CHAPTER 5
Windows Phone 7 Mobile Applications Using Decision Structures

OBJECTIVES
You will have mastered the material in this chapter when you can:

- Write a program for the Windows Phone 7 development platform
- Understand the mobile environment
- Place TextBlock objects in mobile applications
- Place RadioButton objects in applications
- Assign the IsChecked property
- Display a message box
- Make decisions using if…Then statements
- Make decisions using if…Then…Else statements
- Make decisions using nested if statements
- Make decisions using logical operators
- Make decisions using Case statements
- Insert code snippets
- Deploy a mobile program to a Windows Phone 7 emulator
- Understand the Windows Marketplace
Introduction

In the world of computer development, the most exciting trend is the explosion of phone apps (mobile applications) on the cell phone platforms of the major players, such as Microsoft's Windows Phone 7, Apple's iPhone, and Google's Android. Developers have the opportunity to develop apps on a host of different platforms that can be sold directly to millions of mobile phone owners all over the world. Visual Basic 2010 and Visual C# 2010 can be used to develop programs for the Windows Phone 7 platform, which is quickly growing, especially in the business world.

Developers can code Visual Basic applications to make decisions based on the input of users or other conditions that occur. Decision-making is one of the fundamental activities of a computer program. In this chapter, you will learn to write decision-making statements in Visual Basic 2010.

Visual Basic allows you to test conditions and perform different operations depending on the results of that test. You can test for a condition being true or false and change the flow of what happens in a program based on the user’s input.

Chapter Project

The sample program in this chapter is designed to run on a Windows Phone 7 mobile device. The Personal Trainer application is a Windows Phone application developed with Visual Basic for mobile devices and computes how many calories you consume while jogging, biking, or swimming a specified distance anywhere at any time.

The Personal Trainer application requests that the user enter the distance covered by the chosen exercise and the desired exercise type. The mobile application then computes the number of calories burned based on the distance completed using the following values: 100 calories per mile for jogging, 42 calories per mile for biking, and 440 calories per mile for swimming. Figure 5-1 shows the mobile user interface for the application.

In Figure 5-1, the Personal Trainer Phone 7 application displays the titles Exercise Application and Personal Trainer in the title bar. The distance covered is entered in the TextBox object using the on-screen phone keyboard and shows the number of miles completed. The user chooses the exercise type by selecting a RadioButton from the following list: Jogging (the most common choice), Biking, or Swimming. After the user has entered the number of miles completed and selected a type of exercise, the user taps the Calculate button on the multi-touch screen to obtain the calories consumed. The calculation is based on the miles completed multiplied by the number of calories burned by the selected type of exercise. The calories burned are displayed in a Label object. In the example in Figure 5-1, the user entered 10 miles for the distance covered by using the built-in keyboard on the multi-touch screen and then tapped the Biking RadioButton object to select that type of exercise. After tapping the Calculate button, the application displayed 420 calories burned. Due to the mobility of a personal phone, the user can immediately enter their distance covered as soon as they complete their workout to compute the number of calories burned.
Checking the validity of data entered by the user is a requirement of this chapter project program. In Chapter 4, you learned that if you enter nonnumeric data and attempt to use it in a calculation, the program will be terminated. To check for invalid data, the Personal Trainer program ensures that the user enters a numeric value greater than zero in the Miles Completed TextBox object. A warning dialog box appears if the user leaves the Miles Completed TextBox blank or does not enter a valid number. Figure 5-2 on the next page displays a warning dialog box, called a MessageBox, which directs the user to enter a valid number for the distance completed.

Checking input data for validity is an important task in Visual Basic programs. You will learn several data validation techniques in this chapter.

Windows Phone 7 Applications Installation
Programs called mobile applications developed for the Windows Phone 7 platform are written with Visual Basic or C# code using Visual Studio 2010 with two free downloadable add-ons. The standard Visual Studio 2010 Professional program does not support Windows Phone applications, but by installing the two add-ons, components will be added to Visual Studio 2010 Professional to develop these types of mobile applications. To add Windows Phone applications to Visual Studio 2010, open the following two links and install each one in this specific order to complete the chapter project using a phone emulator.
1. Visual Basic for Windows Phone Developer Tools:
2. Windows Phone Developer Tools:

After installation, click New Project, Visual Basic, and then Silverlight For Windows Phone template to ensure that the installation is working properly. The Windows Phone Developer Tools include a phone emulator that allows you to develop applications without having a physical phone, although you should test your application on a real physical device before you deploy it.

Windows Phone 7 Platform Overview
Windows Phone 7 has the same features as the iPhone and Android phones such as an 800 × 480 resolution multi-touch screen, modern user interface, camera, microphone, FM radio, social network capabilities, WiFi support, GPS (location aware applications), and music. Unique to the Microsoft mobile environment is the addition of the current Microsoft Office with the power of Word, Excel, PowerPoint, OneNote, and Outlook. Windows Phone also integrates with Xbox LIVE, which is a great addition for gamers. Windows Phone uses VB and C# code to control the
Silverlight and XNA frameworks. Silverlight is used for most business and traditional apps and XNA is often used for 3-D games.

The front of the hardware case of the Phone 7 device has a multi-touch display that allows you to make selections by tapping the display. The Phone 7 screen incorporates a special touch technology, which only responds to a human fingertip but not to a stylus or other forms of pressure. Phone 7 screens respond to four simultaneous touch points. Also on the case, three hardware buttons are positioned below the display. From left to right, the buttons represent Back, Start, and Search (Figure 5-3).

**IN THE REAL WORLD**

The Gartner Associates stated that the following 10 types of mobile apps are the most popular topics:

1. Money transfer
2. Location-based services
3. Mobile search
4. Mobile browsing
5. Mobile health monitoring
6. Mobile payment
7. Near field communication services
8. Mobile advertising
9. Mobile instant messaging
10. Mobile music

**Creating a Windows Phone 7 Application**

After the Visual Basic for Windows Phone Developer Tools have been installed, the chapter project will be written using Visual Basic 2010 commands and a Windows Phone 7 emulator. The program also can be generated to deploy directly to a
Windows Phone 7 device. In the toolbar at the top, a text box displays “Windows Phone 7 Emulator” by default. The other choice is “Windows Phone 7 Device.” You use this text box to deploy your program to the emulator or an actual phone connected to your computer by a USB cable. To create a Windows Phone Application project, follow these steps:

**STEP 1** Open Visual Studio 2010 Professional after the Visual Basic for Windows Phone Developer Tools have been installed. Click File on the menu bar, and then click New Project. In the left column of installed templates, select Visual Basic and Silverlight For Windows Phone. Select Windows Phone Application. Name the project Personal Trainer.

*The Silverlight For Windows Phone template is selected. The project is named Personal Trainer (Figure 5-4).*
STEP 2 Click the OK button.

The Personal Trainer application opens. The emulator displays the phone device on the tab named MainPage.xaml. The second pane contains the XAML code (pronounced zammel). The third pane contains the Solution Explorer and Properties windows (Figure 5-5).

User Interface Design

The user interface for the Personal Trainer mobile application includes three new objects: a TextBlock, RadioButtons, and MessageBox. The MessageBox appears when the user inputs a negative number. The TextBlock is unique to the Windows Phone Application platform in Visual Basic.

Using the TextBlock Object

The body of the MainPage.xaml (Form object) contains two TextBlock objects with the text MY APPLICATION and page name (Figure 5-6 on the next page). A TextBlock object is similar to a Label object in a Windows application. The MainPage.xaml also
contains a Grid object. A Grid object assists in organizing the Form object. The majority of the objects will be placed on the Grid object; for example, additional TextBlock objects can be added to the Grid. The name of the TextBlock object is placed at the top of the Properties window. The TextBlock object uses the Text property to display text. To change the font color of the TextBlock object, change the Foreground property.

Updating and Adding a TextBlock Object

The TextBlock object has properties similar to the Label object used in Windows applications. The Text property changes the actual displayed text and the FontSize property changes the size of the text. To update and add a TextBlock object, follow these steps:

**STEP 1** To view the entire mobile Form object, place the mouse pointer between the right side of the scroll bar of the emulator Form object and the XAML code window. A two-sided arrow appears. Drag the two-sided arrow to the right until the mobile Form object is fully displayed. Click the MY APPLICATION TextBlock object on the emulator. In the Properties window, click to the right of TextBlock and change ApplicationTitle to lblApplication.

*The MY APPLICATION TextBlock is selected and renamed lblApplication (Figure 5-7 on the next page).*
STEP 2  Change the Text property in the Properties window by double-clicking the right column of the Text property and typing Exercise Application. The Text property of the lblApplication is changed to Exercise Application (Figure 5-8).
STEP 3  Click the page name TextBlock object on the emulator. Name the TextBlock `lblTitle`. Change the Text property to Personal Trainer. Change the FontSize property to size 64.

The second TextBlock is named `lblTitle`. The Text property is changed to Personal Trainer and the FontSize property is now 64 (Figure 5-9).
**STEP 4** Open the Toolbox from the toolbar and drag the TextBlock object to the upper-left corner in the Grid object on the emulator. Name the TextBlock object **lblDistance**. In the Text property, type **Miles Completed:** and then change the FontSize property to size 24. Drag a TextBox object to the right of the lblDistance TextBlock. Name the TextBox object **txtDistance** and change the FontSize property to size 24. Change the VerticalContentAlignment property to Center. Delete the text from the Text property. Change the Width property to 100.

*TextBlock and TextBox objects are added to request the Miles Completed (Figure 5-10).*

**FIGURE 5-10**
Using the RadioButton Object

The Personal Trainer Form object requires RadioButton objects to select the type of exercise (Figure 5-11). RadioButton objects allow the user to make choices. When RadioButton objects are placed on the phone, the user can select only one of the radio buttons. In Windows applications, you can use a GroupBox object to group your RadioButton objects, but in Windows Phone applications, you do not need a special container. If you have two different sets of RadioButton objects, the GroupName property allows you to differentiate multiple groups. The chapter project requires only one group of RadioButtons; for example, in Figure 5-11 the Swimming radio button is selected. If the user clicks the Biking radio button, it will be selected and the Swimming radio button automatically will be deselected. Unlike Windows applications, which use the Text property to display the RadioButton text, Windows Phone applications use the Content property in the Properties window to identify the text.
Adding the RadioButton Objects

The user may select only one type of exercise: Jogging, Biking, or Swimming. To place RadioButton objects on the emulator, you can complete the following steps:

**STEP 1** Drag one RadioButton object from the Toolbox to the Form object. Drag a second RadioButton object from the Toolbox to the Form object, using orange snap lines to align and separate the RadioButton objects vertically.

*The second RadioButton object is aligned vertically with an orange snap line, which separates it vertically from the first RadioButton object (Figure 5-12).*
Release the mouse button to place the second RadioButton object on the Form object. Using the same technique, add a third RadioButton object. Name the RadioButton objects by selecting each RadioButton object and entering the name in the Properties window. The names for the radio buttons, from top to bottom, should be radJogging, radBiking, and radSwimming. Use your arrow keys to align the radio buttons.

The (Name) property is selected. The names radJogging, radBiking, and radSwimming are entered (Figure 5-13).

Change the Content property for each RadioButton by double-clicking the right column of the Content property and typing Jogging for the first RadioButton, Biking for the second RadioButton, and Swimming for the third RadioButton.

The Content property has been changed to the types of exercise: Jogging, Biking, and Swimming (Figure 5-14).
Using the IsChecked Property of RadioButton Objects

You will recall that the RadioButton objects in the Personal Trainer application allow the user to select one exercise type. When the user selects Swimming as the exercise type, as shown in Figure 5-11 on page 12, the RadioButton is selected (the small circle in the radio button is shaded). When a RadioButton is selected, the IsChecked property of the Swimming RadioButton changes from False (unselected) to True (selected).

Often, during design time, you should set the IsChecked property to True for the most commonly selected RadioButton to save the user from having to select the most common choice. In the Personal Trainer application, the user selects Jogging most often. To cause the Jogging RadioButton object named radJogging to appear selected (shaded) when the program begins, you change the IsChecked property for the radJogging RadioButton from False to True (Figure 5-15). In a Windows application the IsChecked property is called the Checked property.

