code is as follows:

```vbnet
Private Sub btnOk_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOk.Click
    ' Compare the word entered with the secret word.
    If txtInput.Text = "PROSPERO" Then
        lblMessage.Text = "Congratulations! That is the secret word!"
    Else
        lblMessage.Text = "Wrong! That is NOT the secret word!"
    End If
End Sub
```

The If...Then...Else statement compares the string entered by the user to PROSPERO in all uppercase letters. But what if the programmer intended to accept the word without regard to case? What if prospero in all lowercase letters is valid as well? One solution would be to modify the If statement to test for all the other possible values. However, to test for all the possible combination of lowercase and uppercase letters would require a large amount of code.

A better approach is to convert the text entered by the user to uppercase letters, and then compare the converted text to PROSPERO. When the user enters the word prospero in any combination of uppercase or lowercase characters, this test will return True. Modify the code by adding a call to the ToUpper method, as shown bold in the following code.

```vbnet
If txtInput.Text.ToUpper() = "PROSPERO" Then
    lblMessage.Text = "Congratulations! That is the secret word!"
Else
    lblMessage.Text = "Wrong! That is NOT the secret word!"
End If
```

Step 8: Run the application. When the form appears, Enter prospero in all lowercase letters and click the Ok button. This time you see the message Congratulations! That is the secret word! You can experiment with various combinations of uppercase and lowercase letters. As long as you type the word prospero the application will recognize it as the secret word.

Step 9: Close the project.
The `ToLower` method can also be used in Tutorial 4-6 to accomplish the same result, as shown in bold in the following code. Just make sure you compare the return value of the `ToLower` method to an all lowercase string.

```vbnet
If txtInput.Text.ToLower() = "prospero" Then
    lblMessage.Text = "Congratulations! That is the secret word!"
Else
    lblMessage.Text = "Wrong! That is NOT the secret word!"
End If
```

### The IsNumeric Function

The intrinsic `IsNumeric` function accepts a string as its argument and returns `True` if the string contains a number. The function returns `False` if the string's contents cannot be recognized as a number. Here is the function's general use:

```vbnet
IsNumeric(StringExpression)
```

Here is an example:

```vbnet
Dim strNumber As String
strNumber = "576"
If IsNumeric(strNumber) Then
    lblMessage.Text = "It is a number"
Else
    lblMessage.Text = "It is NOT a number"
End If
```

In this statement, the expression `IsNumeric(strNumber)` returns `True` because the contents of `strNumber` can be recognized as a number. In the following code segment, however, the expression returns `False`:

```vbnet
strNumber = "123abc"
If IsNumeric(strNumber) Then
    lblMessage.Text = "It is a number"
Else
    lblMessage.Text = "It is NOT a number"
End If
```

When you want the user to enter numeric data, the `IsNumeric` function is useful for checking user input and confirming that it is valid.

### Determining the Length of a String

The `Length` property, a member of the `String` class, returns the number of characters in a string. Here is an example:

```vbnet
Dim strName As String = "Herman"
Dim intNumChars As Integer
intNumChars = strName.Length
```

The code stores 6 in `intNumChars` because the length of the string "Herman" is 6.

You can also determine the length of a control's Text property, as shown in the following code:

```vbnet
If txtInput.Text.Length > 20 Then
    lblMessage.Text = "Please enter no more than 20 characters."
End If
```
There are many situations in which `Length` is useful. One example is when you must display or print a string and have only a limited amount of space.

**WARNING:** If you attempt to get the length of an uninitialized string variable, a runtime error occurs. You can prevent this error by initializing string variables with an empty string, as shown in the following statement:

```vbnet
Dim str As String = String.Empty
```

### Optional Topic: Trimming Spaces from Strings

Sometimes it is necessary to trim leading and/or trailing spaces from a string before performing other operations on the string, such as a comparison. A **leading space** is a space that appears at the beginning, or left side, of a string. For instance, the following string has three leading spaces:

```
   Hello
```

A **trailing space** is a space that appears at the end, or right side, of a string, after the non-space characters. The following string has three trailing spaces:

```
Hello   
```

The `String` class has three methods for removing spaces: `TrimStart`, `TrimEnd`, and `Trim`. Here is the general format of each method:

- `StringExpression.TrimStart()`
- `StringExpression.TrimEnd()`
- `StringExpression.Trim()`

The **TrimStart method** returns a copy of the string expression with all leading spaces removed. The **TrimEnd method** returns a copy of the string expression with all trailing spaces removed. The **Trim method** returns a copy of the string expression with all leading and trailing spaces removed. The following is an example:

```vbnet
strGreeting = "   Hello   
lblMessage1.Text = strGreeting.TrimStart() 
lblMessage2.Text = strGreeting.TrimEnd() 
lblMessage3.Text = strGreeting.Trim() 
```

In this code, the first statement assigns the string "   Hello   " (with three leading spaces and three trailing spaces) to the named variable, `strGreeting`. In the second statement, the `TrimStart` method is called. Its return value, "Hello   ", is assigned to `lblMessage1.Text`. In the third statement, the `TrimEnd` method is called. Its return value, "   Hello", is assigned to `lblMessage2.Text`. In the fourth statement, the `Trim` method is called. Its return value, "Hello", is assigned to `lblMessage3.Text`.

These methods do not modify the string variable, but return a modified copy of the variable. To actually modify the string variable you must use a statement such as the following:

```vbnet
strGreeting = strGreeting.Trim()
```

After this statement executes, the greeting variable no longer contains leading or trailing spaces.

Like the `Length` property, these methods may also be used with a control’s `Text` property. The following is an example:
Dim strName As String
strName = txtName.Text.Trim()

**The Substring Method**

The **Substring method** returns a substring, or a string within a string. There are two formats:

\[
\text{StringExpression.Substring(Start)} \\
\text{StringExpression.Substring(Start, Length)}
\]

The positions of the characters in `StringExpression` are numbered, with the first character at position 0. In the first format shown for the method, an integer argument, `Start`, indicates the starting position of the string to be extracted from `StringExpression`. The method returns a string containing all characters from the `Start` position to the end of `StringExpression`. For example, look at the following code:

Dim strLastName As String
Dim strFullName As String = "George Washington"
strLastName = strFullName.Substring(7)

After this code executes, the variable `strLastName` will contain the string "Washington" because "Washington" begins at position 7 in `strFullName`, and continues to the end of the string.

In the second format shown for `Substring`, a second integer argument, `Length`, indicates the number of characters to extract, including the starting character. For example, look at the following code:

Dim strFirstName As String
Dim strFullName As String = "George Washington"
strFirstName = strFullName.Substring(0, 6)

In this code, the `Substring` method returns the six characters that begin at position 0 in `strFullName`. After the code executes, the variable `strFirstName` contains the string "George".

**Optional Topic: The IndexOf Method**

The **IndexOf method** searches for a character or a string within a string. The method has three general formats:

\[
\text{StringExpression.IndexOf(SearchString)} \\
\text{StringExpression.IndexOf(SearchString, Start)} \\
\text{StringExpression.IndexOf(SearchString, Start, Count)}
\]

In the first format, `SearchString` is the string or character to search for within `StringExpression`. The method returns the character position, or index, of the first occurrence of `SearchString` if it is found within `StringExpression`. If `SearchString` is not found, the method returns –1. For example, look at the following code:

Dim strName As String = "Angelina Adams"
Dim intPosition As Integer
intPosition = strName.IndexOf("e")

After this code executes, the variable `intPosition` equals 3 because the character 'e' is found at character position 3.
In the second format shown for `IndexOf`, a second argument, `Start`, is an integer that specifies a starting position within `StringExpression` for the search to begin. The following is an example:

```vba
Dim strName As String = "Angelina Adams"
Dim intPosition As Integer
intPosition = strName.IndexOf("A", 1)
```

After the code executes, the variable `intPosition` equals 9. The `IndexOf` method begins its search at character position 1 (the second character), so the first `A` is skipped.

**NOTE:** The version of the `IndexOf` method used here performs a case sensitive search. When searching for `A` it does not return the position of `a`.

In the third format shown for `IndexOf`, a third argument, `Count`, is an integer specifying the number of characters within `StringExpression` to search. Here is an example:

```vba
Dim strName As String = "Angelina Adams"
Dim intPosition As Integer
intPosition = strName.IndexOf("A", 1, 7)
```

After the code executes, the variable position equals -1. The `IndexOf` method searches only 7 characters, beginning at character 1. Because `A` is not found in the characters searched, the method returns -1.

**WARNING:** A runtime error will occur if the starting position argument passed to `IndexOf` is negative or specifies a nonexistent position. Get the length of the string before calling `IndexOf`, to ensure the index is in a valid range.

The following code shows how to use the `IndexOf` method to determine if a search string exists within a string:

```vba
Dim strName As String = "Angelina Adams"
If strName.IndexOf("Adams") = -1 Then
    lblMessage.Text = "Adams is not found"
End If
```

Tutorial 4-7 completes a string searching application.

**Tutorial 4-7:**
Completing a string searching application

In this tutorial, you will write code that implements a string searching program. You will have an opportunity to try the `IsNumeric`, `Trim`, and `IndexOf` methods. The user interface is already created, so you can concentrate on the program code that makes it work. Here are its basic features:

- A string is shown at the top of the form, as shown in Figure 4-16, containing various character patterns (abc, ABC, 00123, and so on). It uses a blue font, which appears gray on the printed page.
- The user inputs a string into the text box, indicating which substring they want to find.
• The user clicks the Go button, as shown in Figure 4-17.
• The program displays the position in which the substring was found. You can verify the accuracy of the result by inspecting the numbers in the scale line below the string shown in blue.
• The user can change the starting index position of the search from 0 to another value. In Figure 4-18, the user has selected index position 4 to begin searching. The next matching occurrence of ABC is found at index 20.
• If the user enters a nonnumeric index, an error message box pops up, as shown in Figure 4-19.

**NOTE:** We will be using the *Return* statement in this program, which quits (leaves) a procedure immediately. It is often used when error checking statements determine that the remaining part of the procedure should not execute. The *Exit Sub* statement can be used to carry out the same action as *Return*.

**Figure 4-16** String Finder application, when started

**Figure 4-17** User enters substring they want to find, clicks Go button

**Figure 4-18** Searching for ABC starting at index position 4

**Figure 4-19** User has entered a nonnumeric index

---

**Step 1:** Open the *String Finder* project from the student sample programs folder named *Chap4String Finder*.

**Step 2:** Open Form1 in Design mode. Click each control and view its name in the Properties window. When you begin writing code, you will want to know the control names.

