71. A U.S. geological survey showed that Americans use an average of 1600 gallons of water per person per day, including industrial use. How many gallons of water are used each year in the United States? ***Note:*** The current population of the United States is about 315 million people.

72. According to FHA specifications, each room in a house should have a window area equal to at least 10% of the floor area of the room. What is the minimum window area for a 14-ft by 16-ft room?

Solutions to Practice Problem 3.1

1. 14. Multiplications are performed before additions. If the intent is for the addition to be performed first, the expression should be written (2 + 3)*4.

2. The first assignment statement assigns the value of the variable *var2* to the variable *var1*, whereas the second assignment statement assigns *var1*'s value to *var2*.

3.

	a	b	c
`Private Sub btnEvaluate_Click(...) Handles btnEvaluate.Click`			
`Dim a, b, c As Double`	0	0	0
`a = 3`	3	0	0
`b = 4`	3	4	0
`c = a + b`	3	4	7
`a = c * a`	21	4	7
`lstResults.Items.Add(a – b)`	21	4	7
`b = b * b`	21	16	7
`End Sub`			

Each time an assignment statement is executed, only one variable (the variable to the left of the equal sign) has its value changed.

4. Each of the three following statements increases the value of *var* by 5%.

```
var = var + (0.05 * var)
var = 1.05 * var
var += 0.05 * var
```

3.2 Strings

The most common types of data processed by Visual Basic are numbers and strings. Sentences, phrases, words, letters of the alphabet, names, telephone numbers, addresses, and social security numbers are all examples of strings. Formally, a **string literal** is a sequence of characters that is treated as a single item. String literals can be assigned to variables, displayed in text boxes and list boxes, and combined by an operation called concatenation (denoted by &).

Variables and Strings

VideoNote
Strings

A **string variable** is a name used to refer to a string. The allowable names of string variables are the same as those of numeric variables. The value of a string variable is assigned or altered with assignment statements and displayed in a list box like the value of a numeric variable. String variables are declared with statements of the form

```
Dim varName As String
```

Example 1 The following program shows how assignment statements and the Add method are used with strings. The string variable *president* is assigned a value by the third line, and this value is displayed by the sixth line. The quotation marks surrounding each string literal are not part of the literal and are not displayed by the Add method. (The form for this example contains a button and a list box.) ***Note:*** The Code Editor colors string literals red.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim president As String
  president = "George Washington"
  lstOutput.Items.Clear()
  lstOutput.Items.Add("president")
  lstOutput.Items.Add(president)
End Sub
```

[Run, and then click on the button. The following is displayed in the list box.]

```
president
George Washington
```

If *x*, *y*, . . . , *z* are characters and *strVar* is a string variable, then the statement

```
strVar = "xy...z"
```

assigns the string literal *xy* . . . *z* to the variable and the statement

```
lstBox.Items.Add("xy...z")
```

or

```
lstBox.Items.Add(strVar)
```

displays the string *xy* . . . *z* in a list box. If *strVar2* is another string variable, then the statement

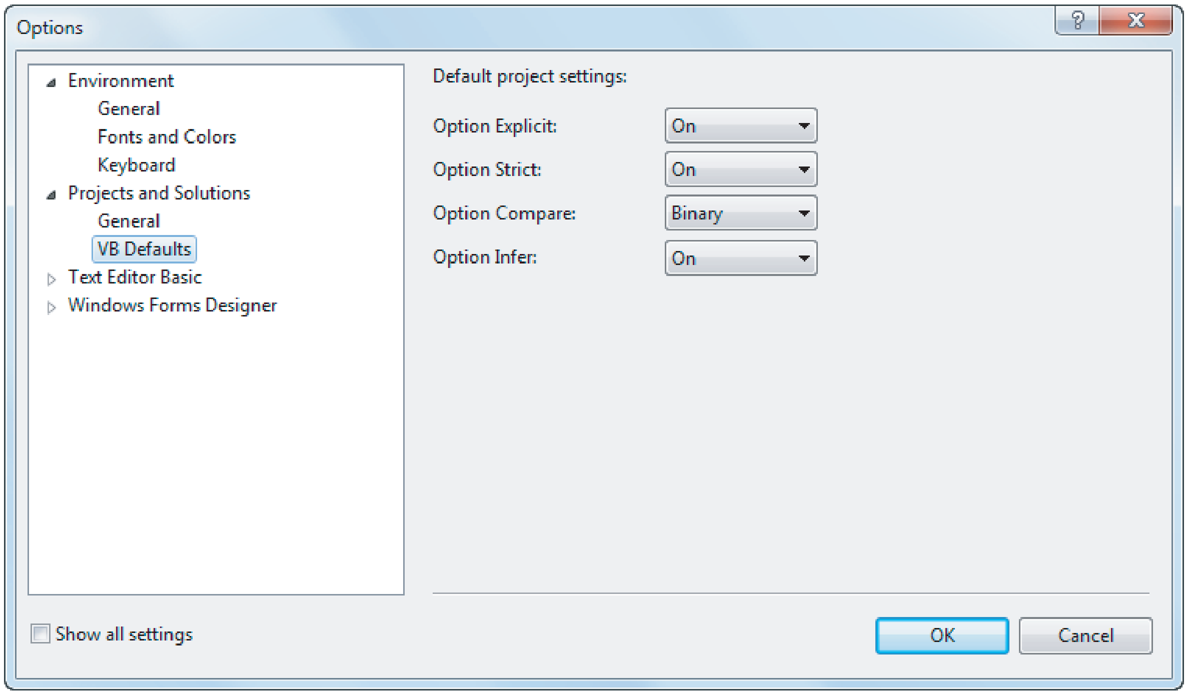
```
strVar2 = strVar
```

assigns the value of the variable *strVar* to the variable *strVar2*. (The value of *strVar* will remain the same.) String literals used in assignment or lstBox.Items.Add statements must be surrounded by quotation marks, but string variables are never surrounded by quotation marks.

Option Explicit and Option Strict

Option Explicit and **Option Strict** both affect programming. Throughout this book, we assume that both options are in effect. Having them enabled is considered good programming practice. Option Explicit requires that all variables be declared with Dim statements. The disabling of this option can lead to errors resulting from the misspelling of names of variables. Option Strict requires explicit conversions in most cases where a value or variable of one type is assigned to a variable of another type. The absence of this option can lead to data loss.

Visual Basic provides a way to enforce Option Explicit and Option Strict for all programs you create. Click on *Options* in the menu bar's *Tools* menu to open the Options dialog box. In the left pane, click on the symbol (⊞ or ▷) to the left of Projects and Solutions to expand that entry. Then click on the subentry VB Defaults. Four default project settings will appear on the right. (See Fig. 3.1.) If the settings for Option Explicit and Object Strict are not already set to On, change them to On. ***Note:*** Option Infer is discussed in Chapter 6.



Default project settings:

Option Explicit: On
Option Strict: On
Option Compare: Binary
Option Infer: On

FIGURE 3.1 **Option default project settings.**

Using Text Boxes for Input and Output

The content of a text box is always a string. Therefore, statements such as