Completing the Phone Form Object

A Button object is tapped on the touch screen after the miles completed and the type of exercise is selected. The Button object uses the Content property to display the word Calculate. Below the Button object, the result is displayed in a TextBlock object called lblResult. An image can also be displayed on a Windows Phone application. It is important to use an image with a small file size for quick loading on the phone device. In a Windows application, the image tool is called a PictureBox object. In a Windows Phone application, the Image object is used to display an image. The Source property provides an Add button to add a link to an image that is placed on the Form object. The image is saved
within the Windows Phone application in an Images folder in the Solution Explorer. To place the Button, TextBlock, and Image objects on the Form object, complete these steps:

**STEP 1** Drag the Button object from the Toolbar to the area below the radio buttons. Name the Button object btnCalculate. Change the Content property to Calculate to change the text on the button.

*The btnCalculate button is placed on the form with the Content property changed to Calculate (Figure 5-16).*

**STEP 2** Drag the TextBlock object for the final result to the area below the Button object. Name this TextBlock object lblResult. Delete the Text property. Change the FontSize property to size 24. Change the Width property to 215.

*The TextBlock is placed below the Calculate button (Figure 5-17).*
**STEP 3** The Exercise.jpg picture can be downloaded from [http://scsite.com/vb2010/ch5/images](http://scsite.com/vb2010/ch5/images). Save the file on your USB drive. Drag the Image object to the bottom of the Form object. Name the object picExercise. Click to the right of the Source property, and then click the ellipsis button.

The Choose image dialog box opens (Figure 5-18).
STEP 4  Click the Add button in the Choose image dialog box. Locate the Exercise.jpg on your USB drive. Click OK. Change the Height property of the Image object to 200 and the Width property to 480. To view the Image object in the emulator, click the Start Debugging button on the toolbar (or press F5). In a few seconds, the Personal Trainer application is displayed in the emulator.

*An Image object is added to the Form object (Figure 5-19).*

![FIGURE 5-19](image)

STEP 5  Position the mouse pointer over the upper-right corner of the emulator and then close the emulator by clicking the Close (X) button. Click the OK button to close the remote connection with the running emulator.
Beginning to Code the Windows Phone Application

To place code in the Windows Phone application to compute the number of calories, follow these steps:

**STEP 1** Double-click the Calculate button to begin coding the btnCalculate event. Close the Toolbox panel on the left.

*The code editor window opens (Figure 5-20).*

![FIGURE 5-20](image)

**STEP 2** Add the comments on the first line of code and the Option Strict On command. Click inside the btnCalculate Click event.

*The standard comments and Option Strict On have been added (Figure 5-21).*

![FIGURE 5-21](image)
Displaying a Message Box

In the Personal Trainer chapter project, a message box, also called a dialog box, opens at the top of the Form object if the user enters a negative value for the miles completed (Figure 5-22).

This message box reminds the user to enter the miles completed. A message box window must be closed before the application can continue. The user can continue the application by clicking the OK button in the message box.

In Visual Basic, the message to the user in a message box window is displayed using a procedure named Show that is found in the MessageBox class. The syntax for the statement to display a message in a message box is shown in Figure 5-23 on the next page. In Windows applications, the MessageBox.Show command works as well, but you can also use the MsgBox command.

The string message shown in the parentheses will appear in the message box when the code is executed. The string message is considered an argument of the procedure. You will recall that an argument is a value that is passed to a procedure. The argument for the MessageBox command contains the message to print in the message box window on top of the phone panel during execution.

The example in Figure 5-23 illustrates the code that could be used in the Calculate button click event handler. This code could be executed if the user clicks the Calculate button without entering a positive number in the Miles Completed text box.
String Concatenation

Recall that when the Personal Trainer application runs, the user enters the miles completed. If the user enters a number that is not greater than zero, such as –5, a message box appears that states “You entered –5, Enter a Positive Number”, as shown in Figure 5-24.

To create the message in the message box, you can use concatenation, which you learned about in Chapter 4. In Figure 5-25, the string message is constructed by joining a string (“You entered”), a variable named decMiles containing the miles completed entered (which must be converted to a string), and a string for the final part of the message (“Enter a Positive Number.”). The code in Figure 5-25 creates the desired message box.

You will recall that the operator to concatenate strings is the ampersand (&). When the statement is executed, the three string elements are joined together (concatenated) to form the one string that is displayed in the message box.
Making Decisions with Conditional Statements

In the Personal Trainer chapter project, which calculates the number of calories burned, the application allows the user to select one of three different types of exercise: Jogging, Biking, or Swimming. The number of calories burned is based on the user’s choice of exercise. To select the exercise type, the user must click one of three radio buttons titled Jogging, Biking, and Swimming. Then, based on the choice, the application uses a different exercise type.

Visual Basic uses decision structures to deal with the different conditions that occur based on the values entered into an application. A decision structure is one of the three fundamental control structures used in computer programming. For example, if the user clicks the Jogging radio button, the number of calories burned is set to 100 calories per mile. The statement that tests the radio button is called a conditional statement. The condition checked is whether the Jogging radio button is selected. If so, the number of calories is set to 100.

When a condition is tested in a Visual Basic program, the condition either is true or false. For example, when checking to determine if the Jogging radio button is selected, the condition can either be true (the Jogging radio button is checked) or false (the Jogging radio button is not checked). All conditional statements result in the tested condition either being true or false.

To implement a conditional statement and the statements that are executed when a condition is true and statements executed when a condition is false, Visual Basic uses the If statement and its variety of formats. You will learn about the If statement in the following sections.

Using an If . . . Then Statement

In the chapter program, an If . . . Then statement is used to determine number of calories burned in one mile. The simplest form of the If . . . Then statement is shown in Figure 5-26.

```
5 If condition Then
6    Statement(s) executed when condition is true
7 End If
```

In Figure 5-26, when the condition tested in the If statement on line 5 is true, the statement(s) between the If and End If keywords will be executed. If the condition is not true, no statements between the If and End If keywords will be executed, and program execution will continue with the statement(s) that follows the End If statement.
Visual Basic automatically indents statements to be executed when a condition is true or not true to indicate the lines of code are within the conditional If . . . Then structure. This is why the statement on line 6 in Figure 5-26 is indented. The End If keyword terminates the If . . . Then block of code. After executing the If . . . Then block, execution continues with any statements that follow the closing End If statement.

**Relational Operators**

In Figure 5-26, the condition portion of the If . . . Then statement means a condition is tested to determine if it is true or false. The conditions that can be tested are:

1. Is one value equal to another value?
2. Is one value not equal to another value?
3. Is one value greater than another value?
4. Is one value less than another value?
5. Is one value greater than or equal to another value?
6. Is one value less than or equal to another value?

To test these conditions, Visual Basic provides relational operators that are used within the conditional statement to express the relationship being tested. The table in Figure 5-27 shows these relational operators.

<table>
<thead>
<tr>
<th>Relational Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Resulting Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>= Equal to</td>
<td>8 = 8</td>
<td>True</td>
</tr>
<tr>
<td>2</td>
<td>&lt;&gt; Not equal to</td>
<td>6 &lt;&gt; 6</td>
<td>False</td>
</tr>
<tr>
<td>3</td>
<td>&gt; Greater than</td>
<td>7 &gt; 9</td>
<td>False</td>
</tr>
<tr>
<td>4</td>
<td>&lt; Less than</td>
<td>4 &lt; 6</td>
<td>False</td>
</tr>
<tr>
<td>5</td>
<td>&gt;= Greater than or equal to</td>
<td>3 &gt;= 3</td>
<td>True</td>
</tr>
<tr>
<td>6</td>
<td>&lt;= Less than or equal to</td>
<td>7 &lt;= 5</td>
<td>False</td>
</tr>
</tbody>
</table>

**FIGURE 5-27**

A condition tested using a relational operator is evaluated as true or false. Example 1 tests whether 8 is equal to 8. Because it is, the resulting condition is true. Example 2 tests if 6 is not equal to 6. Because they are equal, the resulting condition is false. Similarly, Example 5 tests if 3 is greater than or equal to 3. Because they are equal, the resulting condition is true.

As an example of using a conditional operator, consider the following problem where an If statement is used to determine if someone is old enough to vote. If the
value in the `intAge` variable is greater than or equal to 18, then the person is old enough to vote. If not, the person is not old enough to vote. The `If . . . Then` statement to test this condition is shown in Figure 5-28.

```vbnet
8    If intAge >= 18 Then
9       lblVotingEligibility.Text = "You are old enough to vote"
10      End If
```

**FIGURE 5-28**

In Figure 5-28, if the value in the `intAge` variable is greater than or equal to 18, the String value “You are old enough to vote” is assigned to the `Text` property of the `lblVotingEligibility` Label object. If not, then no processing occurs based on the conditional statement and any statement(s) following the `End If` keyword will be executed.

You can see in Figure 5-28 that several keywords are required in an `If . . . Then` statement. The word `If` must be the first item. Next, the condition(s) to be tested are stated, followed by the word `Then`. This keyword is required in an `If` statement.

The `End If` keyword follows the statements to be executed when the condition is true. This entry also is required. It signals to the Visual Basic compiler that statements following it are to be executed regardless of the result of the conditional statement; that is, the `End If` keyword is the last element within the `If` block and no subsequent statements depend on it for execution.

To enter the `If . . . Then` statement shown in Figure 5-28, you can complete the following steps:

**STEP 1** With the insertion point in the correct location in the code, type `if` and then press the SPACEBAR.

*The statement begins with the word `If` (Figure 5-29). The `If` command is displayed in blue because it is a Visual Basic keyword. You can type using uppercase or lowercase letters.*

**STEP 2** Type `inta` to select the variable named `intAge` in the IntelliSense list. Then, type `>=18` as the condition to be tested. Press the ENTER key.
The If . . . Then statement is entered in the code editing window (Figure 5-30). When the ENTER key is pressed, Visual Basic adds the keyword Then to the end of the If statement line of code and inserts spaces between the elements in the statement for ease of reading. In addition, Visual Basic inserts the End If keyword following a blank line. Notice the keywords Then and End If are capitalized and displayed in blue.

FIGURE 5-30

STEP 3 On the blank line (line 12 in Figure 5-30), enter the statement that should be executed when the condition is true. To place the message, “You are old enough to vote” in the Text property of the lblVotingEligibility Label object, insert the code shown in Figure 5-28 on the previous page. Remember to use IntelliSense to reference the lblVotingEligibility Label object.

The resulting statement is entered between the If and End If keywords (Figure 5-31). Notice that Visual Basic automatically indents the line for ease of reading. The blank line allows you to enter more statements. If you have no further statements, you can press the DELETE key to delete the blank line in the If . . . Then statement.

FIGURE 5-31
Comparing Strings

You also can write an If . . . Then statement using the relational operators shown in Figure 5-28 on page 24 to compare String values. A String value comparison compares each character in two strings, starting with the first character in each string. For example, in the two strings in Figure 5-32, the comparison begins with the first character in each string, a. Because the characters are equal, the comparison continues with the second character in each string, b. Because these characters are equal, the comparison continues with the third characters in each string, c. Because all three characters are equal, the strings are considered equal and the resulting condition from the If statement is true.

```vBasic
13 Dim String1 As String = "abc"
14 Dim String2 As String = "abc"
16 If String1 = String2 Then
17 lblStringTest.Text = "Equal"
18 End If
```

**FIGURE 5-32**

All characters found in strings, including letters, numbers, and special characters, are in a sequence from low to high based on the manner in which the characters are coded internally on the computer. When using Visual Studio 2010, characters are stored and sequenced in Unicode, which is a coding methodology that can accommodate more than 60,000 characters. Appendix A in the textbook shows the Unicode sequence for the standard keyboard characters. You will find that the numbers are considered less than uppercase letters, and uppercase letters are considered less than lowercase letters.