**Step 3:** Double-click the Go button to open the Code window.
Step 4: You will be adding code to the btnGo.Click procedure, which handles the Click event for the Go button. Begin by adding a constant that will show the results of searching (shown in bold):

```vbnet
Private Sub btnGo_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnGo.Click

' Value returned by IndexOf when a search fails.
Const intNOT_FOUND As Integer = -1
```

Step 5: Next, add code to the same procedure that checks the starting index value entered by the user into the txtStartIndex field. If it is nonnumeric, we display an error message and exit the procedure. If it is a valid number, we assign it to the intStart variable:

```vbnet
' Make sure the starting index is numeric.
' Then get the starting index of the search.
Dim intStart As Integer
If Not IsNumeric(txtStartIndex.Text) Then
    MessageBox.Show("The starting index must be numeric", "Error")
    Return
Else
    intStart = CInt(txtStartIndex.Text)
End If
```

Step 6: Insert the following code that checks for an empty search string. If it is blank, the procedure exits.

```vbnet
' Before searching, check for a possible empty string.
' Exit the subroutine if the string is empty.
If txtToFind.Text.Length = 0 Then
    lblResults.Text = "There is no string to find!"
    Return
End If
```

Step 7: Insert the following code that searches for the user’s substring by calling the IndexOf method. It shows the results in lblResults.Text:

```vbnet
' Perform the search, returning the index of the first matching string.
Dim intPos As Integer = lblString.Text.IndexOf(txtToFind.Text, intStart)

' Output a message based on the search results. If the string was found, display its index position.
If intPos = intNOT_FOUND Then
    lblResults.Text = "The string was not found"
Else
    lblResults.Text = "The string was found at index " & intPos
End If
```

Step 8: Save and run the program. Search for the substring ABC starting at index 0. The program should find the string at position 3.

Step 9: Search for ABC starting at index 4. The program should find the string at position 20.
4.8 Focus on GUI Design: The Message Box

CONCEPT: Sometimes you need a convenient way to display a message to the user. This section discusses the MessageBox.Show method, which allows you to display a message in a dialog box.

A message box is a pop-up window that displays a message to the user. In Chapter 3 we briefly introduced the MessageBox.Show method, which displays a message box. In this section we will discuss the MessageBox.Show method in greater detail and you will learn more about its capabilities. We will discuss the following general formats of the method call:

- MessageBox.Show(Message)
- MessageBox.Show(Message, Caption)
- MessageBox.Show(Message, Caption, Buttons)
- MessageBox.Show(Message, Caption, Buttons, Icon)
- MessageBox.Show(Message, Caption, Buttons, Icon, DefaultButton)

When MessageBox.Show executes, a message box (a Windows dialog box) pops up. For example, the following statement causes the message box shown in Figure 4-20 to appear:

MessageBox.Show("Operation complete.")
Chapter 4  Making Decisions and Working with Strings

In the second format, Caption is a string to be displayed in the message box’s title bar. The following statement causes the message box shown in Figure 4-21 to appear:

```csharp
MessageBox.Show("Operation complete.", "Status")
```

![Figure 4-20 Message box](image)

In both formats, the message box has only an OK button. In the third format, Buttons is a value that specifies which buttons to display in the message box. Table 4-8 lists the available values for Buttons and describes each.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageBoxButtons.AbortRetryIgnore</td>
<td>Displays Abort, Retry, and Ignore buttons</td>
</tr>
<tr>
<td>MessageBoxButtons.OK</td>
<td>Displays only an OK button</td>
</tr>
<tr>
<td>MessageBoxButtons.OKCancel</td>
<td>Displays OK and Cancel buttons</td>
</tr>
<tr>
<td>MessageBoxButtons.RetryCancel</td>
<td>Displays Retry and Cancel buttons</td>
</tr>
<tr>
<td>MessageBoxButtons.YesNo</td>
<td>Displays Yes and No buttons</td>
</tr>
<tr>
<td>MessageBoxButtons.YesNoCancel</td>
<td>Displays Yes, No, and Cancel buttons</td>
</tr>
</tbody>
</table>

For example, the following statement causes the message box shown in Figure 4-22 to appear:

```csharp
MessageBox.Show("Do you wish to continue?", "Please Confirm", MessageBoxButtons.YesNo)
```

![Figure 4-22 Message box with caption and Yes and No buttons](image)

In some versions of MessageBox.Show, Icon is a value that specifies an icon to display in the message box. The available values for Icon are MessageBoxIcon.Asterisk, MessageBoxIcon.Error, MessageBoxIcon.Exclamation, MessageBoxIcon.Hand, MessageBoxIcon.Information, MessageBoxIcon.Question, MessageBoxIcon.Stop, and MessageBoxIcon.Warning. Figure 4-23 shows the icons matching each value. Note that some values display the same icon as others.
4.8 Focus on GUI Design: The Message Box

For example, the following statement causes the message box shown in Figure 4-24 to appear:

```csharp
MessageBox.Show("Do you wish to continue?", "Please Confirm", _
    MessageBoxButtons.YesNo, MessageBoxIcon.Question)
```

In one version of `MessageBox.Show`, the `DefaultButton` argument specifies which button to select as the default button. The default button is the button clicked when the user presses the `Enter` key. Table 4-9 lists the available values for this argument.

**Table 4-9 DefaultButton values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MessageBoxDefaultButton.Button1</code></td>
<td>Selects the leftmost button on the message box as the default button</td>
</tr>
<tr>
<td><code>MessageBoxDefaultButton.Button2</code></td>
<td>Selects the second button from the left edge of the message box as the default button</td>
</tr>
<tr>
<td><code>MessageBoxDefaultButton.Button3</code></td>
<td>Selects the third button from the left edge of the message box as the default button</td>
</tr>
</tbody>
</table>

For example, the following statement displays a message box and selects `Button2` (the No button) as the default button:

```csharp
MessageBox.Show("Do you wish to continue?", "Please Confirm", _
    MessageBoxButtons.YesNo, MessageBoxIcon.Question, _
    MessageBoxButtons.DefaultButton.Button2)
```
Determining Which Button the User Clicked

When the user clicks any button on a message box, the message box is dismissed. In code, the `MessageBox.Show` method returns an integer that indicates which button the user clicked. You can compare this value with the values listed in Table 4-10 to determine which button was clicked.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Windows.Forms.DialogResult.Abort</code></td>
<td>The user clicked the <em>Abort</em> button</td>
</tr>
<tr>
<td><code>Windows.Forms.DialogResult.Cancel</code></td>
<td>The user clicked the <em>Cancel</em> button</td>
</tr>
<tr>
<td><code>Windows.Forms.DialogResult.Ignore</code></td>
<td>The user clicked the <em>Ignore</em> button</td>
</tr>
<tr>
<td><code>Windows.Forms.DialogResult.No</code></td>
<td>The user clicked the <em>No</em> button</td>
</tr>
<tr>
<td><code>Windows.Forms.DialogResult.OK</code></td>
<td>The user clicked the <em>OK</em> button</td>
</tr>
<tr>
<td><code>Windows.Forms.DialogResult.Retry</code></td>
<td>The user clicked the <em>Retry</em> button</td>
</tr>
<tr>
<td><code>Windows.Forms.DialogResult.Yes</code></td>
<td>The user clicked the <em>Yes</em> button</td>
</tr>
</tbody>
</table>

The following code shows how an If statement can take actions based on which message box button the user clicked:

```vbs
Dim intResult As Integer
intResult = MessageBox.Show("Do you wish to continue?", _
    "Please Confirm", MessageBoxButtons.YesNo)
If intResult = Windows.Forms.DialogResult.Yes Then
    ' Perform an action here
ElseIf intResult = Windows.Forms.DialogResult.No Then
    ' Perform another action here
End If
```

Using ControlChars.CrLf to Display Multiple Lines

If you want to display multiple lines of information in a message box, use the constant `ControlChars.CrLf` (CrLf stands for *carriage return line feed*). Concatenate it with the string you wish to display, where you wish to begin a new line (as shown in this example):

```vbs
MessageBox.Show("This is line 1" & ControlChars.CrLf & _
    "This is line 2")
```

This statement causes the message box in Figure 4-25 to appear. When Visual Basic displays the string "This is line 1" & ControlChars.CrLf & "This is line 2", it interprets ControlChars.CrLf as a command to begin a new line of output.
4.9 The Select Case Statement

**CONCEPT:** In a Select Case statement, one of several possible actions is taken, depending on the value of an expression.

The If...Then...ElseIf statement allows your program to branch into one of several possible paths. It performs a series of tests and branches when one of these tests is true. The Select Case statement, which is a similar mechanism, tests the value of an expression only once, and then uses that value to determine which set of statements to branch to. Following is the general format of the Select Case statement. The items inside the brackets are optional.

---

**TIP:** In code, you can use ControlChars.CrLf to create multiple lines in label text too.
Chapter 4 Making Decisions and Working with Strings

Select Case TestExpression
    [Case ExpressionList
        [one or more statements]]
    [Case ExpressionList
        [one or more statements]]
    ' Case statements may be repeated
    ' as many times as necessary.
    [Case Else
        [one or more statements]]
End Select

The first line starts with Select Case and is followed by a test expression. The test expression may be any numeric or string expression that you wish to test.

Starting on the next line is a sequence of one or more Case statements. Each Case statement follows this general form:

    Case ExpressionList
        one or more statements

After the word Case is an expression list, so-called because it may hold one or more expressions. Beginning on the next line, one or more statements appear. These statements are executed if the value of the test expression matches any of the expressions in the Case statement’s expression list.

A Case Else comes after all the Case statements. This branch is selected if none of the Case expression lists match the test expression. The entire Select Case construct is terminated with an End Select statement.

WARNING: The Case Else section is optional. If you leave it out, however, your program will have nowhere to branch to if the test expression doesn’t match any of the expressions in the Case expression lists.

Here is an example of the Select Case statement:

    Select Case CInt(txtInput.Text)
        Case 1
            MessageBox.Show("Day 1 is Monday.")
        Case 2
            MessageBox.Show("Day 2 is Tuesday.")
        Case 3
            MessageBox.Show("Day 3 is Wednesday.")
        Case 4
            MessageBox.Show("Day 4 is Thursday.")
        Case 5
            MessageBox.Show("Day 5 is Friday.")
        Case 6
            MessageBox.Show("Day 6 is Saturday.")
        Case 7
            MessageBox.Show("Day 7 is Sunday.")
        Case Else
            MessageBox.Show("That value is invalid.")
    End Select

Let’s look at this example more closely. The test expression is CInt(txtInput.Text). The Case statements Case 1, Case 2, Case 3, Case 4, Case 5, Case 6, and Case 7 mark where the program is to branch to if the test expression is equal to the values 1, 2, 3, 4, 5, 6, or 7. The Case Else section is branched to if the test expression is not equal to any of these values.
Suppose the user has entered 3 into the txtInput text box, so the expression `CInt(txtInput.Text)` is equal to 3. Visual Basic compares this value with the first Case statement's expression list:

```
Select Case CInt(txtInput.Text)
  Case 1
    MessageBox.Show("Day 1 is Monday.")
```

The only value in the expression list is 1, and this is not equal to 3, so Visual Basic goes to the next Case:

```
Select Case CInt(txtInput.Text)
  Case 1
    MessageBox.Show("Day 1 is Monday.")
  Case 2
    MessageBox.Show("Day 2 is Tuesday.")
```

Once again, the value in the expression list does not equal 3, so Visual Basic goes to the next Case:

```
Select Case CInt(txtInput.Text)
  Case 1
    MessageBox.Show("Day 1 is Monday.")
  Case 2
    MessageBox.Show("Day 2 is Tuesday.")
  Case 3
    MessageBox.Show("Day 3 is Wednesday.")
```

This time, the value in the Case's expression list matches the value of the test expression, so the `MessageBox.Show` statement on the next line executes. (If there had been multiple statements appearing between the Case 3 and Case 4 statements, all would have executed.) After the `MessageBox.Show` statement executes, the program jumps to the statement immediately following the `End Select` statement.