```
strVar = txtBox.Text
```

and

```
txtBox.Text = strVar
```

can be used to assign the contents of the text box to the string variable *strVar* and vice versa.

Numbers typed into text boxes are stored as strings. With Option Strict set to On, such strings must be explicitly converted to numeric values before they can be assigned to numeric variables or used in numeric expressions. The functions CDbl and CInt convert strings representing numbers into numbers of type Double and Integer, respectively. Going in the other direction, the function CStr converts a number into a string representation of the number. Therefore, statements such as

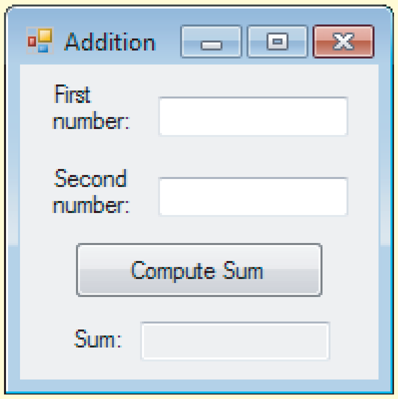
```
dblVar = CDbl(txtBox.Text)
```

and

```
txtBox.Text = CStr(dblVar)
```

can be used to assign the contents of a text box to the double variable *dblVar* and vice versa. CDbl, CInt, and CStr, which stand for "convert to Double," "convert to Integer," and "convert to String," are referred to as **data conversion** or **typecasting functions**.

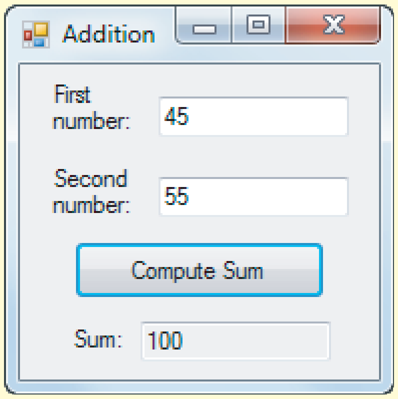
Example 2 The following program adds two numbers supplied by the user.



OBJECT	PROPERTY	SETTING
frmAdd	Text	Addition
lblFirstNum	AutoSize	False
	Text	First number:
txtFirstNum		
lblSecondNum	AutoSize	False
	Text	Second number:
txtSecondNum		
btnCompute	Text	Compute Sum
lblSum	Text	Sum:
txtSum	ReadOnly	True

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  Dim num1, num2, sum As Double
  num1 = CDbl(txtFirstNum.Text)
  num2 = CDbl(txtSecondNum.Text)
  sum = num1 + num2
  txtSum.Text = CStr(sum)
End Sub
```

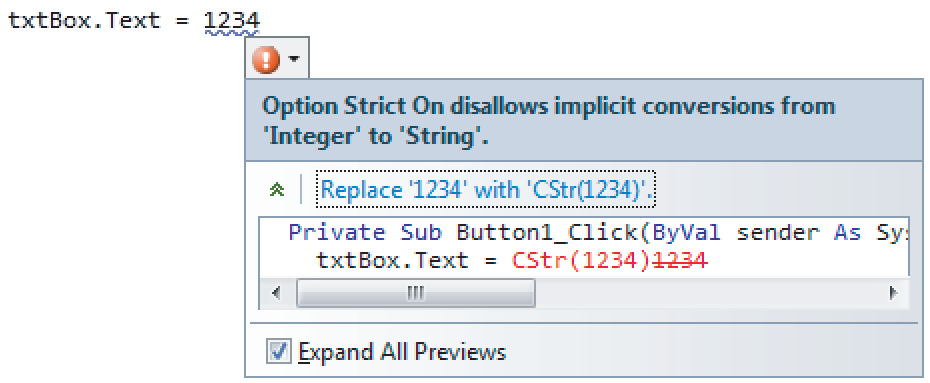
[Run, type 45 into the first text box, type 55 into the second text box, and click on the button.]



