5

General Procedures

5.1 Function Procedures

Visual Basic has two devices, **Function procedures** and **Sub procedures**, that are used to break complex problems into small problems to be solved one at a time. To distinguish them from event procedures, Function and Sub procedures are referred to as **general procedures**. General procedures allow us to write and read a program in such a way that we first focus on the tasks and later on how to accomplish each task. They also eliminate repetitive code and can be reused in other programs.

In this section we show how Function procedures are defined and used. Sub procedures are presented in Sections 5.2 and 5.3.

Visual Basic has many built-in functions. In one respect, functions are like miniature programs. They receive input, they process the input, and they have output. Some functions we encountered earlier are listed in Table 5.1.

TABLE 5.1 Some Visual Basic built-in functions.

Function	Example	Input	Output
Int	Int(2.6) is 2	number	number
Chr	Chr(65) is "A"	number	string
Asc	Asc("Apple") is 65	string	number
FormatNumber	FormatNumber(12345.628, 1) is 12,345.6	number, number	string

Although the input can consist of several values, the output is always a single value. A function is said to **return** its output. For instance, in the first example of Table 5.1, we say that the Int function returns the value 2. The items inside the parentheses are called **arguments**. The first three functions in Table 5.1 have one argument and the fourth function has two arguments. Arguments can be literals (as in Table 5.1), variables, or expressions. Variables are the most common types of arguments. The following lines of code illustrate the use of variables and expressions as arguments for the Int function.

```
Dim num1 As Double = 2.6
Dim num2 As Double = Int(num1)        'variable as an argument

Dim num1 As Double = 1.3
Dim num2 As Double = Int(2 * num1)    'expression as an argument
```

The second line of code above is said to **call** the Int function and to **pass** the value of *num1* to the function.

In addition to using built-in functions, we can define functions of our own. These new functions, called **Function procedures** or **user-defined functions**, are used in the same way as built-in functions. Like built-in functions, Function procedures have a single output that can be of any data type. Function procedures are used in exactly the same way as built-in functions. Function procedures are defined by function blocks of the form

VideoNote
Function procedures

```
Function FunctionName(ByVal var1 As Type1,
                      ByVal var2 As Type2, ...) As ReturnDataType
  statement(s)
  Return expression
End Function
```

The variables appearing in the header are called **parameters**. If the word ByVal is omitted when you type in the header, ByVal will be inserted automatically by the Code Editor when you move the cursor away from the header. For now, think of ByVal as a keyword analogous to Dim; that is, it declares a parameter to be of a certain type and sets aside a portion of memory to hold its value. The scope of each parameter is limited to its function block, as is any variable declared inside the Function procedure.

Function names should be suggestive of the role performed and must conform to the rules for naming variables. By convention, function names begin with an uppercase letter. *ReturnDataType*, which specifies the type of the output, will be one of String, Integer, Double, Date, Boolean, and so on. In the preceding general code, the Return statement specifies the output, which must be of type *ReturnDataType*. Function procedures can contain several Return statements, and must contain at least one.

Function procedures are typed directly into the Code Editor outside any other procedure. After you type the header and then press the Enter key, the editor automatically inserts the line "End Function" and a blank line separating the two lines of code. Also, the smart indenting feature of the Code Editor automatically indents all lines in the block of code between the header and "End Function" statements.

User-Defined Functions Having One Parameter

The following two Function procedures have just one parameter. (Figure 5.1 identifies the different parts of the first function's header.)

```
Function FtoC(ByVal t As Double) As Double
  'Convert Fahrenheit temperature to Celsius
  Return (5 / 9) * (t - 32)
End Function

Function FirstName(ByVal fullName As String) As String
  'Extract the first name from a full name
  Dim firstSpace As Integer
  firstSpace = fullName.IndexOf(" ")
  Return fullName.Substring(0, firstSpace)
End Function
```