### The Select Case Structure in Flowcharts and Pseudocode

The flowchart segment in Figure 4-27 shows the general form of a Case structure. The diamond represents the test expression, which is compared to a series of values. The path of execution follows the value matching the test expression. If none of the values matches a test expression, the default path is followed (Case Else).

As with the If statement, the pseudocode for the Select Case statement looks very similar to the actual programming statements. The following is an example:

```
Select Input
  Case 1
    Display Message "Day 1 is Monday."
  Case 2
    Display Message "Day 2 is Tuesday."
  Case 3
    Display Message "Day 3 is Wednesday."
  Case 4
    Display Message "Day 4 is Thursday."
  Case 5
    Display Message "Day 5 is Friday."
  Case 6
    Display Message "Day 6 is Saturday."
```
Case 7
   Display Message “Day 7 is Sunday.”
Case Else
   Display Message “That value is invalid.”
End Select

Figure 4-27 General form of a Case structure

More about the Expression List

The Case statement’s expression list can contain multiple expressions, separated by commas. For example, the first Case statement in the following code compares intNumber to 1, 3, 5, 7, and 9, and the second Case statement compares it to 2, 4, 6, 8, and 10. In the following code, assume that strStatus is a string variable:

```plaintext
Select Case intNumber
   Case 1, 3, 5, 7, 9
      strStatus = "Odd"
   Case 2, 4, 6, 8, 10
      strStatus = "Even"
   Case Else
      strStatus = "Out of Range"
End Select
```

You can use relational operators in the Case statement, as shown by the following example. The Is keyword represents the test expression in the relational comparison.

```plaintext
Select Case sngTemperature
   Case Is <= 75
```
Finally, you can determine whether the test expression falls within a range of values. This requires the To keyword, as shown in the following code.

```
Select Case intScore
    Case Is >= 90
        strGrade = "A"
    Case 80 To 89
        strGrade = "B"
    Case 70 To 79
        strGrade = "C"
    Case 60 To 69
        strGrade = "D"
    Case 0 To 59
        strGrade = "F"
    Case Else
        MessageBox.Show("Invalid Score")
End Select
```

The numbers used on each side of the To keyword are included in the range. So, the statement Case 80 To 89 matches the values 80, 89, or any number in between.

**TIP:** The To keyword only works properly when the smaller number appears on its left and the larger number appears on its right. You can write an expression such as 10 To 0, but it will not function properly at runtime.

Tutorial 4-8 examines a sales commission calculator application.

---

**Tutorial 4-8:**

**Examining Crazy Al’s Sales Commission Calculator application**

Crazy Al’s Computer Emporium is a retail seller of personal computers. The sales staff at Crazy Al’s works strictly on commission. At the end of the month, each salesperson’s commission is calculated according to Table 4-11.

For example, a salesperson with $16,000 in monthly sales earns a 12% commission ($1,920.00). A salesperson with $20,000 in monthly sales earns a 14% commission ($2,800.00).

**Table 4-11 Sales commission rates**

<table>
<thead>
<tr>
<th>Sales This Month</th>
<th>Commission Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less than $10,000</td>
<td>5%</td>
</tr>
<tr>
<td>$10,000 – $14,999</td>
<td>10%</td>
</tr>
<tr>
<td>$15,000 – $17,999</td>
<td>12%</td>
</tr>
<tr>
<td>$18,000 – $21,999</td>
<td>14%</td>
</tr>
<tr>
<td>$22,000 or more</td>
<td>16%</td>
</tr>
</tbody>
</table>
Because the staff is paid once per month, Crazy Al’s allows each employee to take up to $1,500 per month in advance pay. When sales commissions are calculated, the amount of each employee’s advance pay is subtracted from the commission. If any salesperson’s commission is less than the amount of the advance, he or she must reimburse Crazy Al’s for the difference.

Here are two examples:

- Beverly’s monthly sales were $21,400, so her commission is $2,996. She took $1,500 in advance pay. At the end of the month she gets a check for $1,496.
- John’s monthly sales were $12,600, so his commission is $1,260. He took $1,500 in advance pay. At the end of the month he must pay back $240 to Crazy Al’s.

In this tutorial, you examine the Crazy Al’s Commission Calculator application used to determine a salesperson’s commission.

**Step 1:** Open the Crazy Al project from the student sample programs folder named Chap4\Crazy Al.

**Step 2:** Run the application. The form shown in Figure 4-28 appears.

**Figure 4-28** Crazy Al’s Commission Calculator form

**Step 3:** Enter 16000 as the amount of sales for this month (first text box).

**Step 4:** Enter 1000 as the amount of advance pay taken (second text box). Click the Calculate button. You should see the commission rate, commission, and net pay information, as shown in Figure 4-29.

**Step 5:** Click the Clear button to reset the contents of the input and display fields.

**Figure 4-29** Calculations filled in
Experiment with other values for sales and advance pay.

**Step 6:** When you are finished, click the *Exit* button to end the application.

**Step 7:** Open the *Code* window and find the *btnCalculate_Click* event procedure. The code is as follows:

```vbnet
Private Sub btnCalculate_Click(ByVal eventSender As System.Object, ByVal eventArgs As System.EventArgs) Handles btnCalculate.Click
    Dim decSalesAmount As Decimal ' Monthly sales amount
    Dim decAdvancePayAmount As Decimal ' Advance pay taken
    Dim decCommissionRate As Decimal ' Commission rate
    Dim decCommissionAmount As Decimal ' Commission
    Dim decNetPay As Decimal ' Net pay

    If IsNumeric(txtSalesAmount.Text) = False Then
        lblErrorMessage.Text = "Sales amount must be numeric"
        lblErrorMessage.Visible = True
        Return
    End If

    If IsNumeric(txtAdvancePayAmount.Text) = False Then
        lblErrorMessage.Text = "Advance pay amount must be numeric"
        lblErrorMessage.Visible = True
        Return
    End If

    ' Past this point, the user inputs contain decimal values.
    ' Hide the error message label and do the calculations.
    lblErrorMessage.Visible = False
    decSalesAmount = CDec(txtSalesAmount.Text)
    decAdvancePayAmount = CDec(txtAdvancePayAmount.Text)

    ' Determine the commission rate. Constants are in Decimal format.
    Select Case decSalesAmount
        Case Is < 10000
            decCommissionRate = 0.05D
        Case Is 10000 To 14999.9999
            decCommissionRate = 0.1D
        Case 15000 To 17999.9999
            decCommissionRate = 0.12D
        Case 18000 To 21999.9999
            decCommissionRate = 0.14D
        Case Is >= 22000
            decCommissionRate = 0.15D
    End Select

    ' Calculate the commission and net pay amounts.
    decCommissionAmount = decSalesAmount * decCommissionRate
    decNetPay = decCommissionAmount - decAdvancePayAmount

    ' Display the rate, commission, and net pay.
    lblCommissionRate.Text = decCommissionRate.ToString("p")
    lblCommissionAmount.Text = decCommissionAmount.ToString("c")
    lblNetPay.Text = decNetPay.ToString("c")
End Sub
```
As you can see, the Select Case construct has a Case statement for each level of sales in the commission table.

Checkpoint

4.14 Convert the following If...Then...ElseIf statement into a Select Case statement.

```vbnet
If intQuantity >= 0 And intQuantity <= 9 Then
decDiscount = 0.1
ElseIf intQuantity >= 10 And intQuantity <= 19 Then
decDiscount = 0.2
ElseIf intQuantity >= 20 And intQuantity <= 29 Then
decDiscount = 0.3
ElseIf intQuantity >= 30 Then
decDiscount = 0.4
Else
    MessageBox.Show("Invalid Data")
End If
```

Introduction to Input Validation

**CONCEPT:** Input validation is the process of inspecting input values and determining whether they are valid.

The accuracy of a program's output is only as good as the accuracy of its input. Therefore, it is important that your applications perform input validation on the values entered by the user. **Input validation** is the process of inspecting input values and determining whether they are valid. Now that you know how to use the If statement for conditional processing, you have more validation techniques at your disposal.

**The TryParse Method**

Each of the numeric classes in Visual Basic contains a method named TryParse. This method attempts to convert an input value to a certain numeric or date type, and returns a Boolean value that tells you if the conversion worked. For example, the Integer.TryParse method tries to convert an input value to Integer. It has two input parameters:

```vbnet
Integer.TryParse(valueToConvert, targetValue As Integer) As Boolean
```

The following statements attempt to convert the contents of a TextBox named txtInput to an integer and assign the value to intResult:

```vbnet
Dim intResult As Integer

If Integer.TryParse(txtInput.Text, intResult) Then
    lblResult.Text = "Success!"
Else
    lblResult.Text = "Error: an integer was not found"
End If
```

If the TextBox contains a valid integer string such as "26", the variable named intResult is assigned the value 26. If the TextBox cannot be converted to an integer, an error message is displayed by the program.
Other classes contain `TryParse` methods, including `Short`, `Long`, `Single`, `Double`, `Decimal`, `Date`, and `Boolean`. Each has the same basic format. The following statements call `Date.TryParse` and display an error message if the conversion failed:

```vbnet
Dim dtmTemp As Date
If Not Date.TryParse("05/15/2009 8:15 PM", dtmTemp) Then
    lblResult.Text = "Not a valid date"
End If
```

### Checking Multiple Values

Many applications must validate multiple user input values. You can combine calls to the `TryParse` method by using the `AndAlso` operator. The following example checks the `txtWeight` and `txtDistance` TextBox controls before copying their contents to variables:

```vbnet
If Single.TryParse(txtWeight.Text, sngWeight) AndAlso Single.TryParse(txtDistance.Text, sngDistance) Then
    ' ok to use the sngWeight and sngDistance variables now
Else
    lblResult.Text = "Weight and distance must be integers"
End If
```

See the `TryParse Example` application in the Chapter 4 examples folder for a working demonstration of the `TryParse` method.