First number: 45
Second number: 55
Sum: 100

Auto Correction

The **Auto Correction** feature of IntelliSense suggests corrections when errors occur and allows you to select a correction to be applied to the code. When an invalid statement is entered, a blue squiggly error line appears under the incorrect part of the statement. If the squiggly line has a short red line segment at its right end, the Auto Correction feature is available for the error. When you hover the cursor over the squiggly line, a small Error Correction Options box (!▾) appears. Clicking on the small box produces an Auto Correction helper box that describes the error and makes a suggestion for fixing it. Figure 3.2 shows a typical Auto Correction helper box for a data-type-conversion error. If you click on the line beginning "Replace," the change will be made for you.



```
txtBox.Text = 1234
```

Option Strict On disallows implicit conversions from 'Integer' to 'String'.

Replace '1234' with 'CStr(1234)'.

```
Private Sub Button1_Click(ByVal sender As Sy
  txtBox.Text = CStr(1234)~~1234~~
```

FIGURE 3.2 **An Auto Correction helper box.**

Concatenation

Two strings can be combined to form a new string consisting of the strings joined together. The joining operation is called **concatenation** and is represented by an ampersand (&). For instance, “good” & “bye” is “goodbye”. A combination of strings and ampersands that can be evaluated to form a string is called a **string expression**. The assignment statement and the Add method evaluate expressions before assigning them or displaying them.

Example 3 The following program illustrates concatenation. (The form for this example contains a button and a text box.) Notice the space at the end of the string assigned to *quote1*. If that space weren’t present, then the statement that assigns a value to *quote* would have to be

```
quote = quote1 & " " & quote2.
```

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim quote1, quote2, quote As String
  quote1 = "The ballgame isn't over, "
  quote2 = "until it's over."
  quote = quote1 & quote2
  txtOutput.Text = quote & "   Yogi Berra"
End Sub
```

[Run, and then click on the button. The following is displayed in the text box.]

```
The ball game isn't over, until it's over.   Yogi Berra
```

Visual Basic also allows strings to be concatenated with numbers and allows numbers to be concatenated with numbers. In each case, the result is a string.

Example 4 The following program concatenates a string with a number. Notice that a space was inserted after the word “has” and before the word “keys.” (The form for this example contains a button and a text box.)

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim str As String, numOfKeys As Integer
  str = "The piano keyboard has "
  numOfKeys = 88
  txtOutput.Text = str & numOfKeys & " keys."
End Sub
```

[Run, and then click on the button. The following is displayed in the text box.]

```
The piano keyboard has 88 keys.
```

The statement

```
strVar = strVar & strVar2
```

will append the value of *strVar2* to the end of the current value of *strVar*. The same result can be accomplished with the statement

```
strVar &= strVar2
```

String Properties and Methods: Length Property and ToUpper, ToLower, Trim, IndexOf, and Substring Methods

We have seen that controls, such as text and list boxes, have properties and methods. A control placed on a form is an example of an object. A string is also an object, and, like a control, has both properties and methods that are specified by following the string with a period and the name of the property or method. The Length property gives the number of characters in a string. The ToUpper and ToLower methods convert a string to uppercase and lowercase characters. The Trim method deletes all leading and trailing spaces from a string. The Substring method extracts a sequence of consecutive characters from a string. The IndexOf method searches for the first occurrence of one string in another and gives the position at which the first occurrence is found.

If *str* is a string, then

```
str.Length
```

is the number of characters in the string,

```
str.ToUpper
```

is the string with all its letters capitalized,

```
str.ToLower
```

is the string with all its letters in lowercase, and

```
str.Trim
```

is the string with all spaces removed from the front and back of the string. For instance,

```
"Visual".Length is 6.                     "Visual".ToUpper is VISUAL.
"123 Hike".Length is 8.                   "123 Hike".ToLower is 123 hike.
"a" & "  bcd  ".Trim & "efg" is abcdefg.
```

In Visual Basic, the **position** of a character in a string is identified with one of the numbers 0, 1, 2, 3, ... (In this textbook we will see several instances of enumeration beginning with 0 instead of 1.) A **substring** of a string is a sequence of consecutive characters from the string. For instance, consider the string "Just a moment". The substrings "Jus", "mom", and "nt" begin at positions 0, 7, and 11, respectively.

If *str* is a string, then

```
str.Substring(m, n)
```

is the substring of *str* consisting of *n* characters beginning with the character in position *m* of *str*. If the comma and the number *n* are omitted, then the substring starts at position *m* and continues until the end of *str*. The value of

```
str.IndexOf(str2)
```

is −1 if *str2* is not a substring of *str*; otherwise it is the beginning position of the first occurrence of *str2* in *str*. Some examples using these two methods are as follows:

```
"fanatic".Substring(0, 3) is "fan".        "fanatic".IndexOf("ati") is 3.
"fanatic".Substring(4, 2) is "ti".         "fanatic".IndexOf("a") is 1.
"fanatic".Substring(4) is "tic".           "fanatic".IndexOf("nt") is −1.
```

The IndexOf method has a useful extension. The value of `str.IndexOf(str2,n)`, where *n* is an integer, is the position of the first occurrence of *str2* in *str* in position *n* or greater. For instance, the value of `"Mississippi".IndexOf("ss",3)` is 5.

Like the numeric functions discussed before, string properties and methods also can be applied to variables and expressions.