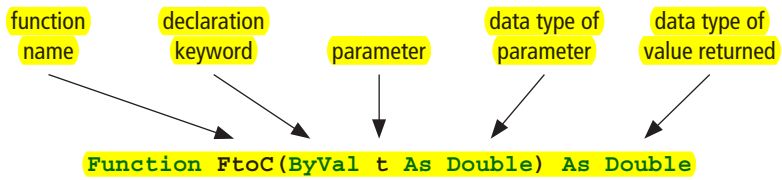
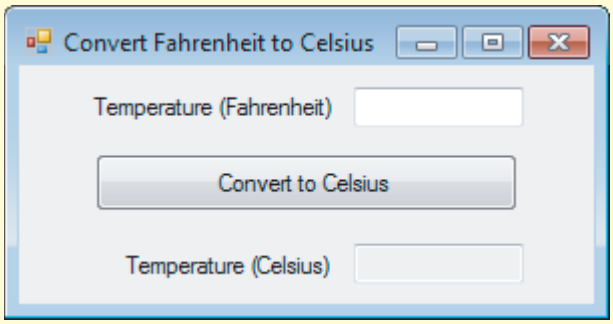


FIGURE 5.1 Header of the FtoC Function procedure.

Example 1 The following program uses the Function procedure FtoC. The fourth line of the btnConvert_Click event procedure, `celsiusTemp = FtoC(fahrenheitTemp)`, calls the function FtoC. The value of the argument *fahrenheitTemp* is assigned to the parameter *t* in the Function procedure header. (We say that the value of *fahrenheitTemp* is passed to the parameter *t*.) After the Function procedure does a calculation using the parameter *t*,

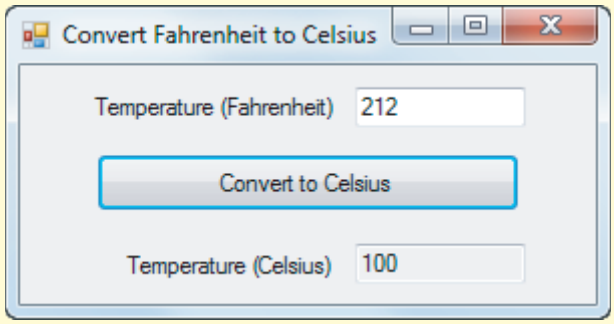
the calculated value is the output of the function FtoC and is assigned to the variable *celsiusTemp*.



OBJECT	PROPERTY	SETTING
frmConvert	Text	Convert Fahrenheit to Celsius
lblTempF	Text	Temperature (Fahrenheit)
txtTempF		
btnConvert	Text	Convert to Celsius
lblTempC	Text	Temperature (Celsius)
txtTempC	ReadOnly	True

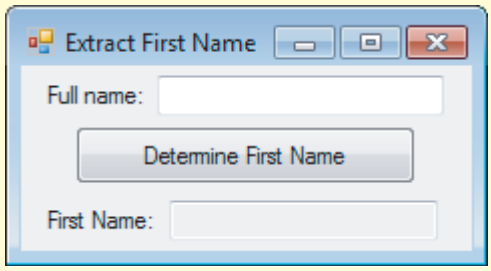
```
Private Sub btnConvert_Click(...) Handles btnConvert.Click
  Dim fahrenheitTemp, celsiusTemp As Double
  fahrenheitTemp = CDbl(txtTempF.Text)
  celsiusTemp = FtoC(fahrenheitTemp)
  txtTempC.Text = CStr(celsiusTemp)
  'Note: The above four lines can be replaced with the single line
  'txtTempC.Text = CStr(FtoC(CDbl(txtTempF.Text)))
End Sub

Function FtoC(ByVal t As Double) As Double
  'Convert Fahrenheit temperature to Celsius
  Return (5 / 9) * (t - 32)
End Function
```

[Run, type 212 into the text box, and then click on the button.]



Example 2 The following program uses the Function procedure FirstName. The fifth line of the btnDetermine_Click event procedure, `txtFirstName.Text = FirstName(fullName)`, passes the value of the argument *fullName* to the parameter *fullName* in the Function procedure. Although the parameter in the Function procedure has the same name as the argument passed to

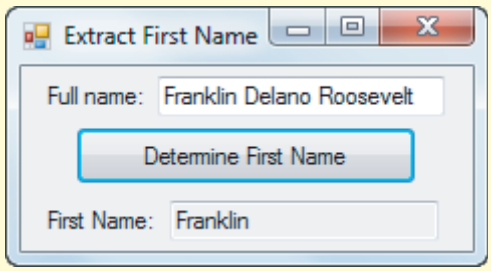
it, they are different variables. This is analogous to the situation in which variables in two different event procedures have the same name, but separate identities.