### Using If Statements to Check Ranges of Values

In addition to checking for valid conversions, we often check input values to make sure they fall within a consistent and reasonable range. For example, consider the *Crazy Al’s Commission Calculator* application. The `btnCalculate_Click` procedure gets the amount of a salesperson’s sales and advance pay taken (values entered by the user) and calculates the sales commission. The following code shows part of the procedure after it has been modified to validate the user’s input:

```vbnet
' Validate the input to prevent the user from entering negative values.
If (decSalesAmount < 0) Or (decAdvancePayAmount < 0) Then
    lblErrorMessage.Text = "Sales & advance pay must be positive"
    lblErrorMessage.Visible = True
Else
    ' Determine the commission rate.
    Select Case decSalesAmount
        Case Is < 10000
            decCommissionRate = 0.05D
        Case 10000 To 14999
            decCommissionRate = 0.1D
        Case 15000 To 17999
            decCommissionRate = 0.12D
        Case 18000 To 21999
            decCommissionRate = 0.14D
        Case Is >= 22000
            decCommissionRate = 0.15D
    End Select
    ' Calculate the commission and net pay amounts.
    decCommissionAmount = decSalesAmount * decCommissionRate
    decNetPay = decCommissionAmount - decAdvancePayAmount
```
Chapter 4 Making Decisions and Working with Strings

' Display the rate, commission, and net pay.
lblCommissionRate.Text = decCommissionRate.ToString("p")
lblCommissionAmount.Text = decCommissionAmount.ToString("c")
lblNetPay.Text = decNetPay.ToString("c")
End If

The If...Then...Else statement displays an error message if decSalesAmount or decAdvancePayAmount contain negative values. The commission calculations are only performed when these variables contain nonnegative numbers.

4.11 Focus on GUI Design: Radio Buttons and Check Boxes

CONCEPT: Radio buttons appear in groups of two or more, allowing the user to select one of several options. A check box allows the user to select an item by checking a box, or deselect the item by unchecking the box.

Radio Buttons

Radio buttons are useful when you want the user to select one choice from several possible choices. Figure 4-30 shows a group of radio buttons.

A radio button may be selected or deselected. Each radio button has a small circle that appears filled-in when the radio button is selected, and appears empty when the radio button is deselected.

Visual Basic provides the RadioButton control, which allows you to create radio buttons. Radio buttons are normally grouped in one of the following ways:

- All radio buttons inside a group box are members of the same group.
- All radio buttons on a form not inside a group box are members of the same group.

Figure 4-31 shows two forms. The form on the left has three radio buttons that belong to the same group. The form on the right has two groups of radio buttons.

At runtime, only one radio button in a group may be selected at a time, which makes them mutually exclusive. Clicking on a radio button selects it, and automatically deselected any other radio button in the same group.

NOTE: The name radio button refers to the old car radios that had push buttons for selecting stations. Only one button could be pushed in at a time. When you pushed a button, it automatically popped out the currently selected button.
Radio Button Properties

Radio buttons have a Text property, which holds the text that is displayed next to the radio button’s circle. For example, the radio buttons in the leftmost form in Figure 4-31 have their Text properties set to *Coffee*, *Tea*, and *Soft Drink*.

Radio buttons have a Boolean property named Checked. The Checked property is set to *True* when the radio button is selected and *False* when the radio button is deselected. Their default value is *False*.

Working with Radio Buttons in Code

The standard prefix for radio button control names is `rad`. You determine if a radio button is selected by testing its Checked property. The following code shows an example. Assume that `radChoice1`, `radChoice2`, and `radChoice3` are radio buttons in the same group:

```vba
If radChoice1.Checked = True Then
    MessageBox.Show("You selected Choice 1")
ElseIf radChoice2.Checked = True Then
    MessageBox.Show("You selected Choice 2")
ElseIf radChoice3.Checked = True Then
    MessageBox.Show("You selected Choice 3")
End If
```

Radio Buttons have a **CheckedChanged** event that is triggered when the user selects or deselects a radio button. If you double-click a radio button in the Design window, a code template for the CheckedChange event procedure is created in the Code window.

Assigning a TabIndex Value and an Access Key to a Radio Button

Radio button controls have a position in the form’s tab order, which may be changed with the TabIndex property. As with other controls, you can assign an access key to a radio button by placing an ampersand (&) in the Text property, just before the character you wish to serve as the access key. The character will appear underlined on the form. At runtime, when the user presses the `Alt`+access key combination, the focus shifts to the radio button, and the radio button is selected.

Selecting a Radio Button in Code

You can use code to select a radio button, using an assignment statement to set the desired radio button’s Checked property to *True*, for example:

```vba
radChoice1.Checked = True
```

**TIP:** If you set a radio button’s Checked property to *True* in Design mode (with the Properties window), it becomes the default radio button for that group. It is selected when the application starts up and it remains selected until the user or application code selects another radio button.

Check Boxes

A check box appears as a small box, labeled with a caption. An example is shown in Figure 4-32.
Visual Basic provides the **CheckBox control**, which allows you to create check boxes. Like radio buttons, check boxes may be selected or deselected at runtime. When a check box is selected, a small check mark appears inside the box. Unlike radio buttons, check boxes are not mutually exclusive. You may have one or more check boxes on a form or in a group box, and any number of them can be selected at any given time.

The standard prefix for a CheckBox control's name is `chk`. Like radio buttons, check boxes have a Checked property. When a check box is selected, or checked, its Checked property is set to `True`. When a check box is deselected, or unchecked, its Checked property is set to `False`. Here is a summary of other characteristics of the check box:

- A check box's caption is stored in the Text property.
- A check box's place in the tab order may be modified with the TabIndex property. When a check box has the focus, a thin dotted line appears around its text. You can check or uncheck it by pressing the `Spacebar`.
- You may assign an access key to a check box by placing an ampersand (`&`) in the Text property, just before the character that you wish to serve as the access key.
- You can use code to select or deselect a check box. Simply use an assignment statement to set the desired check box's Value property, for example:
  ```vba
  chkChoice4.Checked = True
  ```
- You may set a check box's Checked property at design time.
- Like radio buttons, check boxes have a `CheckedChanged` event that is triggered whenever the user changes the state of the check box. If you have written a `CheckedChanged` event procedure for the check box, it will execute whenever the user checks or unchecks the check box.

In Tutorial 4-9, you examine an application that demonstrates radio buttons and check boxes.

---

**Tutorial 4-9:**

**Completing an application with radio buttons and check boxes**

**Step 1:** Open the **Radio Button Check Box Demo** project from the student sample programs folder named `Chap4\Radio Button Check Box Demo`.

**Step 2:** Open `Form1` in **Design** mode, as shown in Figure 4-33.

---

**Figure 4-33** **Radio Button Check Box Demo form**
Step 3: Open the code window for Form1.

Step 4: Double-click the Ok button and insert the following code into the btnOk_Click event handler, which evaluates the user's radio button selection:

```vbs
Private Sub btnOk_Click(ByVal eventSender As System.Object, ByVal eventArgs As System.EventArgs) Handles btnOk.Click

    ' Declare a string variable to hold a message.
    Dim strMessage As String = String.Empty

    ' The following If...ElseIf statement tests the group of radio buttons and copies the first part of the message to strMessage.
    If radChoice1.Checked = True Then
        strMessage = "You selected Choice 1"
    ElseIf radChoice2.Checked = True Then
        strMessage = "You selected Choice 2"
    ElseIf radChoice3.Checked = True Then
        strMessage = "You selected Choice 3"
    End If

    Step 5: Continuing in the same method code area, insert the following code right after the code you inserted in Step 4:

    ' The following If...Then statements test the check boxes and concatenates another part of the message to strMessage.
    If chkChoice4.Checked Then
        strMessage &= " and Choice 4"
    End If
    If chkChoice5.Checked Then
        strMessage &= " and Choice 5"
    End If
    If chkChoice6.Checked Then
        strMessage &= " and Choice 6"
    End If

    ' Now display the message.
    MessageBox.Show(strMessage)
End Sub
```

Because the Checked property is Boolean, you can simplify statements that use it in expressions. The following expression, for example:

If chkChoice4.Checked Then

is equivalent to the longer form:

If chkChoice4.Checked = True Then

Step 6: Save the project and run the application.

Step 7: Click Choice 3, Choice 4, and Choice 6. You should see the message box shown in Figure 4-34. Experiment by clicking different radio buttons and check boxes. Note the results.
Chapter 4 Making Decisions and Working with Strings

Figure 4-34 Message box displayed when the user selects radio buttons and check boxes, and clicks the OK button

Checkpoint

4.15 In code, how do you determine whether a radio button has been selected?

4.16 If several radio buttons are placed on a form, not inside group boxes, how many of them may be selected at any given time?

4.17 In code, how do you determine whether a check box has been selected?

4.18 If several check boxes appear on a form, how many of them may be selected at any given time?

4.19 How can the user check or uncheck a check box that has the focus by using the keyboard?

4.12 Class-Level Variables

CONCEPT: Class-level variables are not local to any procedure. In a form file they are declared outside of any procedure and may be accessed by statements in any procedure in the same form.

A variable's scope is the area of a program in which the variable is visible. All variables you have created so far had local scope, meaning that each was declared and used inside a single method or procedure. Sometimes a variable needs to be created in such a way that it can be shared between two or more methods. We might declare the variable inside a class, for example, giving it class scope (or module scope).

A Visual Basic form is defined by a class, so variables declared within its code area, yet outside of any of its methods, have class scope. When we refer to a class-level variable, we mean a variable that has class scope. The following code example shows the difference between local scope and class scope (variable declarations shown in bold):

```
1: Public Class Form1
2:
3:   ' Class-level variable
4:   Dim decTotalSalary As Decimal
5:
6: Private Sub btnAddWeekly_Click(ByVal sender As System.Object, _
7:   ByVal e As System.EventArgs) Handles btnAddWeekly.Click
8:   
9:   'Local variable
10:  Dim decWeeklyPay As Decimal
11:
12:  decWeeklyPay = CDec(txtPay.Text)
13:  decTotalSalary += decWeeklyPay
```
4.12 Class-Level Variables

The variable \texttt{decTotalSalary} (line 4) is declared at the class level, so it is visible to all methods in the \texttt{Form1} class. We say that it has class scope. It is referenced from two different methods, on lines 13 and 19.

The variable \texttt{decWeeklyPay} (line 10) is declared inside the \texttt{btnAddWeekly\_Click} method, so it has local scope. It is only visible between lines 10 and 14. Our attempt on line 23 to reference it causes a syntax error.

**Overuse of Class-Level Variables**

Overuse of class-level variables can lead to problems as programs become larger and more complex. While debugging a program, if you find the wrong value stored in a class-level variable, you’ll have to track down every statement that accesses it to determine where the bad value is coming from. In a large program, this can be a tedious and time-consuming process.

Also, when two or more procedures modify the same variable, you must ensure that one procedure cannot upset the accuracy or correctness of another procedure by modifying a class-level variable. Class-level variables should only be used when variables must be shared between two or more class methods. In such cases, local variables are not adequate.

**Checkpoint**

4.20 What is the difference between a class-level variable and a local variable?
4.21 Where do you declare class-level variables?