Example 5 The following program uses variables and expressions with the property and methods just discussed.

```
Private Sub btnEvaluate_Click(...) Handles btnEvaluate.Click
  Dim str1, str2, str3 As String
  str1 = "Quick as "
  str2 = "a wink"
  lstResults.Items.Clear()
  lstResults.Items.Add(str1.Substring(0, 7))
  lstResults.Items.Add(str1.IndexOf("c"))
  lstResults.Items.Add(str1.Substring(0, 3))
  lstResults.Items.Add((str1 & str2).Substring(6, 6))
  lstResults.Items.Add((str1 & str2).ToUpper)
  lstResults.Items.Add(str1.Trim & str2)
  str3 = str2.Substring(str2.Length - 4)
  lstResults.Items.Add("The average " & str3 & " lasts .1 second.")
End Sub
```

[Run, and then click on the button. The following is displayed in the list box.]

```
Quick a
3
Qui
as a w
QUICK AS A WINK
Quick asa wink
The average wink lasts .1 second.
```

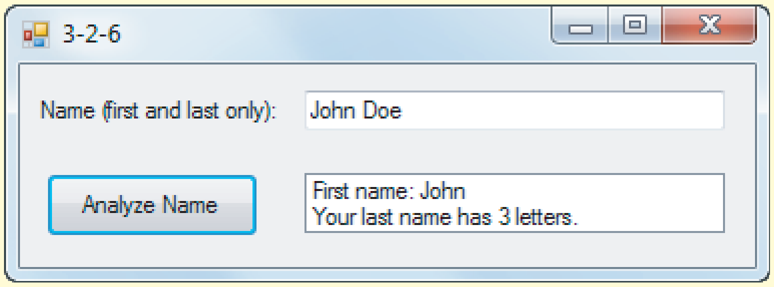
Note: In Example 5, *c* is in the third position of *str1*, and there are three characters of *str1* to the left of *c*. In general, there are *n* characters to the left of the character in position *n*. This fact is used in Example 6.

Example 6 The following program parses a name. The fifth line locates the position, call it *n*, of the space separating the two names. The first name will contain *n* characters, and the last name will consist of all characters to the right of the *n*th character.

```
Private Sub btnAnalyze_Click(...) Handles btnAnalyze.Click
  Dim fullName, firstName, lastName As String
  Dim m, n As Integer
  fullName = txtName.Text
```

```
    n = fullName.IndexOf(" ")
    firstName = fullName.Substring(0, n)
    lastName = fullName.Substring(n + 1)
    m = lastName.Length
    lstResults.Items.Clear()
    lstResults.Items.Add("First name: " & firstName)
    lstResults.Items.Add("Your last name has " & m & " letters.")
End Sub
```

[Run, type "John Doe" into the text box, and then click on the button.]



```
First name: John
Your last name has 3 letters.
```

The Empty String

The string `""`, which contains no characters, is called the **empty string** or the **zero-length string**. It is different from `" "`, the string consisting of a single space.

The statement `lstBox.Items.Add("")` inserts a blank line into the list box. The contents of a text box can be cleared with either the statement

```
txtBox.Clear()
```

or the statement

```
txtBox.Text = ""
```

Initial Value of a String

When a string variable is declared with a Dim statement, it has the keyword Nothing as its default value. To specify a different initial value, follow the declaration statement with an equal sign followed by the initial value. For instance, the statement

```
Dim pres As String = "Adams"
```

declares the variable *pres* to be of type String and assigns it the initial value "Adams".

An error occurs whenever an attempt is made to access a property or method for a string variable having the value Nothing or to display it in a list box. Therefore, unless a string variable is guaranteed to be assigned a value before being used, you should initialize it—even if you just assign the empty string to it.

Widening and Narrowing

VideoNote
Widening and narrowing, scope

The assignment of a value or variable of type Double to a variable of type Integer is called **narrowing** because the possible values of an Integer variable are a subset of the possible values of a Double variable. For the same reason, assigning in the reverse direction is called **widening**. Option Strict requires the use of a conversion function when narrowing, but allows widening without a conversion function. Specifically, a widening statement of the form

```
dblVar = intVar
```

is valid. However, a narrowing statement of the form

```
intVar = dblVar
```

is not valid. It must be replaced with

```
intVar = CInt(dblVar)
```

Great care must be taken when computing with Integer variables. For instance, the value of an expression involving division or exponentiation has type Double and therefore cannot be assigned to an Integer variable without explicit conversion even if the value is a whole number. For instance, Option Strict makes each of the following two assignment statements invalid.

```
Dim m As Integer
m = 2 ^ 3
m = 6 / 2
```

In order to avoid such errors, we primarily use variables of type Integer for counting or identifying positions.

Internal Documentation

Program documentation is the inclusion of **comments** that specify the intent of the program, the purpose of the variables, and the tasks performed by individual portions of the program. To create a comment statement, begin the line with an apostrophe. Such a statement appears green on the screen and is completely ignored when the program is executed. Comments are sometimes called **remarks**. A line of code can be documented by adding an apostrophe, followed by the desired information, after the end of the line. The *Comment Out* button ( ) and the *Uncomment* button ( ) on the Toolbar can be used to comment and uncomment selected blocks of code.