OBJECT	PROPERTY	SETTING
frmFirstName	Text	Extract First Name
lblName	Text	Full name:
txtFullName		
btnDetermine	Text	Determine First Name
lblFirstName	Text	First Name:
txtFirstName	ReadOnly	True

```
Private Sub btnDetermine_Click(...) Handles btnDetermine.Click
  'Determine a person's first name
  Dim fullName As String
  fullName = txtFullName.Text
  txtFirstName.Text = FirstName(fullName)
End Sub

Function FirstName(ByVal fullName As String) As String
  'Extract the first name from a full name
  Dim firstSpace As Integer
  firstSpace = fullName.IndexOf(" ")
  Return fullName.Substring(0, firstSpace)
End Function
```

[Run, type Franklin Delano Roosevelt into the text box, and then click on the button.]



In general, consider a calling statement of the form

```
Dim var1 as Type1 = FunctionName(arg)
```

where the Function procedure header has the form

```
Function FunctionName(ByVal par As parameterType) As ReturnDataType
```

The variables *arg* and *par* must have the same data type, and *Type1* must be the same as *ReturnDataType*. In many cases, code is easier to read when the same name is used for the argument and the parameter it is passed to. Although they needn't have the same name, they must have the same data type.[1]

[1] There are exceptions to this rule. For instance, if *Type1* is a numeric data type, then *parameterType* can have any numeric data type that is wider than *Type1*. In this book, the two data types will always be the same. Similar considerations apply to *ReturnDataType* and *Type1*

User-Defined Functions Having Several Parameters

The following two Function procedures have more than one parameter. In the second function, one-letter names have been used for the parameters so that the mathematical formulas will look familiar and be easy to read. Because the names are not descriptive, the meanings of these parameters are spelled out in comment statements.

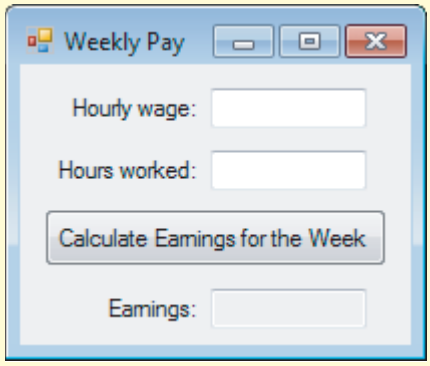
```
Function Pay(ByVal wage As Double, ByVal hrs As Double) As Double
  'Calculate weekly pay with time-and-a-half for overtime
  Dim amount As Double
  Select Case hrs
    Case Is <= 40
      amount = wage * hrs
    Case Is > 40
      amount = (wage * 40) + ((1.5) * wage * (hrs - 40))
  End Select
  Return amount
End Function

Function FutureValue(ByVal p As Double, ByVal r As Double,
                     ByVal c As Integer, ByVal n As Integer) As Double
  'Find the future value of a bank savings account
  'p  principal, the amount deposited
  'r  annual rate of interest in decimal form
  'c  number of times interest is compounded per year
  'n  number of years
  Dim i As Double      'interest rate per period
  Dim m As Integer     'total number of times interest is compounded
  i = r / c
  m = c * n
  Return p * ((1 + i) ^ m)
End Function
```

When a function with several parameters is called, there must be the same number of arguments as parameters in the function. Also, the data types of the arguments must be the same (and in the same order) as the data types of the parameters. For instance, in a statement of the form

```
numVar = FutureValue(arg1, arg2, arg3, arg4)
```

arg1 and *arg2* must be of type Double, and *arg3* and *arg4* must be of type Integer.

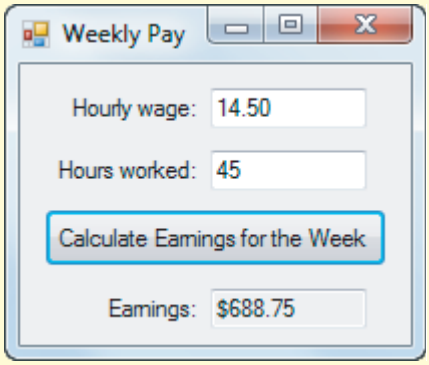
Example 3 The following program uses the Function procedure Pay. Here the arguments have different names than the corresponding parameters. As required, however, they have the same data types.