**Focus on Program Design and Problem Solving: Building the Health Club Membership Fee Calculator Application**

**CONCEPT:** In this section you build the \textit{Health Club Membership Fee Calculator} application. It will use features discussed in this chapter, including \texttt{If} statements, a \texttt{Select Case} statement, radio buttons, and check boxes.

The Bay City Health and Fitness Club charges the following monthly membership rates:

- Standard adult membership: \$40/month
- Child (age 12 and under): \$20/month
- Student: \$25/month
- Senior citizen (age 65 and over): \$30/month
The club also offers the following optional services, which increase the base monthly fee:

- Yoga lessons: add $10 to the monthly fee
- Karate lessons: add $30 to the monthly fee
- Personal trainer: add $50 to the monthly fee

Discounts are available, depending on the length of membership:

- 1–3 months: No discount
- 4–6 months: 5% discount
- 7–9 months: 8% discount
- 10 or more months: 10% discount

The manager of the club has asked you to create a *Health Club Membership Fee Calculator* application. It should allow the user to select a membership rate, select optional services, and enter the number of months of the membership. It should calculate the member’s monthly and total charges for the specified number of months. The application should also validate the number of months entered by the user. An error message should be displayed if the user enters a number less than 1 or greater than 24. (Membership fees tend to increase every two years, so there is a club policy that no membership package can be purchased for more than 24 months at a time.)

Figure 4-35 shows a sketch of the application’s form. The figure also shows the name of each control with a programmer-defined name.

![Figure 4-35 Sketch of the *Health Club Membership Fee Calculator* form](image)

Table 4-12 lists each control, along with any relevant property settings.

<table>
<thead>
<tr>
<th>Control Type</th>
<th>Control Name</th>
<th>Property</th>
<th>Property Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td>(Default)</td>
<td>Text:</td>
<td><em>Membership Fee Calculator</em></td>
</tr>
<tr>
<td>Group box</td>
<td>(Default)</td>
<td>Text:</td>
<td><em>Type of Membership</em></td>
</tr>
<tr>
<td>Radio button</td>
<td>radAdult</td>
<td>Text:</td>
<td><em>Standard &amp; Adult</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Checked:</td>
<td>True</td>
</tr>
<tr>
<td>Radio button</td>
<td>radChild</td>
<td>Text:</td>
<td><em>Child (12 &amp; under)</em></td>
</tr>
<tr>
<td>Radio button</td>
<td>radStudent</td>
<td>Text:</td>
<td><em>Student</em></td>
</tr>
<tr>
<td>Radio button</td>
<td>radSenior</td>
<td>Text:</td>
<td><em>Senior Citizen</em></td>
</tr>
<tr>
<td>Radio button</td>
<td>chkYoga</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radio button</td>
<td>chkKarate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radio button</td>
<td>chkTrainer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text field</td>
<td>txtMonths</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Button</td>
<td>btnCalculate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Button</td>
<td>btnClear</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Button</td>
<td>btnExit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Label</td>
<td>lblMonthlyFee</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Label</td>
<td>lblTotalFee</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(continues)
4.13 Focus on Program Design and Problem Solving: Building the Health Club Membership Fee Calculator Application

Table 4-12 Health Club Membership Fee Calculator controls (continued)

<table>
<thead>
<tr>
<th>Control Type</th>
<th>Control Name</th>
<th>Property</th>
<th>Property Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radio button</td>
<td>radSenior</td>
<td>Text:</td>
<td>Senior Citizen</td>
</tr>
<tr>
<td>Group box</td>
<td>(Default)</td>
<td>Text:</td>
<td>Options</td>
</tr>
<tr>
<td>Check box</td>
<td>chkYoga</td>
<td>Text:</td>
<td>Yoga</td>
</tr>
<tr>
<td>Check box</td>
<td>chkKarate</td>
<td>Text:</td>
<td>Karate</td>
</tr>
<tr>
<td>Check box</td>
<td>chkTrainer</td>
<td>Text:</td>
<td>Personal Trainer</td>
</tr>
<tr>
<td>Group box</td>
<td>(Default)</td>
<td>Text:</td>
<td>Membership Length</td>
</tr>
<tr>
<td>Label</td>
<td>(Default)</td>
<td>Text:</td>
<td>Monthly Fee:</td>
</tr>
<tr>
<td>Text box</td>
<td>txtMonths</td>
<td>Text:</td>
<td>Membership Fees</td>
</tr>
<tr>
<td>Group box</td>
<td>(Default)</td>
<td>Text:</td>
<td>Total:</td>
</tr>
<tr>
<td>Label</td>
<td>(Default)</td>
<td>BorderStyle:</td>
<td>Fixed3D</td>
</tr>
<tr>
<td>Label</td>
<td>lblMonthlyFee</td>
<td>Text:</td>
<td>Initially cleared</td>
</tr>
<tr>
<td>Label</td>
<td>lblTotalFee</td>
<td>BorderStyle:</td>
<td>Fixed3D</td>
</tr>
<tr>
<td>Button</td>
<td>btnCalculate</td>
<td>Text:</td>
<td>Calculate</td>
</tr>
<tr>
<td>Button</td>
<td>btnClear</td>
<td>Text:</td>
<td>Clear</td>
</tr>
<tr>
<td>Button</td>
<td>btnExit</td>
<td>Text:</td>
<td>Exit</td>
</tr>
</tbody>
</table>

Table 4-13 lists and describes the event procedures (event handlers) needed for this application.

**Table 4-13 Health Club Membership Fee Calculator event procedures**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>btnCalculate_Click</td>
<td>First, this procedure validates the number of months entered by the user. Then, if the input is valid, it calculates the monthly fees and the total fee for the time period. Charges for optional services and discounts are included. If the input is not valid, it displays an error message.</td>
</tr>
<tr>
<td>btnClear_Click</td>
<td>Clears the text box, output labels, and check boxes, and resets the radio buttons so that radAdult is selected.</td>
</tr>
<tr>
<td>btnExit_Click</td>
<td>Ends the application.</td>
</tr>
</tbody>
</table>

Figure 4-36 shows a flowchart for the btnCalculate_Click event procedure.

The number of months entered by the user is tested to determine whether it is valid. If the value is less than 1 or greater than 24, an error message is displayed. If the number of months is valid, the fees are calculated.

The first three processes in the calculation are (1) calculate the base monthly fee, (2) calculate and add the cost of optional services, and (3) determine the discount, if any. Each of these processes can be expanded into more detailed flowcharts. Figure 4-37 shows a more detailed view of the calculate the base monthly fee process.
Figure 4-36 Flowchart for btnCalculate_Click

Figure 4-37 Flowchart of calculate the base monthly fee process
The logic in the flowchart can be expressed by the following pseudocode:

\[
\text{If Member is an Adult Then} \\
\quad \text{Monthly Base Fee} = 40 \\
\text{ElseIf Member is a Child Then} \\
\quad \text{Monthly Base Fee} = 20 \\
\text{ElseIf Member is a Student Then} \\
\quad \text{Monthly Base Fee} = 25 \\
\text{ElseIf Member is a Senior Citizen Then} \\
\quad \text{Monthly Base Fee} = 30 \\
\text{End If}
\]

Figure 4-38 shows a more detailed view of the calculate and add the cost of optional services process.
Chapter 4  Making Decisions and Working with Strings

The logic in the flowchart can be expressed with the following pseudocode:

If Yoga is selected Then
   Add 10 to the monthly base fee
End If
If Karate is selected Then
   Add 30 to the monthly base fee
End If
If Personal Trainer is selected Then
   Add 50 to the monthly base fee
End If

Figure 4-39 shows a more detailed view of the determine the discount, if any process.

Figure 4-39  Flowchart of determine the discount, if any process

The logic in the flowchart may be expressed with the following pseudocode:

Select number of months
   Case Is <= 3
      Discount = 0% of monthly base fee
   Case 4 To 6
      Discount = 5% of monthly base fee
   Case 7 To 9
      Discount = 8% of monthly base fee
   Case Is >= 10
      Discount = 10% of monthly base fee
End Select

Tutorial 4-10 builds the Health Club Membership Fee Calculator application.

Tutorial 4-10:
Building the Health Club Membership Fee Calculator application

Step 1:  Create a new Windows application project named Health Club Membership Fee Calculator.
4.13 Focus on Program Design and Problem Solving: Building the Health Club Membership Fee Calculator Application

Step 2: Set up the form as shown in Figure 4-40. Create the group boxes, radio buttons, and check boxes. Refer to the sketch in Figure 4-35 for the control names and Table 4-12 for the relevant property settings of each control.

Step 3: Once you have placed all the controls on the form and set their properties, you can write the application’s code. First, you will write the Dim statements for class-level named constants used when calculating discounts. Click the View Code button ( ) on the Solutions Explorer window to open the Code window. Write the remarks and declarations shown on the next page. (Because these named constants are class-level, they will not be declared inside any procedure.)

Figure 4-40 Membership Fee Calculator form

---

Public Class Form1
Inherits System.Windows.Forms.Form

' Health Club Membership Fee Calculator.
' This application calculates the monthly and
total membership fees for the Bay City Health
and Fitness Club. The base monthly fees are:

' Standard adult membership: $40/month
' Child (12 and under): $20/month
' Student: $25/month
' Senior citizen (65 and over): $30/month

' The club offers the following optional services,
which increase the base monthly fee:

' Yoga lessons: Add $10 to the monthly fee
' Karate lessons: Add $30 to the monthly fee
' Personal trainer: Add $50 to the monthly fee

' Discounts are available, depending on the length
of membership. Here is a list of discounts:

' 1 - 3 months: No discount
' 4 - 6 months: 5% discount
' 7 - 9 months: 8% discount
' 10 or more months: 10% discount
' The following class-level constants are used ' to calculate discounts.
Const decDiscount4to6 As Decimal = 0.05D ' 4 to 6 months
Const decDiscount7to9 As Decimal = 0.08D ' 7 to 9 months
Const decDiscount10orMore As Decimal = 0.1D ' 10 or more mo.

Step 4: Open the Design window and double-click the Calculate button to create the code template for the control’s Click event procedure. Complete the event procedure by typing the following lines (shown in bold):

Private Sub btnCalculate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCalculate.Click
' This method calculates and displays the membership fees.
' Declare local variables.
Dim decBaseFee As Decimal ' Base Monthly Fee
Dim decDiscount As Decimal ' Discount
Dim decTotalFee As Decimal ' Total Membership Fee
Dim intMonths As Integer ' Number of months
' Check the number of months and exit if it contains invalid data.
If Not Integer.TryParse(txtMonths.Text, intMonths) Then
    MessageBox.Show("Months must be a valid integer", "Input Error")
    Return
End If
' Check the month range: must be 1-24.
If (intMonths < 1) Or (intMonths > 24) Then
    MessageBox.Show("Months must be a valid integer", "Input Error")
    Return
End If
We use the first If...Then statement to prevent a runtime error if the user enters a nonnumeric value in the txtMonths control. The Return statement exits the method if an error is found because there is no point in executing any other code if there is invalid data. Similarly, the second If...Then statement checks the range of intMonths so it will always be between 1 and 24.