Using the If . . . Then statement, the following comparisons produce the following resulting conditions:

**Example 1:**

```vBasic
Dim String1 As String = "Powder"
Dim String2 As String = "Power"
If String1 < String2 Then
```

**Resulting Condition:** True because in the fourth character position, the letter d is less than the letter e.

**Example 2:**

```vBasic
Dim String1 As String = "6"
Dim String2 As String = "T"
If String1 < String2 Then
```

**Resulting Condition:** True because in a string comparison, a number is less than an uppercase letter.
Example 3:

Dim String1 As String = "12"
Dim String2 As String = "9"

If String1 < String2 Then

Resulting Condition: True because in a string comparison, the characters in the first position of the string are compared first. Because the value 1 in String1 is less than the value 9 in String2, the entire value in String1 is considered less than the value in String2.

Example 4:

Dim String1 As String = "anchor"
Dim String2 As String = "Anchorline"

If String1 > String2 Then

Resulting Condition: True because a lowercase letter (the a in the first position of String1) is considered greater than an uppercase letter (the A in the first position of String2).

Comparing Different Data Types

Every type of data available in Visual Basic can be compared. Different numeric types can be compared to each other using an If statement. A single string character can be compared to a Char data type. The following examples illustrate some of the allowable comparisons.

Example 1: Decimal compared to Double

If decQuarterlySales > dblSalesQuota Then

If decQuarterlySales = 110,324.54 and dblSalesQuota = 112,435.54, the condition is false.

Example 2: Decimal compared to Integer

If decTirePressureReading > intTirePressureMaximum Then

If decTirePressureReading = 30.21 and intTirePressureMaximum = 30, the condition is true.

Example 3: Double compared to Integer

If dblCurrentTemperature >= intHeatDanger Then

If dblCurrentTemperature = 94.543 and intHeatDanger = 98, the condition is false.
Example 4: String compared to Char

If strChemistryGrade < chrPassingGrade Then

If strChemistryGrade = “B” and chrPassingGrade = “C”, the condition is true.

Visual Basic allows comparisons between most data types. If you are unsure whether a comparison can be made, write an If statement to ensure the comparison is working properly.

Using the If . . . Then . . . Else Statement
An If . . . Then statement executes a set of instructions if a condition is true. If the condition is false, the instructions between the If statement and the End If statement are not executed and program execution continues with the statement(s) following the End If statement.

In many applications, the logic requires one set of instructions to be executed if a condition is true, and another set of instructions to be executed if a condition is false. For example, a requirement in a program could specify that if a student’s test score is 70 or greater, a message stating “You passed the examination” should be displayed, while if the test score is less than 70, a message stating “You failed the examination” should be displayed.

To execute one set of instructions if a condition is true, and another set of instructions if the condition is false, you can use the If . . . Then . . . Else statement. Figure 5-33 illustrates the syntax of the If . . . Then . . . Else statement.

```
16   If condition Then
17       Statement(s) executed if condition is true
18       Else
19       Statement(s) executed if condition is false
20      End If
```

FIGURE 5-33

In the code in Figure 5-33, if the condition tested by the If statement is true, the statement(s) between the Then keyword and the Else keyword will be executed. If the condition tested is false, the statement(s) between the Else keyword and the End If keyword will be executed.

The example in Figure 5-34 on the next page shows the use of the If . . . Then . . . Else statement to calculate student fees by testing the student status.
If the student is a graduate student, the student fees are calculated by multiplying the graduate fee times the number of units. If the student is not a graduate student, fees are calculated by multiplying the undergraduate fee times the number of units. Notice that a student cannot be both an undergraduate student and a graduate student, so either the statement following the Then keyword will be executed or the statement following the Else keyword will be executed.

Comparing to an Arithmetic Expression

An If statement can compare an arithmetic expression to a constant or other data type. For example, in Figure 5-35 the withdrawals from a bank account are compared to the value obtained by adding the current balance to deposits and then subtracting account charges.

In Figure 5-35, if the value in the decWithdrawals variable is greater than the current balance plus the deposits minus the account charges, the Text property of the lblAccountStatus Label object is set to Overdrawn. If the value in decWithdrawals is less than or equal to the value from the arithmetic expression, the message Balance is Positive is placed in the Text property of the lblAccountStatus Label object. Notice that the arithmetic expression is evaluated prior to the comparison. If the condition is true, the statement between the Then and Else keywords is executed. If the condition is false, the statement between the Else and End If keywords is executed.
Using the If . . . Then . . . ElseIf Statement

Complex logic problems might require a more complex structure than the If . . . Then . . . Else logic structure. For example, consider the following logical problem that must be solved in a computer program:

An online store charges a shipping amount based on the dollar amount of the order being shipped. The rules are: 1) If the order amount is above $500, the shipping cost is $30; 2) If the order amount is more than $400 and not greater than $500, the shipping cost is $25; 3) If the order amount is more than $200 and not greater than $400, the shipping cost is $20; 4) If the order amount is equal to or less than $200, the shipping cost is $15.

When one of the conditions is found to be true, the rest of the conditions are not tested because the correct condition has been found. To solve this problem, you should think this way:

1. If the order amount is greater than $500.00, then the shipping cost is $30.00 and no more processing must be done to determine the shipping cost.
2. If, however, the order amount is not greater than $500.00, I must check further to see if it is greater than $400.00 (400.01 through 500.00). If so, the shipping cost is $25.00.
3. If the order amount is not greater than $400.00, the next step is to check if it is greater than $200.00. Notice that if it is greater than $200.00 but not greater than $400.00, it must be in the range $201.00 to $400.00. If this is true, the shipping cost is $20.00.
4. If none of the above is true, then the order amount must be less than or equal to $200. In this case, the shipping cost is $15.00.

As you can see, a simple If . . . Then . . . Else statement could not solve this logic problem because the If . . . Then . . . Else structure tests only a single condition and specifies the processing based on whether the condition is true or false. For a problem where multiple conditions must be tested, the If . . . Then . . . ElseIf statement might be appropriate. The general format of the If . . . Then . . . ElseIf statement is shown in Figure 5-36.

```
105 If decOrderAmount > 500D Then
106     Statement(s) executed if condition is true
107 ElseIf decOrderAmount > 400D Then
108     Statement(s) executed if condition is true
109 ElseIf decOrderAmount > 200D Then
110     Statement(s) executed if condition is true
111 ElseIf decOrderAmount > 0D Then
112     Statement(s) executed if condition is true
113 End If
```

FIGURE 5-36
Once a condition in the code in Figure 5-36 is true, Visual Basic bypasses the rest of the ElseIf statements. For example, assume the order amount is $455. The first condition tests if the order amount is greater than 500. The first condition would test false because 455 is not greater than 500.

Next, the ElseIf entry will test if 455 is greater than 400. Because the value 455 is greater than 400, the condition is true and the statement(s) on line 108 will be executed. The remaining ElseIf statements will not be evaluated because the true condition has been found.

Separate If . . . Then statements are not used in the example in Figure 5-36 on the previous page because each condition would have to be tested even though a condition had already been found to be true. When using an If . . . Then . . . ElseIf statement, after the condition is found to be true, the remaining conditions are not tested, making the process faster and more efficient.

**Trailing Else Statements**

You may want to include a trailing Else statement at the end of an If . . . Then . . . ElseIf conditional statement to handle a condition that does not meet any of the conditions tested. In the example in Figure 5-37, the code is determining if the user is eligible for Social Security benefits. If the user’s age is greater than or equal to 65, the user receives full benefits. If the user’s age is between 0 and 65, the user is not eligible for benefits.

```vbnet
115    If intAge >= 65 Then
116        lblSocialSecurity.Text = "Full Benefits"
117    ElseIf intAge > 0 Then
118        lblSocialSecurity.Text = "Not Eligible for Benefits"
119    Else
120        lblSocialSecurity.Text = "Invalid Age"
121    End If
```

**FIGURE 5-37**

In Figure 5-37, the statement on line 120 that follows the trailing Else statement on line 119 is executed if the number in the intAge variable does not meet the conditions stated in the previous If statements. For example, if the intAge variable contains a negative value such as –12, the Text property of the lblSocialSecurity Label object will be set to “Invalid Age”.

**HEADS UP**

When testing conditions like those in the example in Figure 5-36 on the previous page, make sure you do not leave a number out of the range of numbers being tested. For example, if one ElseIf statement tests decOrderAmount > 400.00 and the next ElseIf statement tests < 400.00, the value 400.00 has not been tested and the program will not properly process the value 400.00.
Nested If Statements

At times, more than one decision has to be made to determine what processing must occur. For example, if one condition is true, a second condition may need to be tested before the correct code is executed. To test a second condition only after determining that a first condition is true (or false), you must place an If statement within another If statement. When you place one If statement within another If statement, the inner If statement is said to be nested within the outer If statement. The syntax of a nested If statement is shown in Figure 5-38.

In Figure 5-38, if the first condition tested is true, the statements following the keyword Then are executed. The statement to be executed when the first condition is true is another If statement (line 124) that tests the second condition. This second If statement is said to be a nested If statement or an inner If statement. If the second condition is true, the statement(s) on line 125 following the keyword Then for the first inner If statement are executed. If the second condition is not true, the statement(s) on line 127 following the keyword Else for the first inner If statement are executed. The End If entry (line 128) follows the first inner If statement, indicating the end of the effect of the first inner If statement.
If the first condition is not true, then the statements following the keyword Else on line 129 for the first If statement are executed. The statement to be executed when the first condition is not true is an If statement that tests the third condition (line 130). If the third condition is true, the statement(s) on line 131 following the Then keyword of the second inner If statement are executed. Finally, if the second inner If statement that tests the third condition is false, the statement(s) on line 133 are executed for the case when condition 1 is false and condition 3 is false.

To illustrate a nested If statement, assume a college has the following admissions policy: If an applying student has a GPA greater than 3.5 and an SAT score greater than 1000, then that student is granted admission. If an applying student has a GPA greater than 3.5 but an SAT score of 1000 or lower, the student is advised to retake the SAT exam. If an applying student has a GPA of 3.5 or lower but an SAT score greater than 1200, the student is granted a probationary admission, which means a 2.5 GPA must be achieved in the first semester of college. If an applying student has a GPA lower than 3.5 and an SAT score of 1200 or lower, the student is denied admission. The nested If statement to process this admission policy is shown in Figure 5-39.

```vbnet
140 If decGPA > 3.5D Then
141     If intSatScore > 1000 Then
142         lblAdmissionStatus.Text = "You have earned admission"
143     Else
144         lblAdmissionStatus.Text = "Retake the SAT exam"
145     End If
146     Else
147         If intSatScore > 1200 Then
148             lblAdmissionStatus.Text = "You have earned probationary admission"
149         Else
150             lblAdmissionStatus.Text = "You have been denied admission"
151         End If
152     End If
```

**FIGURE 5-39**

Notice in Figure 5-39 that the test for greater than 1000 on the SAT (line 141) must take place only after the test for a GPA greater than 3.5 (line 140), because the test for greater than 1000 is required only after it has been determined that the GPA is greater than 3.5. Therefore, a nested If statement is required. In addition, the test for greater than 1200 (line 147) should occur only after it has been determined that the GPA is less than 3.5. As you can see, you should use a nested If statement when a condition must be tested only after another condition has been tested.
Other Nested If Configurations

You can use nested If statements in a variety of forms. Assume, for example, that the admissions policy for a different school is as follows: If an applying student has a GPA greater than 3.5 and an SAT score greater than 1100, then that student is granted admission. If an applying student has a GPA greater than 3.5 but an SAT score of 1100 or lower, the student is advised to retake the SAT exam. If an applying student has a GPA of 3.5 or lower, the student is denied admission. The nested If statement in Figure 5-40 solves this logic problem.

```
154 If decGPA > 3.5D Then
155   If intSatScore > 1100 Then
156     lblAdmissionStatus.Text = "You have earned admission"
157   Else
158     lblAdmissionStatus.Text = "Retake the SAT exam"
159   End If
160 Else
161   lblAdmissionStatus.Text = "You have been denied admission"
162 End If
```

**FIGURE 5-40**

In Figure 5-40, if the GPA is greater than 3.5, then the first inner If statement on line 155 is executed to determine if the SAT score is greater than 1100. If so, the person has earned admission. If not, the person is advised to retake the SAT exam. If the GPA is not greater than 3.5, the student is denied admission. Notice that an If statement does not follow the Else keyword on line 160. An inner If statement need not follow both the If and the Else keywords.