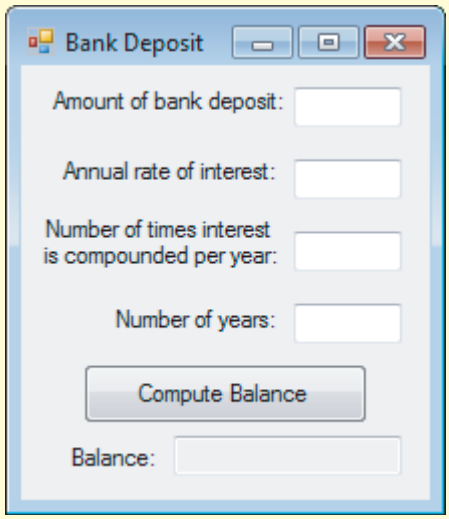
OBJECT	PROPERTY	SETTING
frmPay	Text	Weekly Pay
lblWage	Text	Hourly wage:
txtWage		
lblHours	Text	Hours worked:
txtHours		
btnCalculate	Text	Calculate Earnings for the Week
lblEarnings	Text	Earnings:
txtEarnings	ReadOnly	True

```
Private Sub btnCalculate_Click(...) Handles btnCalculate.Click
  'Calculate a person's weekly pay
  Dim hourlyWage, hoursWorked As Double
  hourlyWage = CDbl(txtWage.Text)
  hoursWorked = CDbl(txtHours.Text)
  txtEarnings.Text = FormatCurrency(Pay(hourlyWage, hoursWorked))
End Sub

Function Pay(ByVal wage As Double, ByVal hrs As Double) As Double
  'Calculate weekly pay with time-and-a-half for overtime
  Dim amount As Double
  Select Case hrs
    Case Is <= 40
      amount = wage * hrs
    Case Is > 40
      amount = (wage * 40) + ((1.5) * wage * (hrs - 40))
  End Select
  Return amount
End Function
```

[Run, enter values into the top two text boxes, and click on the button.]



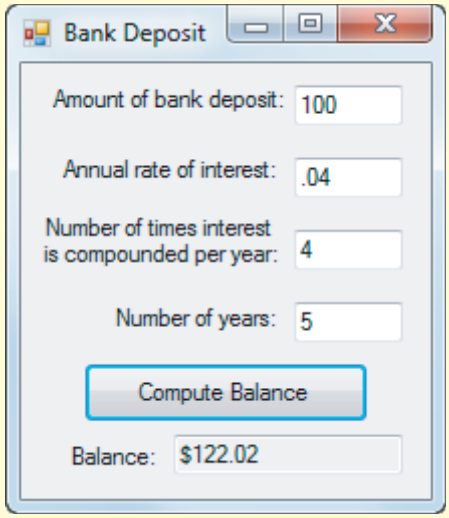
Example 4 The following program uses the Function procedure FutureValue. With the responses shown when the program is run, the program computes the balance in a savings account when $100 is deposited for five years at 4% interest compounded quarterly. Interest is earned four times per year at the rate of 1% per interest period. There will be 5·4, or 20, interest periods.



OBJECT	PROPERTY	SETTING
frmBank	Text	Bank Deposit
lblAmount	Text	Amount of bank deposit:
txtAmount		
lblRate	Text	Annual rate of interest:
txtRate		
lblNumComp	AutoSize	False
	Text	Number of times interest is compounded per year:
txtNumComp		
lblNumYrs	Text	Number of years:
txtNumYrs		
btnCompute	Text	Compute Balance
lblBalance	Text	Balance:
txtBalance	ReadOnly	True

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  'Find the future value of a bank deposit
  Dim p As Double = CDbl(txtAmount.Text)
  Dim r As Double = CDbl(txtRate.Text)
  Dim c As Integer = CInt(txtNumComp.Text)
  Dim n As Integer = CInt(txtNumYrs.Text)
  Dim balance As Double = FutureValue(p, r, c, n)
  txtBalance.Text = FormatCurrency(balance)
End Sub