Step 5: Immediately after the code you just inserted, add statements that calculate the base monthly fee, based on the type of membership requested by the user. The four radio buttons (radAdult, radChild, radStudent, and radSenior) determine the base fee:

' If we reach this point, we assume the input data is valid.
' Calculate the base monthly fee.
If radAdult.Checked = True Then
decBaseFee = 40
ElseIf radChild.Checked = True Then
decBaseFee = 20
ElseIf radStudent.Checked = True Then
decBaseFee = 25
ElseIf radSenior.Checked = True Then
decBaseFee = 30
End If
**Step 6:** Immediately after the code you just inserted, add statements that examine each of the check boxes (chkYoga, chkKarate, chkTrainer) to determine the additional services. Each will add a value to the base fee:

```vbnet
' Look for additional services.
If chkYoga.Checked = True Then
    decBaseFee += 10
End If
If chkKarate.Checked = True Then
    decBaseFee += 30
End If
If chkTrainer.Checked = True Then
    decBaseFee += 50
End If
```

**Step 7:** Based on the number of months, you must multiply a percentage value by the base fee to determine the amount of discount the member will receive. Insert the following code immediately following the previous code:

```vbnet
' Determine the discount, based on membership months.
Select Case intMonths
    Case Is <= 3
        decDiscount = 0
    Case 4 To 6
        decDiscount = decBaseFee * decDiscount4to6
    Case 7 To 9
        decDiscount = decBaseFee * decDiscount7to9
    Case Is >= 10
        decDiscount = decBaseFee * decDiscount10orMore
End Select
```

**Step 8:** Finally, subtract the discount from the base fee and calculate the total fee. Assign the base fee and monthly fee to labels. Insert the following code, finishing the btnCalculate_Click method:

```vbnet
' Adjust for discounts, calculate total fee.
decBaseFee -= decDiscount
decTotalFee = decBaseFee * intMonths

' Display the fees.
lblMonthlyFee.Text = decBaseFee.ToString("c")
lblTotal.Text = decTotalFee.ToString("c")
End Sub
```

**Step 9:** In the Design window, double-click the Clear button and insert the following code in its event handler:

```vbnet
Private Sub btnClear_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnClear.Click

    ' Clear the form, reset the buttons and check boxes.
    radAdult.Checked = True
    chkYoga.Checked = False
    chkKarate.Checked = False
    chkTrainer.Checked = False
    txtMonths.Clear()
    lblMonthlyFee.Text = String.Empty
    lblTotalFee.Text = String.Empty
End Sub
```
Step 10: In the Design window, double-click the Exit button and insert the following code in its event handler:

```vbnet
Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnExit.Click
' End the application by closing the window.
Me.Close()
End Sub
```

Step 11: Build the project by clicking the Build Health Club Membership Fee Calculator selection in the Build menu. Correct any syntax errors that result. Save the project.

Step 12: Run the application. If there are errors, refer to the code previously shown and correct them. If you make corrections, be sure to save the project again. Once the application runs, enter the following test data and confirm that it displays the correct output.

<table>
<thead>
<tr>
<th>Type of Membership</th>
<th>Monthly Fee</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard adult with yoga, karate, and personal trainer for 6 months</td>
<td>$123.50</td>
<td>$741.00</td>
</tr>
<tr>
<td>Child with karate for 3 months</td>
<td>$50.00</td>
<td>$150.00</td>
</tr>
<tr>
<td>Student with yoga for 12 months</td>
<td>$31.50</td>
<td>$378.00</td>
</tr>
<tr>
<td>Senior citizen with karate and personal trainer for 8 months</td>
<td>$101.20</td>
<td>$809.60</td>
</tr>
</tbody>
</table>

Step 13: End the application.
Summary

4.1 The Decision Structure

- Programs often need more than one path of execution. Many algorithms require a program to execute some statements only under certain circumstances. The decision structure accomplishes this.

4.2 The If...Then Statement

- The If...Then statement can cause other statements to execute under certain conditions.
- Relational expressions can only be evaluated as True or False.
- Math operators and function calls can be used with relational operators.

4.3 The If...Then...Else Statement

- The If...Then...Else statement executes one group of statements if a condition is true and another group of statements if the condition is false.

4.4 The If...Then...ElseIf Statement

- The If...Then...ElseIf statement is like a chain of If...Then...Else statements that perform their tests, one after the other, until one of them is found to be true.

4.5 Nested If Statements

- A nested If statement is an If statement in the conditionally executed code of another If statement.

4.6 Logical Operators

- Logical operators connect two or more relational expressions into one (using And, Or, AndAlso, OrElse, or Xor), or reverse the logic of an expression (using Not).
- When determining whether a number is inside a numeric range, it’s best to use the And operator.
- When determining whether a number is outside a range, it’s best to use the Or operator.

4.7 Comparing, Testing, and Working with Strings

- Relational operators can be used to compare strings.
- An empty string is represented by two quotation marks, with no space between them.
- The intrinsic IsNumeric function accepts a string as its argument and returns True if the string contains a number. The function returns False if the string’s contents cannot be recognized as a number.
- The Substring method extracts a specified number of characters from within a specified position in a string.
- The IndexOf method is used to search for a character or a string within a string.

4.8 Focus on GUI Design: The Message Box

- Message boxes are displayed with the MessageBox.Show method. The types of buttons and an icon to display in the message box can be specified.
- The return value of the MessageBox.Show method can be tested to determine which button the user clicked to dismiss the message box.
Chapter 4 Making Decisions and Working with Strings

- The value ControlChars.CrLf can be concatenated with a string to produce multiple line displays.

4.9 The Select Case Statement
- The Select Case statement tests the value of an expression only once, and then uses that value to determine which set of statements to branch to.

4.10 Introduction to Input Validation
- The accuracy of a program's output depends on the accuracy of its input. It is important that applications perform input validation on the values entered by the user.

4.11 Focus on GUI Design: Radio Buttons and Check Boxes
- Radio buttons appear in groups and allow the user to select one of several possible options. Radio buttons placed inside a group box are treated as one group, separate and distinct from any other groups of radio buttons. Only one radio button in a group can be selected at any time.
- Clicking on a radio button selects it and automatically deselects any other radio button selected in the same group.
- Check boxes allow the user to select or deselect items. Check boxes are not mutually exclusive. There may be one or more check boxes on a form, and any number of them can be selected at any given time.

4.12 Class-Level Variables
- Class-Level variables are visible to all procedures (methods) in the same class.

4.13 Focus on Program Design and Problem Solving: Building the Health Club Membership Fee Calculator Application
- This section outlines the process of building the Health Club Membership Fee Calculator application using the features discussed in the chapter.

Key Terms
And operator
AndAlso operator
Boolean expression
CheckBox control
CheckedChanged event
Checked property
class-level variable
class scope
conditionally executed (statement)
ControlChars.CrLf
decision structure
empty string
flag
If...Then
If...Then...Else
If...Then...ElseIf
IndexOf method
input validation
IsNumeric function
leading space
Length property
local scope
logic error
logical operators
message box
MessageBox.Show method
module scope
nested If statement
Not operator
Or operator
OrElse operator
RadioButton control
relational expression
relational operator
scope
Select Case statement
Review Questions and Exercises

Fill-in-the-Blank

1. A _______________ statement, when equal to True, can permit one or more other statements to execute.

2. A(n) _______________ operator determines if a specific relationship exists between two values.

3. Relational expressions can only be evaluated as _______________ or _______________.

4. A(n) _______________ is a Boolean variable that signals when some condition exists in the program.

5. A non-Boolean variable or expression is considered _______________ if its value is 0. If its value is anything other than 0, it is considered _______________.

6. The _______________ statement will execute one group of statements if the condition is true, and another group of statements if the condition is false.

7. The _______________ statement is like a chain of If...Then...Else statements. They perform their tests, one after the other, until one of them is found to be true.

8. A _______________ If statement is an If statement in the conditionally executed code of another If statement.

9. _______________ operators connect two or more relational expressions into one or reverse the logic of an expression.

10. The _______________ method returns the uppercase equivalent of a string.

11. The _______________ returns a lowercase version of a string.

12. The _______________ intrinsic function accepts a string as its argument and returns True if the string contains a number, or False if the string’s contents cannot be recognized as a number.

13. The _______________ method returns the number of characters in a string.

14. The _______________ method returns a copy of a string without leading spaces.

15. The _______________ method returns a copy of a string without trailing spaces.

16. The _______________ method returns a copy of the string without leading or trailing spaces.

17. The _______________ method extracts a specified number of characters from within a specified position in a string.

18. You can display message boxes with the _______________ method.

19. The value _______________ can be concatenated with a string to produce multiple line displays.
20. In a(n) ____________ statement, one of several possible actions is taken, depending on the value of an expression.

21. ____________ is the process of inspecting input values and determining whether they are valid.

22. _______ usually appear in groups and allow the user to select one of several possible options.

23. ____________ may appear alone or in groups and allow the user to make yes/no, or on/off selections.

24. A ______________ variable may be accessed by statements in any procedure in the same file as the variable’s declaration.

**True or False**

Indicate whether the following statements are true or false.

1. T  F: It is not possible to write an expression that contains more than one logical operator.

2. T  F: It is not possible to write expressions that contain math, relational, and logical operators.

3. T  F: You may use the relational operators to compare strings.

4. T  F: Clicking on a radio button selects it, and leaves any other selected radio button in the same group selected as well.

5. T  F: Radio buttons that are placed inside a group box are treated as one group, separate and distinct from any other groups of radio buttons.

6. T  F: When a group of radio buttons appears on a form (outside of a group box), any number of them can be selected at any time.