Example 7 The following rewrite of Example 6 uses internal documentation. The first comment describes the entire program, the comment in line 5 gives the meaning of the variable, and the final comment describes the purpose of the three lines that follow it.

```
Private Sub btnAnalyze_Click(...) Handles btnAnalyze.Click
  'Determine a person's first name and the length of the second name
  Dim fullName, firstName, lastName As String
  Dim m As Integer
  Dim n As Integer   'location of the space separating the two names
  fullName = txtName.Text
  n = fullName.IndexOf(" ")
  firstName = fullName.Substring(0, n)
  lastName = fullName.Substring(n + 1)
  m = lastName.Length
  'Display the desired information in a list box
  lstResults.Items.Clear()
  lstResults.Items.Add("First name: " & firstName)
  lstResults.Items.Add("Your last name has " & m & " letters.")
End Sub
```

Some of the benefits of documentation are as follows:

1. Other people can easily understand the program.
2. You can understand the program when you read it later.
3. Long programs are easier to read because the purposes of individual pieces can be determined at a glance.

Good programming practice dictates that programmers document their code at the same time that they are writing it. In fact, many software companies require a certain level of documentation before they release a version, and some judge a programmer's performance on how well their code is documented.

Line Continuation

Thousands of characters can be typed in a line of code. If you use a statement with more characters than can fit in the window, Visual Basic scrolls the Code Editor toward the right as needed. However, most programmers prefer having lines that are no longer than the width of the Code Editor. A long statement can be split across two or more lines by ending each line (except the last) with an underscore character (_) preceded by a space. For instance, the line

```
Dim quotation As String = "Good code is its own best documentation."
```

can be written as

```
Dim quotation As String = "Good code is its own " & _
                          "best documentation."
```

A new-to-VB2010 feature called **implicit line continuation** allows underscore characters to be omitted from the end of a line that obviously has a continuation—for instance, a line that ends with an ampersand, an arithmetic operator, or a comma. We use this feature throughout this textbook. For example, the line above will be written

```
Dim quotation As String = "Good code is its own " &
                          "best documentation."
```

Line continuation, with or without an underscore character, cannot be used inside a pair of quotation marks. Whenever you want to display a literal string on two lines of the screen, you must first break it into two shorter strings joined with an ampersand. IntelliSense is extremely dependable in letting you know if you have broken a line improperly.

Line continuation, with or without an underscore character, does not work with comment statements. Therefore, each line of a comment statement must begin with its own apostrophe.

Scope of a Variable

When a variable is declared in an event procedure with a Dim statement, a portion of memory is set aside to hold the value of the variable. As soon as the End Sub statement for the procedure is reached, the memory location is freed up; that is, the variable ceases to exist. The variable is said to be **local** to the procedure or to have **local scope**. In general, the **scope** of a variable is the portion of the program that can refer to it.

When variables of the same name are declared with Dim statements in two different event procedures, Visual Basic gives them separate memory locations and treats them as two different variables. A value assigned to a variable in one procedure will not affect the value of the identically named variable in the other procedure.

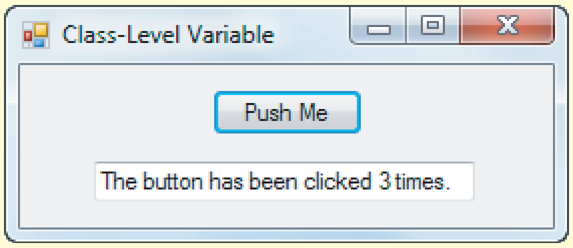
Visual Basic provides a way to make a variable recognized by every procedure in a form's code. Such a variable is called a **class-level variable** and is said to have **class-level scope**. The

Dim statement for a class-level variable can be placed anywhere between the statements `Public Class` *`formName`* and `End Class`, provided that the Dim statement is not inside an event procedure. Normally, we place the `Dim` statement just after the `Public Class` *`formName`* statement. (We refer to this region as the **Declarations section** of the Code Editor.) When a class-level variable has its value changed by a procedure, the value persists even after the procedure has finished executing. If an event procedure declares a local variable with the same name as a class-level variable, then the name refers to the local variable for code inside the procedure.

Example 8 The following program uses a class-level variable to keep track of the number of times a button has been clicked.

```
Public Class frmCount
  Dim numTimes As Integer = 0   'Class-level variable
  Private Sub btnPushMe_Click(...) Handles btnPushMe.Click
    numTimes += 1
    txtOutput.Text = "The button has been clicked " &
                     numTimes & " times."
  End Sub
End Class
```

[Run, and click on the button three times.]



```
The button has been clicked 3 times.
```

Comments

1. From the Code Editor, you can determine the type of a variable by letting the mouse pointer hover over the variable name until a tooltip giving the type appears. This feature of IntelliSense is called **Quick Info**.
2. Variable names should describe the role of the variable. Also, some programmers use a prefix, such as *dbl* or *str*, to identify the type of a variable. For example, they would use names like *dblInterestRate* and *strFirstName*. This naming convention is not needed in Visual Basic for the reason mentioned in Comment 1, and is no longer recommended by Microsoft.
3. The functions CInt and CDbl are user friendly. If the user types a number into a text box and precedes it with a dollar sign or inserts commas as separators, the values of CInt(txtBox.Text) and CDbl(txtBox.Text) will be the number with the dollar sign and/or commas removed.
4. The Trim method is useful when reading data from a text box. Sometimes users type spaces at the end of the input. Unless the spaces are removed, they can cause havoc elsewhere in the program.
5. There are several alternatives to CStr for casting a value to a string value. For instance, the statement

   ```
   strVar = CStr(dblVar)
   ```

can be replaced with any of the following statements:

```
strVar = CType(dblVar, String)
strVar = Convert.ToString(dblVar)
strVar = dblVar.ToString
```

Some alternatives to the use of CInt and CDbl are

```
dblVar = CType(strVar, Double)
intVar = CType(strVar, Integer)
intVar = CType(dblVar, Integer)
intVar = Integer.Parse(strVar)
dblVar = Double.Parse(strVar)
dblVar = Convert.ToDouble(strVar)
intVar = Convert.ToInt32(strVar)
intVar = Convert.ToInt32(dblVar)
```

These alternatives are common to all the Visual Studio languages and therefore are preferred by advanced programmers. We have decided to use CStr, CDbl, and CInt for the following two reasons:

(a) These functions make statements less cluttered, and therefore easier for beginning programmers to read.