Sometimes, after a condition is found to be true, a statement must be executed before the inner If statement is executed. For example, assume that if the GPA for a student is greater than 3.5, then the student should be informed that their GPA is acceptable for admission. The code in Figure 5-41 implements this condition.

```
164 If decGPA > 3.5D Then
165   lblGPAStatus.Text = "Your GPA is acceptable"
166   If intSatScore > 1100 Then
167     lblAdmissionStatus.Text = "You have earned admission"
168   Else
169     lblAdmissionStatus.Text = "Retake the SAT exam"
170   End If
171 Else
172   lblGPAStatus.Text = "Your GPA is not acceptable"
173   lblAdmissionStatus.Text = "You have been denied admission"
174 End If
```

**FIGURE 5-41**
In Figure 5-41 on the previous page, on line 165 the message “Your GPA is acceptable” is assigned to the Text property of the lblGPASStatus Label object prior to checking the SAT score. As you can see, after the first condition has been tested, one or more statements can be executed prior to executing the inner If statement. This holds true for the Else portion of the If statement as well.

Matching If, Else, and End If Entries

When you write a nested If statement, the inner If statement must be fully contained within the outer If statement. To accomplish this, you must ensure that each Else entry has a corresponding If entry, and an inner If statement must be terminated with an End If entry before either the Else entry or the End If entry for the outer If statement is encountered. If you code the statement incorrectly, one or more entries in the nested If statement will be identified with a blue squiggly line, indicating an error in the structure of the statement.

You also must place the correct statements with the correct If and Else statements within the nested If statement. For example, in Figure 5-42, the code is incorrect because the statement following the Else statements has been switched.

```
164 If decGPA > 3.5D Then
165   lblGPASStatus.Text = "Your GPA is acceptable"
166   If intSatScore > 1100 Then
167     lblAdmissionStatus.Text = "You have earned admission"
168  Else
169    lblAdmissionStatus.Text = "You have been denied admission"
170  End If
171  Else
172    lblGPASStatus.Text = "Your GPA is not acceptable"
173    lblAdmissionStatus.Text = "Retake the SAT exam"
174  End If
```

FIGURE 5-42

You must be precise when placing the executing statements in the nested If statement. It is easy to miscode a nested If statement.

Nesting Three or More Levels of If Statements

If statements are not limited to two levels of nesting. Three or more levels can be included in a nested If statement. When this is done, however, the nested If statement can become more difficult to understand and code. If more than two levels are required to solve a logic problem, great care must be taken to ensure errors such as the one shown in Figure 5-42 do not occur.
Testing the Status of a RadioButton Object in Code

In the Personal Trainer chapter project, which finds the number of calories burned, the user selects one RadioButton object to select the exercise type. The code must check each RadioButton to determine if that RadioButton has been selected by the user. When the user selects a radio button, the IsChecked property for that button is changed from False to True. In addition, the IsChecked property for other RadioButton objects is set to False. This IsChecked property can be tested in an If statement to determine if the RadioButton object has been selected.

To test the status of the IsChecked Property for a RadioButton object, the general statement shown in Figure 5-43 can be written.

```csharp
If radJogging.IsChecked Then
    Statement(s) to be executed if radio button is checked
End If
```

Figure 5-43

Notice in Figure 5-43 that the RadioButton property is not compared using a relational operator. Instead, when a property that can contain only True or False is tested, only the property must be specified in the If statement. When the property contains True, then the If statement is considered true, and when the property contains False, the If statement is considered false.

Testing RadioButtons with the If . . . Then . . . ElseIf Statement

When a program contains multiple RadioButton objects, only one of the radio buttons can be selected. The statement that can be used to check multiple radio buttons is the If . . . Then . . . ElseIf statement because once the checked radio button is detected, checking the remaining radio buttons is unnecessary.

In the Personal Trainer application, the user will click one of three radio buttons (Jogging, Biking, or Swimming) to select the type of exercise. To use an If . . . Then . . . ElseIf statement to check the status of the radio buttons, the most likely choice should be checked first. By doing this, the fewest number of tests will have to be performed. Therefore, the first If statement should test the status of the Jogging radio button (radJogging). If the radJogging button is checked, the number of calories should be set to the value in the decJoggingCost variable, which is 100. No further testing should be done (Figure 5-44 on the next page).
If the radJogging button is not checked, then the radBiking button should be tested. If it is checked, the calories per mile should be set to the value in the decBikingCost variable (42) and no further testing should be done. If the radBiking button is not checked, then the radSwimming button should be tested. If the other two buttons are not checked, then the radSwimming button must be checked because one of the three must be checked. The calories per mile will be set to the value in the decSwimmingCost variable (440).

**Block-Level Scope**

In Chapter 4 you learned that the scope of a variable is defined by where it is declared within a program. For example, if a variable is declared within an event handler, then only code within that event handler can reference the variable. Code in one event handler within a program cannot reference a variable declared in another event handler.

Within an event handler, an If . . . Then . . . Else statement (the code beginning with the If keyword and ending with the corresponding Else keyword, or the code beginning with the Else keyword and ending with the End If keyword) is considered a block of code. Variables can be declared within the block of code. When this occurs, the variable can be referenced only within the block of code where it is declared. For example, variables defined within an If . . . Then block of code fall out of scope (cannot be referenced) outside that block of code. To illustrate this concept, the code in Figure 5-45 shows a variable, intYears, declared within an If . . . Then block of code.

```vbnet
If intAge < 18 Then
    Dim intYears As Integer
    intYears = 18 - intAge
    lblMessage.Text = "You can vote in " & intYears & " years(s)."
Else
    lblMessage.Text = "You can vote!"
End If
```

**FIGURE 5-45**

Not For Sale
In Figure 5-45 on the previous page, on line 12 the variable intYears is declared as an Integer variable. On line 13, the variable is used in an arithmetic statement to receive the result of the calculation, 18 – intAge, which determines the number of years less than 18 that is stored in intAge. The result in intYears is concatenated with literals in the statement on line 14. The intYears variable can be referenced in any statements between the If keyword and the Else keyword. It cannot be referenced anywhere else in the program. Note that it cannot be referenced even in the Else portion of the If statement. When a statement referencing the intYears variable is written outside the area between the If keyword and the Else keyword, a compilation error will occur and the program will not be able to be compiled and executed.

Although the scope of the intYears variable in Figure 5-45 is between the If keyword on line 11 and the Else keyword on line 15, you should realize that the variable itself perseveres during the execution of the event handler procedure. Therefore, if the If statement in Figure 5-45 is executed a second time, the value in the intYears variable will be the same as when the If statement was completed the first time. To avoid unexpected results when the If statement is executed the second time, you should initialize block variables at the beginning of the block. In Figure 5-45, the statement on line 13 sets the value in the intYears variable immediately after the variable is declared, which is good programming technique.

**Using Logical Operators**

The If statements you have seen thus far test a single condition. In many cases, more than one condition must be true or one of several conditions must be true in order for the statements in the Then portion of the If . . . Then . . . Else statement to be executed. When more than one condition is included in an If . . . Then . . . Else statement, the conditions are called a **compound condition**. For example, consider the following business traveling rule: “If the flight costs less than $300.00 and the hotel is less than $120.00 per night, the business trip is approved.” In this case, both conditions (flight less than $300.00 and hotel less than $120.00 per night) must be true in order for the trip to be approved. If either condition is not true, then the business trip is not approved.

To create an If statement that processes the business traveling rule, you must use a **logical operator**. The most common set of logical operators are listed in Figure 5-46.

<table>
<thead>
<tr>
<th>Logical Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>And</strong></td>
<td>All conditions tested in the If statement must be true</td>
</tr>
<tr>
<td><strong>Or</strong></td>
<td>One condition tested in the If statement must be true</td>
</tr>
<tr>
<td><strong>Not</strong></td>
<td>Negates a condition</td>
</tr>
</tbody>
</table>

**FIGURE 5-46**

For the business traveling rule specified previously, you should use the And logical operator.
Using the And Logical Operator

The **And logical operator** allows you to combine two or more conditions into a compound condition that can be tested with an If statement. If any of the conditions stated in the compound condition are false, the compound condition is considered false and the statements following the Else portion of the If statement will be executed. The code in Figure 5-47 uses the And logical operator to implement the business traveling rule.

```
117 If decFlightCost < 300D And decHotelCost < 120D Then
118    lblTripMessage.Text = "Your business trip is approved"
119  Else
120     lblTripMessage.Text = "Your business trip is denied"
121  End If
```

**FIGURE 5-47**

In Figure 5-47, both conditions in the compound condition (flight cost less than 300 and hotel cost less than 120) must be true in order for the business trip to be approved. If one of the conditions is false, then the compound condition is considered false and the If statement would return a false indication. For example, if the flight cost is 300 or more, the trip will not be approved regardless of the hotel cost. Similarly, if the hotel cost is 120 or more, the trip will not be approved regardless of the flight cost. This process is illustrated in the diagram in Figure 5-48.

```
132  Dim decFlightCost As Decimal
133  Dim decHotelCost As Decimal
134  decFlightCost = 295D
135  decHotelCost = 125D
137  If decFlightCost < 300D And decHotelCost < 120D Then
```

**FIGURE 5-48**

In Figure 5-48, the flight cost is 295, so it is less than 300 and the first part of the compound condition is true. Following the And logical operator, the hotel cost (125) is not less than 120. Therefore, the second part of the compound condition is false. With the And logical operator, when either condition is false, the If statement considers the compound condition to be false. The result of the If statement in Figure 5-48 is that the compound condition is considered to be false.
Using the Not Logical Operator

When the **Not logical operator** is used to state conditions that are best expressed in a negative way, the condition is true if any tested condition is false. Even if four conditional statements are included in the compound condition, if one conditional statement in the compound condition is true, the entire statement is considered true.

As an example, assume a college has an acceptance policy that states each student must either have a minimum of a 3.5 grade point average (GPA) or at least a 1080 score on the SAT college entrance exam to be accepted for enrollment. If the student meets one or both conditions, the student would be accepted. The If statement in Figure 5-49, which uses the Not logical operator, will solve this problem.

Using the Or Logical Operator

When the **Or logical operator** is used to connect two or more conditions, the compound condition is true if any tested condition is true. Even if four conditional statements are included in the compound condition, if one conditional statement in the compound condition is true, the entire statement is considered true.

As an example, assume a college has an acceptance policy that states each student must either have a minimum of a 3.5 grade point average (GPA) or at least a 1080 score on the SAT college entrance exam to be accepted for enrollment. If the student meets one or both conditions, the student would be accepted. The If statement in Figure 5-49, which uses the Or logical operator, will solve this problem.

In Figure 5-49 if the GPA is 3.2, but the SAT score is 1130, the compound condition would be considered true because at least one of these conditions (intSATScore >= 1080) is true (Figure 5-50).

Using the Not Logical Operator

The **Not logical operator** allows you to state conditions that are best expressed in a negative way. In essence, the Not logical operator reverses the logical value of a condition on which it operates. For example, if a shoe store sells shoe sizes under size 14 from their showroom but requires special orders for larger sizes, the code could use the Not logical operator as shown in Figure 5-51 on the next page to negate the condition in the statement.
Making Decisions with Conditional Statements

The statement in Figure 5-51 works, but the use of the Not logical operator makes the If statement somewhat difficult to understand. Generally, a statement that avoids the Not logical operator is more easily understood. For example, the code in Figure 5-52 accomplishes the same task as the code in Figure 5-51, but is easier to understand.

```vbnet
155     If Not decShoeSize >= 14 Then
156         lblOrderPolicy.Text = "Showroom shoe style available"
157     Else
158         lblOrderPolicy.Text = "Special order needed"
159     End If
```

**FIGURE 5-51**

```
162     If decShoeSize < 14 Then
163         lblOrderPolicy.Text = "Showroom shoe style available"
164     Else
165         lblOrderPolicy.Text = "Special order needed"
166     End If
```

**FIGURE 5-52**

Other Logical Operators

The Visual Basic programming language provides three other lesser used logical operators. These are shown in the table in Figure 5-53.