7. T  F: You may have one or more check boxes on a form, and any number of them can be selected at any given time.

8. T  F: The **If**...Then statement is an example of a sequence structure.

9. T  F: The **Dim** statement for a class-level variable appears inside a procedure.

10. T  F: The **Substring** method returns a lowercase copy of a string.

**Multiple Choice**

1. Relational operators allow you to ________________ numbers.
   a. Add
   b. Multiply
   c. Compare
   d. Average

2. This statement can cause other program statements to execute only under certain conditions.
   a. **MessageBox.Show**
   b. **Decide**
   c. **If**
   d. **Execute**
3. This is a variable, usually a Boolean, that signals when a condition exists.
   a. Relational operator
   b. Flag
   c. Arithmetic operator
   d. Float

4. This statement is like a chain of If statements. They perform their tests, one after
   the other, until one of them is found to be true.
   a. If...Then
   b. If...Then...ElseIf
   c. Chain...If
   d. Relational

5. When placed at the end of an If...Then...ElseIf statement, this provides
   default action when none of the ElseIf statements have true expressions.
   a. Trailing If
   b. Trailing Select
   c. Trailing Otherwise
   d. Trailing Else

6. When an If statement is placed within the conditionally executed code of another
   If statement, it is known as this type of statement.
   a. A nested If
   b. A complex If
   c. A compound If
   d. An invalid If

7. This operator connects two expressions into one. One or both expressions must be
   true for the overall expression to be true. It is only necessary for one to be true, and
   it does not matter which.
   a. And
   b. Or
   c. Xor
   d. Not

8. This operator connects two expressions into one. Both expressions must be true for
   the overall expression to be true.
   a. And
   b. Or
   c. Xor
   d. Not

9. This operator reverses the logical value of an expression. It makes a true expression
   false and a true expression true.
   a. And
   b. Or
   c. Xor
   d. Not

10. This operator connects two expressions into one. One, and only one, of the expres-
    sions must be true for the overall expression to be true. If both expressions are true,
    or if both expressions are false, the overall expression is false.
    a. And
    b. Or
    c. Xor
    d. Not
11. When determining whether a number is inside a numeric range, it’s best to use this logical operator.
   a. And  
   b. Or  
   c. Xor  
   d. Not

12. When determining whether a number is outside a range, it’s best to use this logical operator.
   a. And  
   b. Or  
   c. Xor  
   d. Not

13. In code you should test this property of a radio button or a check box to determine whether it is selected.
   a. Selected  
   b. Checked  
   c. On  
   d. Toggle

14. This method which is part of the numeric classes, attempts to convert a value to a certain numeric or date type.
   a. NumericConvert  
   b. InputConvert  
   c. TryParse  
   d. TryConvert

15. str is a string variable. This statement returns the length of the string stored in str.
   a. Length(str)  
   b. str.Length  
   c. str.StringSize  
   d. CharCount(str)

16. Use this method to display a message box and determine which button the user clicked to dismiss the message box.
   a. MessageBox.Show  
   b. MessageBox.Button  
   c. MessageBox  
   d. MessageBox.UserClicked

Short Answer

1. Describe the difference between the If...Then...ElseIf statement and a series of If...Then statements.

2. In an If...Then...ElseIf statement, what is the purpose of a trailing Else?

3. What is a flag and how does it work?

4. Can an If statement test expressions other than relational expressions? Explain.

5. Briefly describe how the And operator works.

6. Briefly describe how the Or operator works.

7. How is the Xor operator different from the Or operator?
8. How is the AndAlso operator different from the And operator?

9. How is the OrElse operator different from the Or operator?

**What Do You Think?**

1. Why are the relational operators called relational?

2. Answer the following questions about relational expressions with a yes or no.
   a. If it is true that \( x > y \) and it is also true that \( x < z \), does that mean \( x < z \) is true?
   b. If it is true that \( x \geq y \) and it is also true that \( z = x \), does that mean that \( z = y \) is true?
   c. If it is true that \( x \neq y \) and it is also true that \( x \neq z \), does that mean that \( z \neq y \) is true?

3. Why do most programmers indent the conditionally executed statements in a decision structure?

4. Explain why you cannot convert the following If...Then...ElseIf statement into a Select Case statement.
   ```vbs
   If sngTemperature = 100 Then
     intX = 0
   ElseIf intPopulation > 1000 Then
     intX = 1
   ElseIf sngRate < .1 Then
     intX = -1
   End If
   ```

**Find the Error**

1. What is syntactically incorrect in each of the following statements?
   a. If intX > 100
      ```vbs
      MessageBox.Show("Invalid Data")
      End If
      ```
   b. Dim str As String = "Hello"
      Dim intLength As Integer
      intLength = Length(str)
   c. If intZ < 10 Then
      ```vbs
      MessageBox.Show("Invalid Data")
      ```
   d. Dim str As String = "123"
      If str.IsNumeric Then
      ```vbs
      MessageBox.Show("It is a number.")
      ```
   e. Select Case intX
      ```vbs
      Case < 0
        MessageBox.Show("Value too low.")
      Case > 100
        MessageBox.Show("Value too high.")
      Case Else
        MessageBox.Show("Value just right.")
      End Select
      ```

**Algorithm Workbench**

1. Read the following instructions for cooking a pizza, and then design a flowchart with a decision structure that shows the necessary steps to cook the pizza with either thin and crispy or thick and chewy crust.
266  Chapter 4  Making Decisions and Working with Strings

a. For thin and crispy crust, do not preheat the oven. Bake pizza at 450 degrees for 15 minutes.
b. For thick and chewy crust, preheat the oven to 400 degrees. Bake pizza for 20 minutes.

2. Write an If statement that assigns 0 to intX when intY is equal to 20.
3. Write an If statement that multiplies decPayRate by 1.5 when hours is greater than 40.
4. Write an If statement that assigns 0.2 to decCommissionRate when sales is greater than or equal to $10,000.00.
5. Write an If statement that sets the variable intFees to 50 when the Boolean variable blnIsMax equals True.
6. Write an If...Then...Else statement that assigns 1 to intX when intY is equal to 100. Otherwise it should assign 0 to intX.
7. The string variable strPeople contains a list of names, such as Bill Jim Susan Randy Wilma and so on. Write code that searches people for Gene. If Gene is found in strPeople, display a message box indicating that Gene was found.
8. Write an If...Then statement that prints the message The number is valid if the variable sngSpeed is within the range 0 through 200.
9. Write an If...Then statement that prints the message The number is not valid if the variable sngSpeed is outside the range 0 through 200.
10. Convert the following If...Then...ElseIf statement into a Select Case statement.

If intSelection = 1 Then
    MessageBox.Show("Pi times radius squared")
ElseIf intSelection = 2 Then
    MessageBox.Show("Length times width")
ElseIf intSelection = 3 Then
    MessageBox.Show("Pi times radius squared times height")
ElseIf intSelection = 4 Then
    MessageBox.Show("Well okay then, good bye!")
Else
    MessageBox.Show("Not good with numbers, eh?")
End If

Programming Challenges

1. Larger and Smaller

Create an application that allows the user to enter two integers on a form similar to the one shown in Figure 4-41. The application should determine which value is larger than the other, or it should determine that the values are equal. Before comparing the numbers, use the TryParse method to verify that both inputs are valid integers. If an error is found, display an appropriate message to the user. Use a Label control to display all messages. The Exit button should close the window.
2. Roman Numeral Converter

Create an application that allows the user to enter an integer between 1 and 10 into a text box on a form similar to the one shown in Figure 4-42. Use a Select Case statement to identify which Roman numeral is the correct translation of the integer. Display the Roman numeral in a Label control. If the user enters an invalid value, display an appropriate error message and do not attempt the conversion. Include an Exit button that closes the window.

Input validation: Do not accept a number less than 1 or greater than 10. If the user enters a number outside this range, display an error message.
3. **Fat Percentage Calculator**

   Create an application that allows the user to enter the number of calories and fat grams in a food. The application should display the percentage of the calories that come from fat. If the calories from fat are less than 30% of the total calories of the food, it should also display a message indicating the food is low in fat. (Display the message in a label or a message box.) The application’s form should appear similar to the one shown in Figure 4-43.

   One gram of fat has 9 Calories, so:

   \[
   \text{Calories from fat} = \text{fat grams} \times 9
   \]

   The percentage of calories from fat can be calculated as:

   \[
   \text{Percentage of calories from fat} = \frac{\text{Calories from fat}}{\text{total calories}}
   \]

   **Figure 4-43** Fat Gram Calculator form

   *Input validation:* Make sure the number of calories and fat grams are numeric, and are not less than 0. Also, the number of calories from fat cannot be greater than the total number of calories. If that happens, display an error message indicating that either the calories or fat grams were incorrectly entered.

   Use the following test data to determine if the application is calculating properly:

<table>
<thead>
<tr>
<th>Calories and Fat</th>
<th>Percentage Fat</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 calories, 8 fat grams</td>
<td>Percentage of calories from fat: 36%</td>
</tr>
<tr>
<td>150 calories 2 fat grams</td>
<td>Percentage of calories from fat: 12% (a low-fat food)</td>
</tr>
<tr>
<td>500 calories, 30 fat grams</td>
<td>Percentage of calories from fat: 54%</td>
</tr>
</tbody>
</table>

4. **Running the Race**

   Create an application that allows the user to enter the names of three runners and the time it took each of them to finish a race. The application should display who came in first, second, and third place. You can assume that the two runners will never have exactly the same finishing times. The application’s form should appear similar to the one shown in Figure 4-44. The *Clear* button should clear all text boxes and calculated labels. The *Exit* button should close the window. Include the following input error checking: No runner name can be blank, and finishing times must be both numeric and positive.

   Use the following test data to determine if the application is calculating properly:

<table>
<thead>
<tr>
<th>Names and Times</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>John, 87 seconds</td>
<td>First place: Carol</td>
</tr>
<tr>
<td>Carol, 74 seconds</td>
<td>Second place: John</td>
</tr>
<tr>
<td>Shelly, 94 seconds</td>
<td>Third place: Shelly</td>
</tr>
</tbody>
</table>
5. **Software Sales**

A software company sells three packages, Package A, Package B, and Package C, which retail for $99, $199, and $299, respectively. Quantity discounts are given according to the following table:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Discount</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 through 19</td>
<td>20%</td>
</tr>
<tr>
<td>20 through 49</td>
<td>30%</td>
</tr>
<tr>
<td>50 through 99</td>
<td>40%</td>
</tr>
<tr>
<td>100 or more</td>
<td>50%</td>
</tr>
</tbody>
</table>

Create an application that allows the user to enter the number of units sold for each software package. The application’s form should resemble Figure 4-45.
Chapter 4 Making Decisions and Working with Strings

The application should calculate and display the order amounts and the grand total in a Label control. Error checking: make sure all inputs are valid integers and not negative. The Clear button must clear all text boxes and calculated labels. The Exit button must close the window. Suggestion: use a separate Select Case statement for each software package. Each Select Case statement should list the given quantity ranges, and for each, determine the software package price.

Input validation: Make sure the number of units for each package is numeric, and is not negative.

Use the following test data to determine if the application is calculating properly:

<table>
<thead>
<tr>
<th>Units Sold</th>
<th>Amount of Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package A: 15 units</td>
<td>Package A: $1,188.00</td>
</tr>
<tr>
<td>Package B: 75 units</td>
<td>Package B: $8,955.00</td>
</tr>
<tr>
<td>Package C: 120 units</td>
<td>Package C: $17,940.00</td>
</tr>
<tr>
<td>Grand Total:</td>
<td>$28,083.00</td>
</tr>
</tbody>
</table>

Design Your Own Forms

6. Bank Charges

A bank charges $10 per month, plus the following check fees for a commercial checking account:

- $0.10 each for less than 20 checks
- $0.08 each for 20 through 39 checks
- $0.06 each for 40 through 59 checks
- $0.04 each for 60 or more checks

Create an application that allows the user to enter the number of checks written. The application should compute and display the bank’s service fees for the month. All checks for the month are assigned the same charge, based on the total number of checks written during the month. Suggestion: use a Select Case statement to assign the per-check processing fee.