(b) When an incorrect conversion is detected by the Code Editor, the Auto Correction helper box recommends and implements the use of the CStr, CDbl, and CInt functions.

Practice Problems 3.2

1. What is the value of "Computer".IndexOf("E")?
2. What is the difference in the output produced by the following two statements? Why is CStr used in the first statement, but not in the second?

```
txtBox.Text = CStr(8 + 8)
txtBox.Text = 8 & 8
```

3. Give an example of a prohibited statement that invokes an Auto Correction helper box with the heading "Option Strict On disallows implicit conversion from 'String' to 'Double'." Also, give the suggestion for fixing the error.

EXERCISES 3.2

In Exercises 1 through 28, determine the output displayed in the text box or list box by the lines of code.

1.
```
txtBox.Text = "Visual Basic"
```

2.
```
lstBox.Items.Add("Hello")
```

3.
```
Dim var As String
var = "Ernie"
lstBox.Items.Add(var)
```

4.
```
Dim var As String
var = "Bert"
txtBox.Text = var
```

5.
```
txtBox.Text = "f" & "lute"
```

6.
```
lstBox.Items.Add("a" & "cute")
```

7.
```
Dim var As Double
var = 123
txtBox.Text = CStr(var)
```

8.
```
Dim var As Double
var = 3
txtBox.Text = CStr(var + 5)
```

9.
```
txtBox.Text = "Your age is " & 21 & "."
```

10.
```
txtBox.Text = "Fred has " & 2 & " children."
```

11.
```
Dim r, b As String
r = "A ROSE"
b = " IS "
txtBox.Text = r & b & r & b & r
```

12.
```
Dim s As String, n As Integer
s = "trombones"
n = 76
txtBox.Text = n & " " & s
```

13.
```
Dim num As Double
txtBox.Text = "5"
num = 0.5 + CDbl(txtBox.Text)
txtBox.Text = CStr(num)
```

14.
```
Dim num As Integer = 2
txtBox.Text = CStr(num)
txtBox.Text = CStr(1 + CInt(txtBox.Text))
```

15.
```
txtBox.Text = "good"
txtBox.Text &= "bye"
```

16.
```
Dim var As String = "eight"
var &= "h"
txtBox.Text = var
```

17.
```
Dim var As String = "WALLA"
var &= var
txtBox.Text = var
```

18.
```
txtBox.Text = "mur"
txtBox.Text &= txtBox.Text
```

19.
```
lstBox.Items.Add("aBc".ToUpper)
lstBox.Items.Add("Wallless".IndexOf("lll"))
lstBox.Items.Add("five".Length)
lstBox.Items.Add("   55 ".Trim & " mph")
lstBox.Items.Add("UNDERSTUDY".Substring(5, 3))
```

20.
```
lstBox.Items.Add("8 Ball".ToLower)
lstBox.Items.Add("colonel".IndexOf("k"))
lstBox.Items.Add("23.45".Length)
lstBox.Items.Add("revolutionary".Substring(1))
lstBox.Items.Add("whippersnapper".IndexOf("pp", 5))
```

21.
```
Dim a As Integer = 4
Dim b As Integer = 2
Dim c As String = "Municipality"
Dim d As String = "pal"
lstBox.Items.Add(c.Length)
lstBox.Items.Add(c.ToUpper)
lstBox.Items.Add(c.Substring(a, b) & c.Substring(5 * b))
lstBox.Items.Add(c.IndexOf(d))
```

22.
```
Dim m As Integer = 4
Dim n As Integer = 3
Dim s As String = "Microsoft"
Dim t As String = "soft"
lstOutput.Items.Add(s.Length)
lstOutput.Items.Add(s.ToLower)
lstOutput.Items.Add(s.Substring(m, n – 1))
lstOutput.Items.Add(s.IndexOf(t))
```

23. How many positions does a string of eight characters have?

24. What is the highest numbered position for a string of eight characters?

25. (True or False) If *n* is the length of *str*, then `str.Substring(n – 1)` is the string consisting of the last character of *str*.

26. (True or False) If *n* is the length of *str*, then `str.Substring(n – 2)` is the string consisting of the last two characters of *str*.

In Exercises 27 through 32, identify any errors.

27.
```
Dim phoneNumber As Double
phoneNumber = "234-5678"
txtBox.Text = "My phone number is " & phoneNumber
```

28.
```
Dim quote As String
quote = I came to Casablanca for the waters.
txtBox.Text = quote & ": " & "Bogart"
```

29.
```
Dim end As String
end = "happily ever after."
txtBox.Text = "They lived " & end
```

30.
```
Dim hiyo As String
hiyo = "Silver"
txtBox = "Hi-Yo " & hiYo
```

31.
```
Dim num As Double = 1234
txtBox.Text = CStr(num.IndexOf("2"))
```

32.
```
Dim num As Integer = 45
txtBox.Text = CStr(num.Length)
```

In Exercises 33 through 36, write an event procedure with the header `Private Sub btnCompute_Click(...) Handles btnCompute.Click`, and having one line for each step. Display each result by assigning it to the txtOutput.Text **property. Lines that display data should use the given variable names.**

33. The following steps give the name and birth year of a famous inventor:

(a) Declare all variables used in steps (b)–(e).
(b) Assign "Thomas" to the variable *firstName*.
(c) Assign "Alva" to the variable *middleName*.
(d) Assign "Edison" to the variable *lastName*.
(e) Assign 1847 to the variable *yearOfBirth*.
(f) Display the inventor's full name followed by a comma and his year of birth.