<table>
<thead>
<tr>
<th>Logical Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xor</td>
<td>When one condition in the compound condition is true, but not both, the compound condition is true</td>
</tr>
<tr>
<td>AndAlso</td>
<td>As soon as a condition is found to be false, no further conditions are tested and the compound condition is false</td>
</tr>
<tr>
<td>OrElse</td>
<td>As soon as a condition is found to be true, no further conditions are tested and the compound condition is true</td>
</tr>
</tbody>
</table>

**FIGURE 5-53**
Order of Operations for Logical Operators

You can combine more than one logical operator in the same If . . . Then statement. In an If statement, arithmetic operators are evaluated first, relational operators are evaluated next, and logical operators are evaluated last. The order of operations for logical operators is shown in Figure 5-54.

<table>
<thead>
<tr>
<th>Logical Operator</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not</td>
<td>Highest Precedence</td>
</tr>
<tr>
<td>And, AndAlso</td>
<td>Next Precedence</td>
</tr>
<tr>
<td>Or, OrElse, Xor</td>
<td>Last Precedence</td>
</tr>
</tbody>
</table>

FIGURE 5-54

In most cases, if a developer uses multiple relational or logical operators in an If statement, the order of precedence should be established through the use of parentheses in order to clarify the sequence of evaluation. As in arithmetic expressions, conditional expressions within parentheses are evaluated before conditional expressions outside parentheses.

Select Case Statement

In some programming applications, different operations can occur based upon the value in a single field. For example, in Figure 5-55 on the next page, the user enters the number of the day in the week and the program displays the name of the day. The program must evaluate the number of the day value and display the correct name of the day.

In Figure 5-55, if the number of the day is 1, then the value Monday should be displayed. If the number of the day is 2, then Tuesday should be displayed, and so on. If the number of the day is 6 or 7, then the value Weekend should be displayed. If the user does not enter a value of 1 through 7, the user should be told to enter a value between 1 and 7.

To solve this problem, a series of If . . . Then . . . ElseIf statements could be used. An easier and clearer way to solve the problem, however, is to use the Select Case statement.

When using a Select Case statement, the value in a single field, such as the day number, is evaluated and a different action, such as displaying the name of the day, is taken based on the value in the field.
A general example of the Select Case statement is shown in Figure 5-56.

```vbnet
168 Select Case Test Expression
169 Case First Expression
170 Case Second Expression
171 Case Third Expression
172 Case Else
173 Statement(s) for when the Case Conditions do not match the
test expressions above
174 End Select
175
FIGURE 5-56
```
The coding for the Determine Day of Week application is shown in Figure 5-57.

```
13 Select Case intDayNumber
14     Case 1
15         lblDayOfWeek.Text = "Monday"
16     Case 2
17         lblDayOfWeek.Text = "Tuesday"
18     Case 3
19         lblDayOfWeek.Text = "Wednesday"
20     Case 4
21         lblDayOfWeek.Text = "Thursday"
22     Case 5
23         lblDayOfWeek.Text = "Friday"
24     Case 6
25         lblDayOfWeek.Text = "Weekend"
26     Case 7
27         lblDayOfWeek.Text = "Weekend"
28     Case Else
29         lblDayOfWeek.Text = "Enter 1 through 7"
30 End Select
```

**FIGURE 5-57**

The Select Case statement begins with the Select Case command. The test expression entry is used to specify the value or variable that contains the value to be tested in the Select Case statement. In Figure 5-57, the variable is intDayNumber. So, when the Select Case statement is executed, each of the cases will be compared to the value in the intDayNumber variable.

Each Case statement specifies the value for which the test expression is checked. For example, the first Case statement on line 14 in Figure 5-57 specifies the value 1. If the value in the variable intDayNumber is equal to 1, the statement(s) following the first Case statement up to the second Case statement (line 16) are executed. In Figure 5-57, the assignment statement on line 15 that sets the Text property of the lblDayOfWeek to Monday is executed if the value in intDayNumber is equal to 1. More than one statement can follow a Case statement.

If the expression following the first Case statement is not true, then the next Case statement is evaluated. In Figure 5-47, the Case statement on line 16 checks if the value in intDayNumber is equal to 2. If so, the Text property of the lblDayOfWeek is set to Tuesday. This process continues through the remainder of the Case statements.

The Case Else statement on line 28 is an optional entry that includes all conditions not specifically tested for in the other Case statements. In Figure 5-57, if the value in the intDayNumber variable is not equal to 1 through 7, then the statement following the Case Else statement is executed. While not required, good programming practice dictates that the Case Else statement should be used so that all cases are accounted for and the program performs a specific action regardless of the value found in the test expression.

The End Select statement is required to end the Select Case statement. When you enter the Select Case statement in Visual Studio 2010, IntelliSense automatically includes the End Select statement.
Select Case Test Expressions

The example in Figure 5-57 used an integer as the test expression value, but any data type can be used in the test expression. For example, the test expression in Figure 5-58 uses the Text property of the txtStudentMajor TextBox object as a String value.

```
217 Select Case txtStudentMajor.Text
218 Case "Accounting"
219   lblDepartment.Text = "Business"
220 Case "Marketing"
221   lblDepartment.Text = "Business"
222 Case "Electrical Engineering"
223   lblDepartment.Text = "Engineering"
224 Case "Biochemistry"
225   lblDepartment.Text = "Chemistry"
226 Case "Shakespearean Literature"
227   lblDepartment.Text = "English"
228 Case "Web Design and E-Commerce"
229   lblDepartment.Text = "CIS"
230 Case Else
231   lblDepartment.Text = "Other"
232 End Select
```

FIGURE 5-58

In Figure 5-58, the Select Case statement is used to test the value in the Text property of the txtStudentMajor TextBox object and move the corresponding department name to the Text property of the lblDepartment object. The Case statements specify the values to be tested in the text box. The use of a string for the Select Case statement works in the same manner as other data types.

Using Relational Operators in a Select Case Statement

You can use relational operators in a Select Case statement. You must, however, use the keyword Is with the relational operator. For example, in Figure 5-41 on page 30, an If . . . Then . . . ElseIf statement was used to determine the shipping cost. That same processing could be accomplished using a Select Case statement, as shown in Figure 5-59.

```
191 Select Case decOrderAmount
192 Case Is > 500D
193   decShippingCost = 30D
194 Case Is > 400D
195   decShippingCost = 25D
196 Case Is > 200D
197   decShippingCost = 20D
198 Case Is > 0D
199   decShippingCost = 15D
200 Case Else
201   decShippingCost = 0D
202 End Select
```

FIGURE 5-59
Using Ranges in Select Case Statements

Another way to specify values in a Select Case statement is to use ranges. In Figure 5-60, the Case statements illustrate testing for six different conditions.

As you can see, a range of values in a Case statement is specified by stating the beginning value, the word To, and then the ending value in the range. The Case statements will test the value in the intGradeLevel variable, and the appropriate statements will be executed.

You also can write Case statements with more than one distinct value being tested. In Figure 5-61, the Case statement tests the individual values of 1, 3, 8, 11, and 17 against the value specified in the intDepartmentNumber variable.

`Select Case intGradeLevel
    Case 1 To 3
        lblGradeLevelExam.Text = "Early elementary"
    Case 4 To 6
        lblGradeLevelExam.Text = "Late elementary"
    Case 7 To 8
        lblGradeLevelExam.Text = "Middle school"
    Case 9 To 10
        lblGradeLevelExam.Text = "Early high school"
    Case 11
        lblGradeLevelExam.Text = "Late high school"
    Case 12
        lblGradeLevelExam.Text = "Final exam"
    Case Else
        lblGradeLevelExam.Text = "Invalid grade level"
End Select`

Figure 5-60

As you can see, a range of values in a Case statement is specified by stating the beginning value, the word To, and then the ending value in the range. The Case statements will test the value in the intGradeLevel variable, and the appropriate statements will be executed.

You also can write Case statements with more than one distinct value being tested. In Figure 5-61, the Case statement tests the individual values of 1, 3, 8, 11, and 17 against the value specified in the intDepartmentNumber variable.

`Select Case intDepartmentNumber
    Case 1, 3, 8, 11, 17
End Select`

Figure 5-61

Notice in Figure 5-61 that each value in the Case statement is separated by a comma.

The code in Figure 5-62 shows a mixture of the two techniques, using both commas and a To statement.

`Select Case intDepartmentNumber
    Case 2, 4, 7, 12 To 16, 22
End Select`

Figure 5-62
Selecting Which Decision Structure to Use

In some instances, you might be faced with determining if you should use the Select Case statement or the If . . . Then . . . ElseIf statement to solve a problem. Generally, the Select Case statement is most useful when more than two or three values must be tested for a given variable. For example, in Figure 5-58 on page 45, six different values are checked in the Text property of the txtStudentMajor TextBox object. This is a perfect example of the use of the Select Case statement.

The If . . . Then . . . ElseIf statement is more flexible because more than one variable can be used in the comparison, and compound conditions with the And, Or, and Not logical operators can be used.

Code Snippets

Visual Basic includes a code library of almost five hundred pieces of code, called IntelliSense code snippets, that you can insert into an application. Each snippet consists of a complete programming task such as an If . . . Then . . . Else decision structure, sending an e-mail message, or drawing a circle. Inserting these commonly used pieces of code is an effective way to enhance productivity. You also can create your own snippets and add them to the library.

In addition to inserting snippets in your program, you can display a code snippet to ensure you understand the syntax and requirements for a given type of statement. To display and insert a code snippet for the If . . . Then . . . Else statement, you can complete the following steps:

**STEP 1** Right-click the line in the code editing window where you want to insert the snippet.

*Visual Studio displays a shortcut menu (Figure 5-63). It is important to right-click in the code editing window in the exact location where you want the code snippet to appear. If you right-click outside this location, the shortcut menu might list choices that are customized to that area of code and not include the code snippet for which you were searching. In addition, if you click in the wrong place, the snippet will be positioned in the incorrect location in your program.*
STEP 2  Click Insert Snippet on the shortcut menu.

*Visual Studio displays a menu of folders containing snippets (Figure 5-64). The code snippets in each folder correspond to their folder titles.*

![Figure 5-64](image)

STEP 3  Double-click the folder Code Patterns - If, For Each, Try Catch, Property, etc, which contains commonly used code such as the If . . . Then . . . Else statement.

*Visual Studio displays a menu of folders for code patterns (Figure 5-65).*

![Figure 5-65](image)

STEP 4  Double-click the Conditionals and Loops folder because an If . . . Then . . . Else statement is a conditional statement.

*Visual Studio displays the list of Conditionals and Loops code snippets (Figure 5-66). Some of these statements will be unfamiliar until you complete Chapter 6, but you can see that the list of code snippets includes a number of different types of If statements.*
STEP 5  Double-click the If . . . Else . . . End If Statement code snippet.

The If . . . Else . . . End If Statement code snippet is inserted into the code on the line selected in Step 1 (Figure 5-67). The highlighted text must be replaced by the condition(s) to be tested in the If statement. The code to be executed when the condition is true and the code to be executed when the condition is false must be added.

You must modify the code snippet shown in Figure 5-67 in order for the code to work properly. You may find that modifying the code in the snippet, particularly in a complicated code snippet, is more work than using IntelliSense to enter the statement.

Code snippets are also helpful for learning or reviewing the format and syntax of a statement. For example, if you wanted to review the syntax of an If . . . ElseIf . . . Else . . . End If statement, you could insert the statement into the code editing window and examine it. You could then either click the Undo button to remove the statement or you could comment out the snippet code and keep it for your review. In many cases of checking syntax, reviewing a snippet is faster and clearer than consulting Visual Basic help.
Deploying and Executing a Windows Phone Application

After designing the user interface and coding the Windows Phone application, it is time to deploy and test the application on the Phone 7 emulator. After the application has been tested, you can upload the completed app to the Windows Marketplace to share for free or to sell for profit. The Microsoft site for phone developers is located at http://create.msdn.com/. To deploy and execute a Windows Phone application to the phone emulator, follow these steps:

**STEP 1** Click the Start Debugging button on the toolbar.

*The emulator opens displaying an initial Internet Explorer title. After a few seconds the Personal Trainer Windows Phone application opens (Figure 5-68).*
**STEP 2** Tap the Miles Completed **TextBox** object on the emulator.