Input validation: Do not accept a negative value for the number of checks written. Ensure that all values are numeric. The Clear button must clear the text box and the label that displays the monthly service charge.

Use the following test data to determine if the application is calculating properly:

<table>
<thead>
<tr>
<th>Number of Checks</th>
<th>Total Fees</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>$ 11.50</td>
</tr>
<tr>
<td>25</td>
<td>$ 12.00</td>
</tr>
<tr>
<td>45</td>
<td>$ 12.70</td>
</tr>
<tr>
<td>75</td>
<td>$ 13.00</td>
</tr>
</tbody>
</table>

7. Shipping Charges

The Fast Freight Shipping Company charges the rates listed in the following table.

<table>
<thead>
<tr>
<th>Weight of the Package (in kilograms)</th>
<th>Shipping Rate per Mile</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 kg or less</td>
<td>$0.01</td>
</tr>
<tr>
<td>Over 2 kg, but not more than 6 kg</td>
<td>$0.015</td>
</tr>
<tr>
<td>Over 6 kg, but not more than 10 kg</td>
<td>$0.02</td>
</tr>
<tr>
<td>Over 10 kg, but not more than 20 kg</td>
<td>$0.025</td>
</tr>
</tbody>
</table>
Create an application that allows the user to enter the weight of the package and the distance it is to be shipped, and then displays the charges.

**Input validation:** Do not accept values of 0 or less for the weight of the package. Do not accept weights of more than 20 kg (this is the maximum weight the company will ship). Do not accept distances of less than 10 miles or more than 3000 miles. These are the company’s minimum and maximum shipping distances. Suggestion: use the OrElse operator to combine the two range conditions that check for package weights that are too small or too large. Use exception handling to check for nonnumeric data.

Use the following test data to determine if the application is calculating properly:

<table>
<thead>
<tr>
<th>Weight and Distance</th>
<th>Shipping Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5 Kg, 100 miles</td>
<td>$ 1.00</td>
</tr>
<tr>
<td>5 Kg, 200 miles</td>
<td>$ 3.00</td>
</tr>
<tr>
<td>8 Kg, 750 miles</td>
<td>$ 15.00</td>
</tr>
<tr>
<td>15 Kg, 2000 miles</td>
<td>$ 50.00</td>
</tr>
</tbody>
</table>

8. **Speed of Sound**

The following table shows the approximate speed of sound in air, water, and steel.

<table>
<thead>
<tr>
<th>Medium</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Air</td>
<td>1,100 feet per second</td>
</tr>
<tr>
<td>Water</td>
<td>4,900 feet per second</td>
</tr>
<tr>
<td>Steel</td>
<td>16,400 feet per second</td>
</tr>
</tbody>
</table>

Create an application that displays a set of radio buttons allowing the user to select air, water, or steel. Provide a text box to let the user enter the distance a sound wave will travel in the selected medium. Then, when the user clicks a button, the program should display the amount of time it will take. Format the output to two decimal places.

**Input validation:** Do not accept distances less than 0. Always check for nonnumeric data.

Use the following test data to determine if the application is calculating properly:

<table>
<thead>
<tr>
<th>Medium and Distance</th>
<th>Speed of Sound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Air, 10,000 feet</td>
<td>9.09 seconds</td>
</tr>
<tr>
<td>Water, 10,000 feet</td>
<td>2.04 seconds</td>
</tr>
<tr>
<td>Steel, 10,000 feet</td>
<td>0.61 seconds</td>
</tr>
</tbody>
</table>

9. **Freezing and Boiling Points**

The following table lists, in degrees Fahrenheit, the freezing and boiling points of several substances. Create an application that allows the user to enter a temperature. The program should then display a list of the substances that freeze at that temperature, followed by a list of substances that will boil at the same temperature.

<table>
<thead>
<tr>
<th>Substance</th>
<th>Freezing Point</th>
<th>Boiling Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethyl alcohol</td>
<td>–173°</td>
<td>172°</td>
</tr>
<tr>
<td>Mercury</td>
<td>–38°</td>
<td>676°</td>
</tr>
<tr>
<td>Oxygen</td>
<td>–362°</td>
<td>–306°</td>
</tr>
<tr>
<td>Water</td>
<td>32°</td>
<td>212°</td>
</tr>
</tbody>
</table>
Use the following test data and sample outputs to determine if the application is calculating properly:

<table>
<thead>
<tr>
<th>Temperature</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>– 20°</td>
<td>Water will freeze and oxygen will boil.</td>
</tr>
<tr>
<td>– 50°</td>
<td>Mercury and water will freeze and oxygen will boil.</td>
</tr>
<tr>
<td>–200°</td>
<td>Ethyl alcohol, mercury, and water will freeze and oxygen will boil.</td>
</tr>
<tr>
<td>–400°</td>
<td>Ethyl alcohol, mercury, oxygen, and water will freeze.</td>
</tr>
</tbody>
</table>

10. Long-Distance Calls

A long-distance provider charges the following rates for telephone calls:

<table>
<thead>
<tr>
<th>Rate Category</th>
<th>Rate per Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daytime (6:00 a.m. through 5:59 P.M.)</td>
<td>$0.07</td>
</tr>
<tr>
<td>Evening (6:00 p.m. through 11:59 P.M.)</td>
<td>$0.12</td>
</tr>
<tr>
<td>Off-Peak (12:00 a.m. through 5:59 A.M.)</td>
<td>$0.05</td>
</tr>
</tbody>
</table>

Create an application that allows the user to select a rate category (from a set of radio buttons) and enter the number of minutes of the call, then displays the charges. Include a Clear button that clears the input and calculated values, and an Exit button that closes the window. Error checking: the minutes input by the user must be numeric, and it must be greater than zero.

Use the following test data to determine if the application is calculating properly:

<table>
<thead>
<tr>
<th>Rate Category and Minutes</th>
<th>Charge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daytime, 20 minutes</td>
<td>$ 1.40</td>
</tr>
<tr>
<td>Evening, 20 minutes</td>
<td>$ 2.40</td>
</tr>
<tr>
<td>Off-peak, 20 minutes</td>
<td>$ 1.00</td>
</tr>
</tbody>
</table>

11. Internet Service Provider, Part 1

An Internet service provider offers three subscription packages to its customers, plus a discount for nonprofit organizations:

a. Package A: 10 hours of access for $9.95 per month. Additional hours are $2.00 per hour.
b. Package B: 20 hours of access for $14.95 per month. Additional hours are $1.00 per hour.
c. Package C: Unlimited access for $19.95 per month.
d. Nonprofit Organizations: The service provider gives all nonprofit organizations a 20% discount on all packages.

The user should select the package the customer has purchased (from a set of radio buttons) and enter the number of hours used. A check box captioned Nonprofit Organization should also appear on the form. The application should calculate and display the total amount due. If the user selects the Nonprofit Organization check box, a 20% discount should be deducted from the final charges. Implementation note: all rates, limits, and discounts must be declared using symbolic constants (using the Const keyword).

Input validation: The number of hours used in a month cannot exceed 744. The value must be numeric.
Use the following data to determine if the application is calculating properly:

Package and Hours | The Monthly Charge
--- | ---
Package A, 5 hours, nonprofit | $7.96
Package A, 25 hours | $39.95
Package B, 10 hours, nonprofit | $11.96
Package B, 25 hours | $19.95
Package C, 18 hours, nonprofit | $15.96
Package C, 25 hours | $19.95

12. Internet Service Provider, Part 2 (Advanced)

Make a copy of your solution program from Programming Challenge 11. Then, using the copied program, modify it so the form has a check box captioned Display Potential Savings. When this check box is selected, the application should also display the amount of money that Package A customers would save if they purchased Package B or C, or the amount that Package B customers would save if they purchased Package C. If there would be no savings, the message should indicate that. Use the following test data to determine if the application is calculating properly:

Package and Hours | Total Monthly Savings
--- | ---
Package A, 5 hours, nonprofit | $7.96, no savings with Packages B or C
Package A, 25 hours | $39.95, save $20.00 with Package B, and save $20.00 with Package C
Package B, 10 hours, nonprofit | $11.96, no savings with Package C
Package B, 25 hours | $19.95, no savings with Package C

13. Mass and Weight

Scientists measure an object’s mass in kilograms and its weight in newtons. If you know the amount of mass of an object, you can calculate its weight, in newtons, with the following formula:

\[
\text{Weight} = \text{mass} \times 9.8
\]

Create a VB application that lets the user enter an object’s mass and calculates its weight. If the object weighs more than 1000 newtons, display a message indicating that it is too heavy. If the object weighs less than 10 newtons, display a message indicating that it is too light.

14. Book Club Points

Serendipity Booksellers has a book club that awards points to its customers based on the number of books purchased each month. The points are awarded as follows:

- If a customer purchases 0 books, he or she earns 0 points.
- If a customer purchases 1 book, he or she earns 5 points.
- If a customer purchases 2 books, he or she earns 15 points.
- If a customer purchases 3 books, he or she earns 30 points.
- If a customer purchases 4 or more books, he or she earns 60 points.

Create a VB application that lets the user enter the number of books that he or she has purchased this month and displays the number of points awarded.

15. Body Mass Index Program Enhancement

In Programming Challenge 13 in Chapter 3 you were asked to create a VB application that calculates a person’s body mass index (BMI). Recall from that exercise that the BMI is often used to determine whether a person with a sedentary lifestyle is overweight or underweight for their height. A person’s BMI is calculated with the following formula:

\[
\text{BMI} = \frac{\text{weight} \times 703}{\text{height}^2}
\]
In the formula, weight is measured in pounds and height is measured in inches. Enhance the program so it displays a message indicating whether the person has optimal weight, is underweight, or is overweight. A sedentary person’s weight is considered to be optimal if his or her BMI is between 18.5 and 25. If the BMI is less than 18.5, the person is considered to be underweight. If the BMI value is greater than 25, the person is considered to be overweight.

16. Magic Dates

The date June 10, 1960, is special because when we write it in the following format, the month times the day equals the year.

6/10/60

Create a VB application that lets the user enter a month (in numeric form), a day, and a two-digit year. The program should then determine whether the month times the day is equal to the year. If so, it should display a message saying the date is magic. Otherwise it should display a message saying the date is not magic.

17. Time Calculator

Create a VB application that lets the user enter a number of seconds and works as follows:

- There are 60 seconds in a minute. If the number of seconds entered by the user is greater than or equal to 60, the program should display the number of minutes in that many seconds.
- There are 3,600 seconds in an hour. If the number of seconds entered by the user is greater than or equal to 3,600, the program should display the number of hours in that many seconds.
- There are 86,400 seconds in a day. If the number of seconds entered by the user is greater than or equal to 86,400, the program should display the number of days in that many seconds.