34. The following steps compute the price of ketchup:

(a) Declare all variables used in steps (b)–(d).
(b) Assign "ketchup" to the variable *item*.
(c) Assign 1.80 to the variable *regularPrice*.
(d) Assign .27 to the variable *discount*.
(e) Display the phrase "1.53 is the sale price of ketchup."

35. The following steps display a copyright statement:

(a) Declare the variable used in step (b).
(b) Assign "Prentice Hall, Inc." to the variable *publisher*.
(c) Display the phrase "(c) Prentice Hall, Inc."

36. The following steps give advice:

(a) Declare the variable used in step (b).
(b) Assign "Fore" to the variable *prefix*.
(c) Display the phrase "Forewarned is Forearmed."

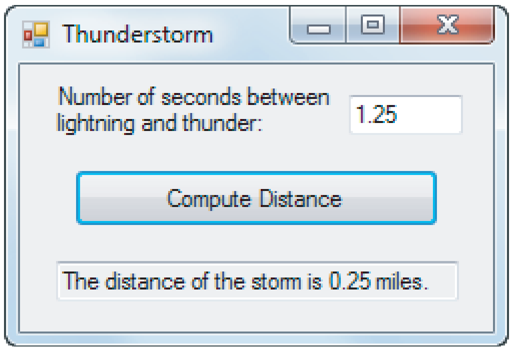
In Exercises 37 and 38, write a line of code to carry out the task. Specify where in the program the line of code should be placed.

37. Declare the variable *str* as a string variable visible to all parts of the program.

38. Declare the variable *str* as a string variable visible only to the btnTest_Click event procedure.

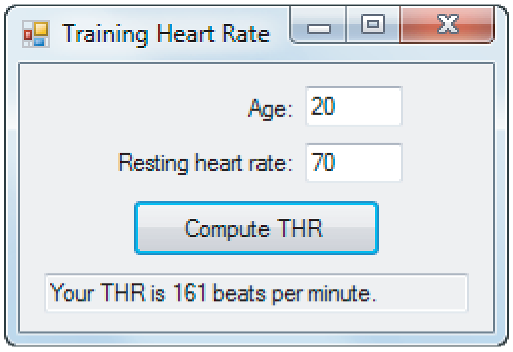
In Exercises 39 through 42, the interface is specified. Write a program to carry out the stated task.

39. If *n* is the number of seconds between lightning and thunder, the storm is *n*/5 miles away. Write a program that reads the number of seconds between lightning and thunder and reports the distance of the storm. A sample run is shown in Fig. 3.3.



```
1.25
The distance of the storm is 0.25 miles.
```

FIGURE 3.3 **Sample output of Exercise 39.**



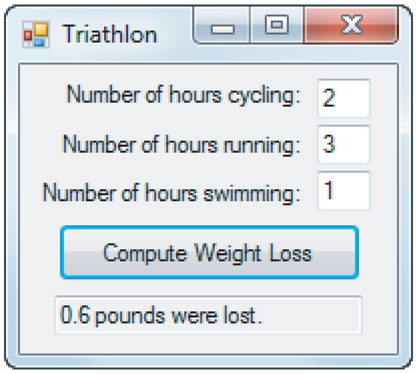
```
Age: 20
Resting heart rate: 70
Your THR is 161 beats per minute.
```

FIGURE 3.4 **Sample output of Exercise 40.**

40. The American College of Sports Medicine recommends that you maintain your *training heart rate* during an aerobic workout. Your training heart rate is computed as $.7*(220 - a) + .3*r$, where *a* is your age and *r* is your resting heart rate (your pulse when you first awaken). Write a program to read a person's age and resting heart rate and display the training heart rate. (Determine *your* training heart rate.) A sample run is shown in Fig. 3.4.

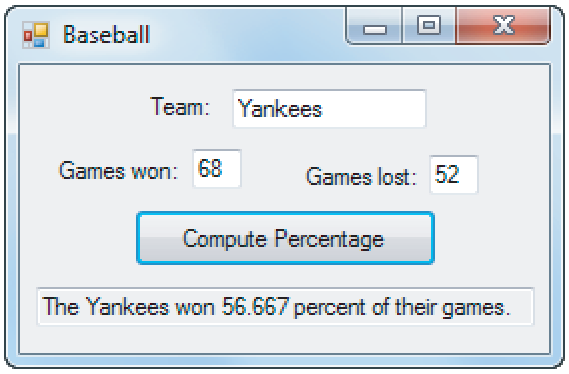
41. The number of calories burned per hour by cycling, running, and swimming are 200, 475, and 275, respectively. A person loses 1 pound of weight for each 3500 calories burned.

Write code to read the number of hours spent at each activity and then display the number of pounds worked off. A sample run is shown in Fig. 3.5.



2
3
1
0.6 pounds were lost.