*The on-screen keyboard opens at the bottom of the emulator (Figure 5-69).*
**STEP 3** To view the numeric values on the keyboard, tap the &123 key. Tap the 5 key.

*The number 5 is entered into the Miles Completed TextBox object (Figure 5-70).*

![5 entered in Miles Completed TextBox object](image-url)
STEP 4  Tap the Biking RadioButton object. Tap the Calculate Button object.  

*The result is displayed in the lblResult Label object (Figure 5-71).*
STEP 5 Position the mouse pointer over the upper-right corner of the emulator, which displays the floating toolbar. You can use this toolbar to change the orientation or the emulator size.

*A floating toolbar appears on the upper-right side of the emulator (Figure 5-72).*

![Floating toolbar](image)

**FIGURE 5-72**

STEP 6 Click the X button on the floating toolbar to close the emulator. Click the OK button to stop the remote connection with the emulator.

*The emulator closes when the X button is clicked on the floating toolbar.*
Windows Phone Marketplace

After you have designed your mobile application, you can share your app through the Windows Phone Marketplace. To upload and sell your apps at the Windows Marketplace, first register as a member, which costs $99 per year at http://create.msdn.com (Figure 5-73). Paid membership enables you to publish and manage applications at Windows Phone Marketplace and to post to the App Hub forums to get support from the developer community. You can register without paying the membership fee and then post to the App Hub forums, but you won’t be able to submit applications to the App Hub. Microsoft provides a special free offer to students. You can register at Dreamspark.com and become a free member of the Windows Phone Marketplace. A student membership allows you to deploy and share any free apps that you create. To sell your apps, you must have a paid membership. Windows Marketplace has similar agreements with the iTunes/Apple and the Android stores for profit sharing with the developer which typically splits the profit of apps sales whereas the developer earns 70 percent and the application store earns 30 percent.

Program Design

As you have learned, the requirements document identifies the purpose of the program being developed; the application title; the procedures to be followed when using the program; any equations and calculations required in the program; any conditions within the program that must be tested; notes and restrictions that must be followed by the program; and any other comments that would be helpful to
understanding the problem. The requirements document for the Personal Trainer application is shown in Figure 5-74. The Use Case Definition document is shown in Figure 5-75 on the next page.

**REQUIREMENTS DOCUMENT**

**Date submitted:** January 29, 2014  
**Application title:** Personal Trainer App  
**Purpose:** This application calculates the number of calories burned by a specific type of exercise based on the distance completed.  
**Program Procedures:** The user should enter distance completed and select the type of exercise. The number of calories burned will be displayed.  
**Algorithms, Processing, and Conditions:**  
1. The user must be able to enter the number of miles completed.  
2. The user must be able to select one of three types of exercise: jogging, biking, or swimming.  
3. The user can initiate the calculation and display the number of calories burned.  
4. The application computes the number of calories based on the number of miles completed and the type of exercise. Jogging burns 100 calories per mile, biking burns 42 calories per mile, and swimming burns 440 calories per mile.  
5. The estimate calculation is: number of miles * number of calories per mile.  
**Notes and Restrictions:**  
1. If the user enters a negative number for the number of miles, the user should be advised and asked for a valid entry.  
**Comments:** None

**FIGURE 5-74**
USE CASE DEFINITION

1. The mobile app opens and displays the title Exercise Application Personal Trainer, a text box requesting the number of miles completed, radio buttons to select the type of exercise, and one button labeled Calculate.
2. The user enters the miles completed and selects one of the exercise types.
3. The user clicks the Calculate button.
4. The user will be warned if a negative number is entered.
5. The program displays the calories burned.
6. The user clicks the Close button on the upper-right corner of the phone interface.

FIGURE 5-75

Event Planning Document

You will recall that the event planning document consists of a table that specifies an object in the user interface that will cause an event, the action taken by the user to trigger the event, and the event processing that must occur. The event planning document for the Personal Trainer program is shown in Figure 5-76.

FIGURE 5-76

<table>
<thead>
<tr>
<th>Program Name:</th>
<th>Developer:</th>
<th>Object:</th>
<th>Date:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personal Trainer</td>
<td>Corinne Hoisington</td>
<td>MainPage.xaml</td>
<td>January 29, 2014</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OBJECT</th>
<th>EVENT TRIGGER</th>
<th>EVENT PROCESSING</th>
</tr>
</thead>
<tbody>
<tr>
<td>btnCalculate</td>
<td>Click</td>
<td>Convert data entered to numeric&lt;br&gt;Ensure data is greater than zero&lt;br&gt;Display error message if data is negative&lt;br&gt;Assign calories burned per mile based on exercise&lt;br&gt;Calculate calories burned (calories X miles)&lt;br&gt;Display calories burned</td>
</tr>
<tr>
<td>btnClear</td>
<td>Click</td>
<td>Clear input text box&lt;br&gt;Clear cost estimate&lt;br&gt;Set the Pine radio button to checked&lt;br&gt;Clear the Oak radio button&lt;br&gt;Clear the Cherry radio button&lt;br&gt;Set focus on input text box</td>
</tr>
<tr>
<td>frmWoodCabinets</td>
<td>Load</td>
<td>Set focus on input text box&lt;br&gt;Clear the placement zeros for cost</td>
</tr>
</tbody>
</table>
Designing and Coding the Program

After the events and the tasks within the events have been identified, the developer is ready to create the program. As you have learned, creating the program means designing the user interface and then entering Visual Basic statements to accomplish the tasks specified on the event planning document. As the developer enters the code, she also will implement the logic to carry out the required processing.

Note to the Learner

As you will recall, in the following activity you should complete the tasks within the specified steps. Each task is accompanied by a Hint Screen. The Hint Screen indicates where in the Visual Studio window you should perform the activity and reminds you of the method to use to create the user interface or enter code. If you need further help completing the step, refer to the figure number identified by the term ref: in the step.

Guided Program Development

To design the user interface for the Personal Trainer program and enter the code required to process each event in the program, complete the steps on the following pages.
Guided Program Development

Phase 1: Design the Form

1. **Create a Windows Phone Application**
   Open Visual Studio after installing the Visual Basic and Windows Phone Development Tools mentioned earlier in the chapter. Create a new Visual Basic Windows Phone Application project by completing the following: Click the New Project button on the Standard toolbar; select and expand Visual Basic in the left pane; select Silverlight For Windows Phone, in the center pane select Windows Phone Application; name the project Personal Trainer in the Name text box; then click the OK button in the New Project dialog box.

2. **Change the Layout**
   Place the mouse pointer between the right side of the scroll bar of the emulator Form object and the XAML code window. When a two-sided arrow appears, drag the two-sided arrow to the right until the emulator Form object is fully displayed.

3. **Change the TextBlock Objects**
   To change the TextBlock objects, click the first TextBlock object MY APPLICATION, and then change the name at the top of the Properties window to lblApplication. Scroll down the Properties window until the Text property is displayed, double-click the right column of the Text property, type Exercise Application, and then press the ENTER key. Click the second TextBlock object page name, and then change the name at the top of the Properties window to lblTitle. Scroll down the Properties window until the Text property is displayed, double-click the right column of the Text property, type Personal Trainer, and then press the ENTER key. Change the FontSize property to size 64.

(continues)
Guided Program Development  continued

- **Add a TextBlock** Drag a TextBlock object onto the emulator and name the TextBlock `lblDistance`. Set the Text property for the TextBlock object to `Miles Completed: ` and then set the font size to `Size 24`.

- **Add a TextBox** Drag a TextBox object onto the emulator Form object and name the TextBox `txtDistance`. Clear the Text property. Set the font size to `24`. Change the `VerticalContentAlignment` property to `Center`. Change the `Width` property to `100`. Position the label to resemble Figure 5-77 (ref: Figure 5-10).

Three TextBlock objects and a TextBox object occupy the top of the emulator (Figure 5-77).
Add Radio Buttons Place three RadioButton objects on the emulator. Name the first RadioButton radJogging and change its Text property to Jogging. Name the second RadioButton radBiking and change its Text property to Biking. Name the third RadioButton radSwimming and change its Text property to Swimming. Select the three RadioButtons (ref: Figure 5-14).

Set Radio Button Properties Click the radJogging RadioButton object and change its IsChecked property from False to True. Jogging is the most common type of exercise (ref: Figure 5-15).

The radio buttons are included on the emulator Form object (Figure 5-78). The radJogging radio button is selected because it is the most widely used exercise type.
Guided Program Development  continued

3

- **Add Calculate Button** Drag a Button object onto the emulator. Name the Button object `btnCalculate` and change its Content property to Calculate. Resize the button to fit the entire word Calculate.

- **Add a Result TextBlock** Drag one more TextBlock object below the Calculate button. Name the TextBlock `lblResult`, delete its Text property, and resize the TextBox object to a width of 215. Change the font size to 24 (ref: Figure 5-16).

**Add a Picture to the Emulator** Download the Exercise.jpg picture from [http://scsite.com/vb2010/ch5/images](http://scsite.com/vb2010/ch5/images). Drag an Image object to the bottom of the emulator Form object. Name the picture `picExercise`. Click to the right of the Source property. Click the ellipsis button to open the Choose image dialog box. Click the Add button and locate the Exercise.jpg image. Select the file and click Open. Click the OK button. Change the Height property to 200 and the Width property to 480. Move the image so it fits perfectly on the phone device (ref: Figure 5-19).

The mobile user interface is complete (Figure 5-79).
**Guided Program Development**

**Phase 2: Code the Application**

1. **Code the Comments** Double-click the btnCalculate Button object on the emulator Form object to open the code editing window and create the btnCalculate_Click event handler. Close the Toolbox. Click in front of the first words, Partial Public Class MainPage and press the ENTER key to create a blank line. Insert the first four standard comments. Insert the Option Strict On command at the beginning of the code to turn on strict type checking.

2. **Comment the btnCalculate_Click Event Handler** Click inside the btnCalculate event. Enter a comment to describe the purpose of the btnCalculate_Click event.

3. **Declare and Initialize the Variables** This application requires six variables: 1) decMiles: Holds the number of miles completed; 2) decCalories: Holds the number of calories burned based on the exercise type; 3) decCaloriesBurned: Is assigned to the final number of calories burned; 4) decJogging: Is assigned the value 100D; 5) decBiking: Is assigned the value 42D; 6) decSwimming: Is assigned the value 440D. The D represents a literal type. Declare and initialize these six variables.

4. **Convert the txtDistance to a Decimal** The miles completed value must be converted from a String value into a Decimal data type.
Guided Program Development  continued

- **Write the If Statement to Test for a Positive Number** The value must be checked to ensure it is a positive number. Write the If statement to check if the converted numeric value is greater than zero.

- **Write the If Statements to Determine Number of Calories Per Mile** When the value is greater than zero, the number of calories per mile is determined by checking the status of the RadioButton objects and placing the appropriate number of calories in the decCalories variable. Using the If . . . Then . . . ElseIf structure, write the statements to identify the checked radio button and place the appropriate number of calories in the decCalories variable (ref: Figure 5-43).

- **Calculate and Display the Cost Estimate** The next step is to calculate the total number of calories burned by the exercise and mileage completed by multiplying the value in the decCalories variable times the miles completed in the variable decMiles. Then you should display the total calories burned in the lblResult label. Write the statements to calculate and display the calories burned.

- **Display Message Box If Value Entered Is Not Greater Than Zero** After the processing is finished for the true portion of the If statements, the Else portion of the If statements must be written. Write the code to display the message box containing the error message when the value entered by the user is not greater than zero (ref: Figure 5-24).
The code for the click event of the Calculate button is completed (Figure 5-80).

RESULT OF STEP 1

- Run the Application After you have completed the code, you should run the application to ensure it works properly.

- Test the Application Test the application with the following data:
  1) Miles Completed: 10, exercise type: Biking
  2) Miles Completed: 2, exercise type: Swimming
  3) Miles Completed: 5, exercise type: Jogging
  4) Miles Completed: -2, exercise type: Biking
  6) Use other values to thoroughly test the program.

- Close the Program After testing the mobile application, close the window by clicking the Close button (X).
Code Listing

The complete code for the sample program is shown in Figure 5-81.

FIGURE 5-81
Summary

In this chapter you have learned to make decisions based on the user’s input.

The items listed in the table in Figure 5-82 include all the new Visual Studio and Visual Basic skills you have learned in this chapter.

### Visual Basic Skills

<table>
<thead>
<tr>
<th>Skill</th>
<th>Figure Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explore the Personal Trainer Windows Phone application</td>
<td>Figure 5-1</td>
</tr>
<tr>
<td>Add RadioButton objects</td>
<td>Figure 5-12</td>
</tr>
<tr>
<td>Display a MessageBox</td>
<td>Figure 5-20</td>
</tr>
<tr>
<td>Concatenate strings</td>
<td>Figure 5-25</td>
</tr>
<tr>
<td>Code an If...Then statement</td>
<td>Figure 5-26</td>
</tr>
<tr>
<td>Use relational operators</td>
<td>Figure 5-27</td>
</tr>
<tr>
<td>Enter an If...Then statement</td>
<td>Figure 5-28</td>
</tr>
<tr>
<td>Compare strings</td>
<td>Figure 5-32</td>
</tr>
<tr>
<td>Code If...Then...Else statements</td>
<td>Figure 5-33</td>
</tr>
<tr>
<td>Compare values using an arithmetic expression</td>
<td>Figure 5-35</td>
</tr>
<tr>
<td>Code an If...Then... ElseIf statement</td>
<td>Figure 5-36</td>
</tr>
<tr>
<td>Code a nested If statement</td>
<td>Figure 5-38</td>
</tr>
<tr>
<td>Test the status of a RadioButton object in code</td>
<td>Figure 5-43</td>
</tr>
<tr>
<td>Understand block-level scoping</td>
<td>Figure 5-45</td>
</tr>
<tr>
<td>Code logical operators</td>
<td>Figure 5-46</td>
</tr>
<tr>
<td>Code Select Case statements</td>
<td>Figure 5-56</td>
</tr>
<tr>
<td>Insert code snippets</td>
<td>Figure 5-63</td>
</tr>
</tbody>
</table>

**FIGURE 5-82**
Learn It Online

Start your browser and visit scsite.com/vb2010/ch5. Follow the instructions in the exercises below.

1. **Chapter Reinforcement TF, MC, SA** Click one of the Chapter Reinforcement links for Multiple Choice, True/False, or Short Answer below the Learn It Online heading.

2. **Practice Test** Click the Practice Test link below Chapter 5. Answer each question, enter your first and last name at the bottom of the page, and then click the Grade Test button. When the graded practice test is displayed on your screen, submit the graded practice test to your instructor. Continue to take the practice test until you are satisfied with your score.

3. **Crossword Puzzle Challenge** Click the Crossword Puzzle Challenge link below the Learn It Online heading. Read the instructions, and then click the Continue button. Work the crossword puzzle. When you are finished, click the Submit button. When the crossword puzzle is redisplayed, submit it to your instructor.

Knowledge Check

1. When you create a Windows Phone application, after you select New Project, which template do you select under Visual Basic?

2. On a Windows application you use a Label object to display text. Which object do you use to display text on a Windows Phone application?

3. Write an If . . . Then statement that tests if the value in the variable intTemp is between 32 and 95 degrees. If the number is in that range, set the Text property for the lblWarning Label object to “Normal Temperature”.

4. Write an If . . . Then . . . Else statement that assigns 30 to a variable named intMinutes if strRoadway is equal to “Interstate”. Otherwise, assign 60 to intMinutes.

5. List the three most common logical operators and explain their use.

6. Rewrite the following line of code without a Not logical operator but keeping the same logical processing:
   
   `If Not intHeight <= 75 Then`

7. The intent of the following statement is to check if the radDormStudent RadioButton object is checked. What is the error in the statement? Rewrite the statement so it is correct.
   
   `If radDormStudent = IsChecked Then`

8. The intent of following statement is to check if the value in the intGrade variable is less than 0 or greater than 100. What is the error in the statement? Rewrite the statement so it is correct.
   
   `If intGrade < 0 And intGrade > 100 Then`

9. Name the six relational operators and state the purpose of each operator.

10. Fix this statement.
    
    `If decWage > 7.35 and < 15.25 Then`
Knowledge Check continued

11. Why do most developers indent the code within a decision structure?

12. Write a statement that creates the dialog box shown in Figure 5-83 in a Windows Phone application. Use a single numerical value to create the button and picture icon.

13. If you sell a Windows Phone application at the Windows Phone Marketplace, what percentage of the profits do you earn?

14. What is the difference between the Or logical operator and the Xor logical operator?

15. Write a data validation statement that would check to ensure the value in the intAge variable is between 1 and 120. If the age is not valid, display an error message box stating that the age is not valid.

16. How many radio buttons can be selected at one time?

17. Using the concatenation operator (&), write a statement that would create the compound word teahouse from the following two strings: strFirst = “tea” and strSecond= “house”. Assign the compound word to the strCompound String variable.

18. Which property on a Button object displays text in a mobile application?

19. Write a Select Case statement using the fewest Case statements possible to display the number of days in each month. The user enters the number of the month, such as 8, which is converted to an integer and assigned to the intMonth variable. The Select Case statement should display a message box that states the number of days in the month, such as “31 Days”.

20. Which logical operator has the highest precedence in the order of operations?

Debugging Exercises

1. Explain how the two statements shown in Figure 5-84 are evaluated.

```
If strResponse = "Red" AndAlso strAnswer = "Green" Then
    If strResponse = "Red" AndAlso strAnswer = "Green" Then
```

(continues)
2. Identify the error in the code shown in Figure 5-85 and explain how to correct the code.

```vbnet
If dblCommission >= 2500 Then
    Dim intBonus As Integer
    intBonus = 500
Else
    intBonus = 0
End If
```

**FIGURE 5-85**

3. The Select Case statement shown in Figure 5-86 contains one or more errors. Identify the error(s) and rewrite the statements correctly.

```vbnet
Select Case intNumberOfSeats
    Case > 5000
        strVenueType = "Stadium"
    Case > 2000
        strVenueType = "Amphitheater"
    Case > 1000
        strVenueType = "Auditorium"
    Case > 200
        strVenueType = "Theater"
    Case > 0
        strVenueType = "Club"
    Else Case
        strVenueType = "Error"
Select End
```

**FIGURE 5-86**

4. The Select Case statement shown in Figure 5-87 contains one or more errors. Identify the error(s) and rewrite the statements correctly.

```vbnet
Select Case charFlightCode
    Case 'F', 'A'
        lblFare.Text = 'First Class'
    Case 'B', 'Q'
        lblFare.Text = 'Business Class'
    Case 'Y', 'S', 'M'
        lblFare.Text = 'Full Fare Economy'
    Case 'K', 'C'
        lblFare.Text = 'Preferred Economy'
    Case 'U', 'J', 'P', 'G'
        lblFare.Text = 'Economy'
    Else
        lblFare.Text = 'Unknown'
End Select
```

**FIGURE 5-87**
Debugging Exercises continued

5. The If...Then...Else statement shown in Figure 5-88 contains one or more errors. Identify the error(s) and rewrite the statements correctly.

```vbnet
If strShippingMethod = "Overnite" Then
    If strDeliveryTime = "Morning" Then
        decDeliveryCost = 29.00D
    Else
        decDeliveryCost = 24.00D
    End If
Else
    If strShippingMethod = "Two Days" Then
        decDeliveryCost = 14.00D
    Else
        decDeliveryCost = 4.00D
    End If
End If
```

FIGURE 5-88

Program Analysis

1. Write an If...Then...Else decision structure to compare the two numbers in the intPay1 and intPay2 variables. Display a message box stating intPay1 is greater than intPay2, or intPay1 is less than or equal to intPay2.

2. Write an If statement that displays the message box “Snow is possible” if the value in the variable decTemp is within the range 0 to 32.

3. Write an If...Then...Else statement that checks the value in the variable chrGender for the value M (Male) or F (Female) and assigns the information shown in Figure 5-89 to lblCollegeExpectation.Text based on the gender. If the variable chrGender contains a value other than M or F, assign the message “Invalid Gender” to lblCollegeExpectation.Text.

<table>
<thead>
<tr>
<th>Gender</th>
<th>College Expectation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>75% plan to graduate from college</td>
</tr>
<tr>
<td>Female</td>
<td>85% plan to graduate from college</td>
</tr>
</tbody>
</table>

FIGURE 5-89

(continues)
4. Write a Select Case statement that tests the user’s age in a variable named intAge and assigns the name of the favorite snack of that age group to the variable strSnack, according to the preferences shown in Figure 5-90.

<table>
<thead>
<tr>
<th>Age</th>
<th>Favorite Snack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Under age 7</td>
<td>Yogurt</td>
</tr>
<tr>
<td>Age 7 to 12</td>
<td>Potato Chips</td>
</tr>
<tr>
<td>Age 13 to 18</td>
<td>Chocolate</td>
</tr>
<tr>
<td>Over Age 18</td>
<td>Gum</td>
</tr>
</tbody>
</table>

FIGURE 5-90

5. Rewrite the Select Case statement shown in Figure 5-91 as an If . . . Then . . . Else statement.

```vba
If chrDepartment = "B", "b" Then
    strDept = "Baby / Infant Clothing"
ElseIf chrDepartment = "T", "t" Then
    strDept = "Technology"
End Select
```

FIGURE 5-91

6. Rewrite the If . . . Then . . . Else statement shown in Figure 5-92 as a Select Case statement.

```vba
If intGrade >= 9 And intGrade <= 12 Then
    lblSchool.Text = "High School"
ElseIf intGrade >= 7 Then
    lblSchool.Text = "Middle School"
ElseIf intGrade >= 1 Then
    lblSchool.Text = "Elementary School"
Else
    lblSchool.Text = "Invalid Grade"
End If
```

FIGURE 5-92

7. What is the output of the code shown in Figure 5-93 if the word Black is entered in the txtSkiSlope text box?
8. After the execution of the Select Case structure in Figure 5-94, what value will be found in the Text property of lblFemaleHeight if the user enters the number 74 into the txtEnterHeight text box? If the number 81 is entered? If the number 59 is entered?

```vbnet
Select Case txtSkiSlope Text
    Case "Green"
        MessageBox("Beginner Slope")
    Case "Blue"
        MessageBox("Intermediate Slope")
    Case "Black"
        MessageBox("Expert Slope")
    Case Else
        MessageBox("Invalid Entry")
End Select
```

FIGURE 5-94

9. In each of the following examples, is the condition True or False?
   a. “C” >= “C”
   b. “G” >= “g”
   c. “Amazed” < “Amaze”
   d. “Cool” < > “cool”
   e. “40” >= “Forty”
   f. (“Paris” < “Barcelona”) And (“Amsterdam” <= “Prague”)
   g. (“Ford” > “Chevrolet”) Or (“Toyota” < “Honda”)
   h. 3 ^ 2 <= 3 * 4
   i. Not (“CNN” >= “ABC”)
   j. Not (“Tim” > “Tom”) And Not (“Great” < > “great”)
Complete one or more of the following case programming assignments. Submit the program and materials you create to your instructor. The level of difficulty is indicated for each case programming assignment.

### PARKING TICKET FINES

Design a Windows Phone application and write the code that will execute according to the program requirements in Figure 5-95. The data to use is shown in Figure 5-96 on the next page. Before writing the code, create an event planning document for each event in the program, as shown in Figure 5-97. The completed Form object and other objects in the user interface are shown in Figure 5-98.

#### REQUIREMENTS DOCUMENT

- **Date submitted:** May 6, 2015
- **Application title:** Parking Ticket Fines
- **Purpose:** This Windows Phone application calculates a parking ticket fine.
- **Program Procedures:** The user selects the type of parking violation. The user then requests that the program calculate and display the parking ticket fine.
- **Algorithms, Processing, and Conditions:**
  1. The user selects the type of parking violation.
  2. If no RadioButton object is selected, an error message requests that the user select a violation.
  3. The fine is calculated based on the chart in Figure 5-96.
- **Notes and Restrictions:** None
- **Comments:** Obtain an image for this program from scsite.com/vb2010/ch5/images. The image file name is Parking.