FIGURE 3.5 **Sample output of Exercise 41.**



Yankees
68
52
The Yankees won 56.667 percent of their games.

FIGURE 3.6 **Sample output of Exercise 42.**

42. Write code to read the name of a baseball team, the number of games won, and the number of games lost, and display the name of the team and the percentage of games won. A sample run is shown in Fig. 3.6.

In Exercises 43 through 48, write a program to carry out the task. The program should use variables for each of the quantities and display the outcome in a text box with a label as in Example 2.

43. Request a company's annual revenue and expenses as input, and display the company's net income (revenue minus expenses). (Test the program with the amounts $550,000 and $410,000.)

44. Request a company's earnings-per-share for the year and the price of one share of stock as input, and then display the company's price-to-earnings ratio (that is, price/earnings). (Test the program with the amounts $5.25 and $68.25.)

45. Calculate the amount of a waiter's tip, given the amount of the bill and the percentage tip as input. (Test the program with $20 and 15 percent.)

46. Convert a percentage to a decimal. For instance, if the user enters 125% into a text box, then the output should be 1.25.

47. Write a program that contains a button and a read-only text box on the form, with the text box initially containing 100. Each time the button is clicked on, the number in the text box should decrease by 1.

48. Write a program that requests a (complete) phone number in a text box and then displays the area code in another text box when a button is clicked on.

49. Write a program that requests a sentence, a word in the sentence, and another word and then displays the sentence with the first word replaced by the second. For example, if the user responds by typing "What you don't know won't hurt you." into the first text box and *know* and *owe* into the second and third text boxes, then the message "What you don't owe won't hurt you." is displayed.

50. Write a program that requests a letter, converts it to uppercase, and gives its first position in the sentence "THE QUICK BROWN FOX JUMPS OVER A LAZY DOG." For example, if the user responds by typing *b* into the text box, then the message "B first occurs in position 10." is displayed.

51. The formula $s = \sqrt{24d}$ gives an estimate of the speed in miles per hour of a car that skidded d feet on dry concrete when the brakes were applied. Write a program that requests the distance skidded and then displays the estimated speed of the car. (Try the program for a car that skids 54 feet.)

52. Write a program that requests a positive number containing a decimal point as input and then displays the number of digits to the left of the decimal point and the number of digits to the right of the decimal point.

53. Write a program that allows scores to be input one at a time, and then displays the average of the scores upon request. (See Fig. 3.7.) The user should type a score into the top text box and then click on the *Record* button. This process can be repeated as many times as desired. At any time the user should be able to click on the *Calculate* button to display the average of all the scores that were entered so far. ***Note:*** This program requires two class-level variables.

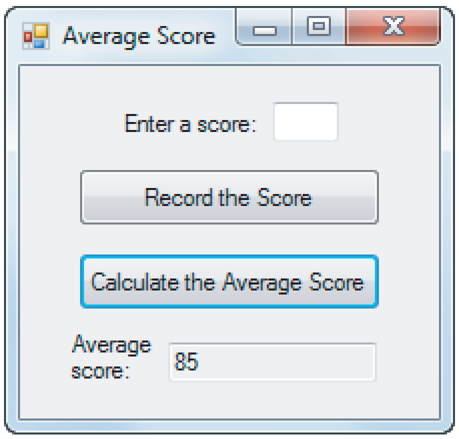
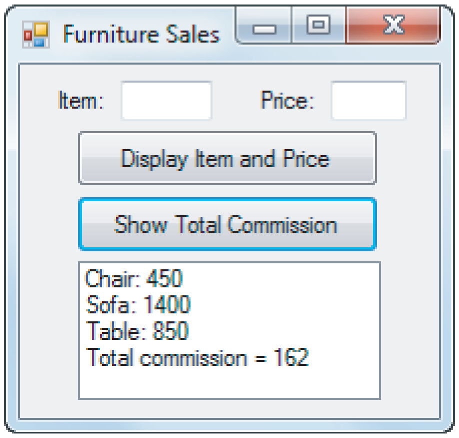


FIGURE 3.7 Sample output of Exercise 53.



```
Chair: 450
Sofa: 1400
Table: 850
Total commission = 162
```

FIGURE 3.8 Sample output of Exercise 54.

54. Write a program that allows furniture sales (item and price) to be displayed in a list box one at a time, and then shows the total commission of the sales (6%) upon request. (See Fig. 3.8.) The user should type each item and price into the text boxes and then click on the *Display* button. This process can be repeated as many times as desired. At any time the user should be able to press the *Show* button to display the total commission of all the sales that were entered. ***Note:*** This program requires a class-level variable.

55. Add an event procedure to Example 2 so that txtSum will be cleared whenever either the first number or the second number is changed.

Solutions to Practice Problems 3.2

1. −1. There is no uppercase letter E in the string "Computer". IndexOf distinguishes between uppercase and lowercase.
2. The first statement displays 16 in the text box, whereas the second statement displays 88. With Option Strict in effect, the first statement would not be valid if CStr were missing, since 8 + 8 is a number and txtBox.Text is a string. Visual Basic treats the second statement as if it were

```
txtBox.Text = CStr(8) & CStr(8)
```

3. Some possibilities are

PROHIBITED STATEMENT	SUGGESTION FOR FIXING
`Dim x As Double = "23"`	Replace '"23"' with 'CDbl("23")'.
`dblVar = txtBox.Text`	Replace 'txtBox.Text' with 'CDbl(txtBox.Text)'.
`dblVar = 2 & 3`	Replace '2 & 3' with 'CDbl(2 & 3)'.