#### FIGURE 5-95
Case Programming Assignments

Parking Ticket Fines (continued)

<table>
<thead>
<tr>
<th>Parking Violation</th>
<th>Fine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expired Meter</td>
<td>$35</td>
</tr>
<tr>
<td>No Parking Zone</td>
<td>$75</td>
</tr>
<tr>
<td>Blocking Driveway</td>
<td>$150</td>
</tr>
<tr>
<td>Illegal Handicap Parking</td>
<td>$500</td>
</tr>
</tbody>
</table>

FIGURE 5-96

USE CASE DEFINITION

1. The Windows Phone application opens.
2. The user selects the type of parking offense.
3. The user clicks the Display Fine button to display the parking ticket fine.

FIGURE 5-97
Case Programming Assignments

2. AMUSEMENT PARK

Design a Windows Phone application and write the code that will execute according to the program requirements in Figure 5-99 and the Use Case Definition in Figure 5-100 on the next page. Before writing the code, create an event planning document for each event in the program. The completed Form object and other objects in the user interface are shown in Figure 5-101 on the next page.

REQUIREMENTS DOCUMENT

Date submitted: April 15, 2014

Application titles: Buy Single Day Tickets at the Amusement Park
Amusement Park

Purpose: This Windows Phone application calculates the cost of the amusement park tickets.

Program Procedures:

1. The user enters the number of tickets.
2. The types of club memberships accepted are AAA (15% off full purchase cost), AARP (17% off full purchase cost), and Military ID (20% off full purchase price). The user can only select one type of membership.
3. The user must be able to initiate the calculation and display the total final cost of the tickets.

Notes and Restrictions:

If a negative number is entered for tickets, the user should be advised and asked for a valid entry.

Comments: Obtain an image for this program from scsite.com/vb2010/ch5/images. The image file name is Amusement.

FIGURE 5-99
Case Programming Assignments

Amusement Park (continued)

USE CASE DEFINITION

1. The Windows Phone application opens.
2. The user enters the number of single day tickets needed.
3. The user clicks the Calculate button.
4. The program displays the total cost of the tickets after the discount is applied.
5. The user is warned in a message box if a negative number is entered for the number of tickets.
Case Programming Assignments

3. PATIENT WEIGHT CONVERTER

Design a Windows Phone application and write the code that will execute according to the program requirements in Figure 5-102 and the Use Case Definition in Figure 5-103 on the next page. Before writing the code, create an event planning document for each event in the program. The completed Form object and other objects in the user interface are shown in Figure 5-104 on the next page.

REQUIREMENTS DOCUMENT

Date submitted: May 11, 2014
Application titles: Weight Conversion App
Patient Weight Conversion
Purpose: Nurses throughout the hospital need to convert weight to and from the metric system. This Windows Phone application converts the weight of the patient from pounds to kilograms and kilograms to pounds.
Program Procedures: The user enters the weight of the patient, selects the conversion type (pounds to kilograms or kilograms to pounds), and displays the converted weight of the patient.

Algorithms, Processing, and Conditions:
1. A user must be able to enter the weight of the patient in pounds or kilograms.
2. The user must be able to select the type of conversion: Pounds to kilograms or kilograms to pounds.
3. The user must be able to initiate the weight conversion and the display of the patient’s converted weight.
4. The conversion formulas are:
   kilograms = pounds / 2.2
   pounds = kilograms × 2.2

Notes and Restrictions:
1. If a negative number is entered for the weight, the user should be advised and asked for a valid entry.
2. If the value entered is greater than 500 for the conversion from pounds to kilograms or greater than 225 for the conversion from kilograms to pounds, the user should be advised and asked for a valid entry.
3. The default conversion choice should be pounds to kilograms.
4. The converted weight should be displayed with one digit to the right of the decimal point.

Comments: Obtain an image for this program from scsite.com/vb2010/ch5/images. The image file name is Scale.
Case Programming Assignments

Patient Weight Converter (continued)

USE CASE DEFINITION

1. The Windows Phone application opens.
2. The user enters the patient’s weight.
3. The user selects the conversion type (pounds to kilograms or kilograms to pounds).
4. The user clicks the Display button to display the converted weight value.
Case Programming Assignments

**HEALTH CLUB MEMBERSHIP**

Design a Windows Phone application and write the code that will execute according to the program requirements in Figure 5-105. Before designing the user interface, create a Use Case definition. Before writing the code, create an event planning document for each event in the program.

**REQUIREMENTS DOCUMENT**

**Date submitted:** April 22, 2014

**Application title:** Health Club Membership

**Purpose:** This Windows Phone application calculates the prepayment amount for a new member of a health club.

**Program Procedures:**

- The user should enter the name of the new member, the number of months the new user would like to prepay, and the type of membership. The health club prepayment cost will be computed and displayed for the entered number of months. The per month costs for the three types of membership are:
  - Single membership $38 per month
  - Family membership $58 per month
  - Senior membership $27 per month

**Algorithms, Processing, and Conditions:**

1. The user must enter the name of the new member, the type of membership, and the number of months the new member would like to prepay.
2. Based on the type of membership, the prepayment cost is calculated using the following formula: Number of prepay months × Cost per month.
3. The user must be able to initiate the calculation and display of the prepay amount for the health club membership.

**Notes and Restrictions:**

1. If the user enters a negative number for the number of months or if the user leaves the number of months input area blank, the user should be advised and asked for a valid entry.
2. The default membership type is a single membership.

**FIGURE 5-105**
Case Programming Assignments

5

SHIPPING COST

Design a Windows Phone application and write the code that will execute according to the program requirements in Figure 5-106. Before designing the user interface, create a Use Case definition. Before writing the code, create an event planning document for each event in the program. Figure 5-107 lists the weight and two-day shipping rates for packages.

---

REQUIREMENTS DOCUMENT

Date submitted: May 6, 2014

Application titles: Mailing Packages
                   Shipping Cost

Purpose: This Windows Phone application calculates the cost of shipping a package with two-day delivery.

Program Procedures: The user enters the weight of the package (in pounds) and selects the destination of the package. The application will determine the cost of shipping. The destination of the package can be the continental U.S., Hawaii, or Alaska. If the package is going to Hawaii, a 20% surcharge is added to the shipping cost. If the package is going to Alaska, a 26% surcharge is added to the shipping cost.

Algorithms, Processing, and Conditions:

1. The user must be able to enter the number of pounds the package weighs and indicate that the package is being mailed to the continental U.S., Hawaii, or Alaska.

2. The shipping costs are calculated based on the rates in the table in Figure 5-107 on the next page. A 20% surcharge is added if the shipping destination is Hawaii.
   A 26% surcharge is added if the shipping destination is Alaska.

3. The user must be able to initiate the calculation and the display of the shipping cost.

Notes and Restrictions:

The maximum weight for a package is 30 pounds. If the weight is greater than 30 pounds, or if the value entered is not greater than zero, the user should be advised and asked for a valid entry.
Case Programming Assignments

*Shipping Cost* (continued)

<table>
<thead>
<tr>
<th>For Weight Not Over (Pounds)</th>
<th>2-Day Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$3.69</td>
</tr>
<tr>
<td>4</td>
<td>$4.86</td>
</tr>
<tr>
<td>6</td>
<td>$5.63</td>
</tr>
<tr>
<td>8</td>
<td>$5.98</td>
</tr>
<tr>
<td>10</td>
<td>$6.28</td>
</tr>
<tr>
<td>30</td>
<td>$15.72</td>
</tr>
</tbody>
</table>

*FIGURE 5-107*
Case Programming Assignments

6 PAYROLL CALCULATOR

Design a Windows Phone application and write the code that will execute according to the program requirements in Figure 5-108. Before designing the user interface, create a Use Case definition. Before writing the code, create an event planning document for each event in the program.

REQUIREMENTS DOCUMENT

Date submitted: May 6, 2014
Application title: Payroll Calculator
Purpose: This Windows Phone application calculates the payroll for employees of the Food For All local grocery store.
Program Procedures: The user enters the employee’s name, hours worked, and pay per hour. If the employee works more than 40 hours per week, the grocery store pays time-and-a-half for overtime. The tax rate can be the single rate (18%) or at the family rate (15%). The application should compute and display the gross pay, the tax based on the single or family rate, and the net pay.
Algorithms, Processing, and Conditions: 1. The user must be able to enter the employee’s name, hours worked, and pay per hour.
2. The user must be able to indicate if the tax rate is at the single rate (18%) or the family rate (15%).
3. The user must be able to initiate the calculation and display of the gross pay, the tax amount based on the single or family rate, and the net pay.
Notes and Restrictions: 1. The minimum value for hours worked is 5 hours. The maximum for hours worked is 60. If the user enters an hours worked value not within the range, the user should be advised and asked for a valid entry.
2. The minimum pay per hour is $8.00. The maximum pay per hour is $40.00 per hour. If the user enters a pay per hour value not within the range, the user should be advised and asked for a valid entry.
Case Programming Assignments

TECHNOLOGY CONFERENCE REGISTRATION

Based on the case project shown in Figure 5-109, create a requirements document and a Use Case Definition document, and then design a Windows Phone application. Before writing the code, create an event planning document for each event in the program.

It is important that developers update their skills by attending developers’ conferences. The Dynamic International Management Consortium (DIMC) runs and manages the ADSE (Active Developers Skill Enhancement) Conference two times per year. To encourage companies to send multiple employees to the conference, the cost per attendee is determined based on the number of attending developers from a given company. The table below specifies the cost per attendee.

<table>
<thead>
<tr>
<th>Number of Conference Registrations per Company</th>
<th>Cost per Attendee</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$695</td>
</tr>
<tr>
<td>2-4</td>
<td>$545</td>
</tr>
<tr>
<td>5-8</td>
<td>$480</td>
</tr>
<tr>
<td>8 or more</td>
<td>$395</td>
</tr>
</tbody>
</table>

DIMC has requested that you develop a Windows application that can determine and display the total cost per company for developers attending the conference. DIMC has a conference policy that states if any member of a company has attended a previous DIMC conference, the company receives a 15% discount from the total cost of its employees who attend. The policy also states that no more than 16 people from a single company can attend the conference. DIMC has asked that you design the program so that the user must enter valid data.
Case Programming Assignments

8  CAR RENTAL

Based on the case project shown in Figure 5-110, create a requirements document and a Use Case Definition document, and then design a Windows Phone application. Before writing the code, create an event planning document for each event in the program.

The Adventure Car Rental Company has asked that you create a Windows Phone application for the rental of an adventure vehicle. The user selects the number of rental days, up to 7 days. The user also can select one of three types of vehicles. The types of vehicles and the cost per day for each vehicle is shown in the table below.

<table>
<thead>
<tr>
<th>Vehicle Model</th>
<th>Cost per Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jeep Wrangler</td>
<td>$55.00</td>
</tr>
<tr>
<td>Jeep Grand Cherokee</td>
<td>$85.00</td>
</tr>
<tr>
<td>Land Rover</td>
<td>$125.00</td>
</tr>
</tbody>
</table>

The customer has a choice of filling the gas tank themselves at the end of their use, or prepaying for a full tank of gas ($52 total). If the vehicle will be driven by more than one driver, a multiple driver cost of $22 per day is added to the cost of the vehicle. Adventure has asked that the Windows Phone application determine and display the total rental cost of the vehicle for the amount of time and the options chosen by the user. Adventure also has requested that you include all appropriate checking for invalid data entry by the user.
Case Programming Assignments

GOURMET FOOD TRUCK MOBILE APPLICATION

Based on the case project shown in Figure 5-111, create a requirements document and a Use Case Definition document, and then design a Windows Phone application. Before writing the code, create an event planning document for each event in the program.

Gourmet Food Trucks that serve high end fare are all the rage in big cities today. Each food truck would like to take orders on a Windows Phone 7 device that allows the user to select one of five gourmet specials of the day. (Create your own list using a search engine for ideas.) When displaying the specials, the application also displays prices for the specials and three types of bread choices at no additional cost. If you have two different sets of RadioButton objects, the GroupName property allows you to differentiate multiple groups. Allow the user to enter the state percentage of tax and compute the total cost of one single order.

FIGURE 5-111