Function FutureValue(ByVal p As Double, ByVal r As Double,
                     ByVal c As Integer, ByVal n As Integer) As Double
  'Find the future value of a bank savings account
  'p  principal, the amount deposited
  'r  annual rate of interest in decimal form
  'c  number of times interest is compounded per year
  'n  number of years
  Dim i As Double     'interest rate per period
  Dim m As Integer    'total number of times interest is compounded
  i = r / c
  m = c * n
  Return p * ((1 + i) ^ m)
End Function
```

[Run, type 100, .04, 4, and 5 into the text boxes, and then click on the button.]

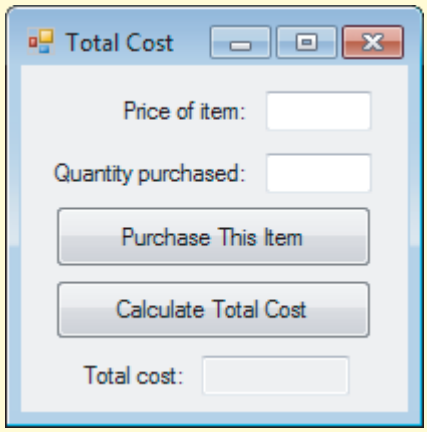


User-Defined Functions Having No Parameters

A Function procedure needn't have any parameters.

Example 5 The following program allows a person to determine the total cost of several purchases. The user enters the data for a purchase in the top two text boxes and then clicks on the *Purchase This Item* button. The user can continue to enter data for further purchases. After

all purchases have been made, the user clicks on the *Calculate Total Cost* button to display the sum of the costs of the purchases plus the sales tax.



OBJECT	PROPERTY	SETTING
frmCost	Text	Total Cost
lblPrice	Text	Price of item:
txtPrice		
lblQuantity	Text	Quantity purchased:
txtQuantity		
btnPurchase	Text	Purchase This Item
btnCalculate	Text	Calculate Total Cost
lblTotal	Text	Total cost:
txtTotal	ReadOnly	True

```
Dim subTotal As Double
Const SALES_TAX = 0.05

Private Sub btnPurchase_Click(...) Handles btnPurchase.Click
  subTotal += CostOfItem()
  txtPrice.Clear()
  txtQuantity.Clear()
  txtPrice.Focus()
End Sub

Private Sub btnCalculate_Click(...) Handles btnCalculate.Click
  Dim totalCost As Double = subTotal + (SALES_TAX * subTotal)
  txtTotal.Text = FormatCurrency(totalCost)
End Sub

Function CostOfItem() As Double
  Dim price As Double = CDbl(txtPrice.Text)
  Dim quantity As Integer = CInt(txtQuantity.Text)
  Dim cost = price * quantity
  Return cost
End Function
```

[Run, enter 5 and 2, click on the *Purchase* button, enter 6 and 1, click on the *Purchase* button, and click on the *Calculate* button.]

$16.80 is displayed in the "Total cost" text box.

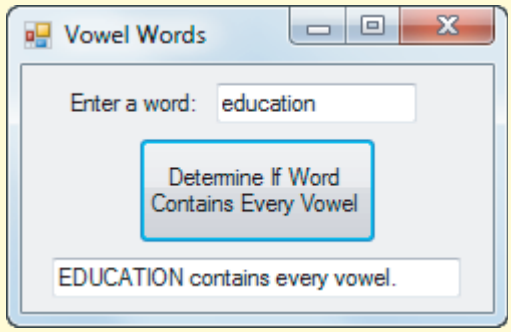
User-Defined Boolean-Valued Functions

So far, the values returned by Function procedures have been numbers or strings. However, a Function procedure can also return a Boolean value—that is, True or False. The following program uses a Boolean-valued function and demonstrates an important feature of Function procedures: namely, a Function procedure can contain more than one Return statement. When the first Return statement is encountered, it determines the function value, and the execution of code in the function block terminates.

Example 6 The following program uses a Boolean-valued function to determine whether a word input by the user is a vowel word—that is, contains every vowel. Some examples of vowel words are "sequoia", "facetious", and "dialogue". The Function procedure IsVowelWord examines the word for vowels one at a time and terminates when a vowel is found to be missing.

```
Private Sub btnDetermine_Click(...) Handles btnDetermine.Click
  Dim word As String
  word = txtWord.Text.ToUpper
  If IsVowelWord(word) Then
    txtOutput.Text = word & " contains every vowel."
  Else
    txtOutput.Text = word & " does not contain every vowel."
  End If
End Sub

Function IsVowelWord(ByVal word As String) As Boolean
  If word.IndexOf("A") = -1 Then
    Return False
  End If
  If word.IndexOf("E") = -1 Then
    Return False
  End If
  If word.IndexOf("I") = -1 Then
    Return False
  End If
  If word.IndexOf("O") = -1 Then
    Return False
  End If
  If word.IndexOf("U") = -1 Then
    Return False
  End If
  Return True      'All vowels are present.
End Function
```

[Run, type a word into the top text box, and click on the button.]



Comments

1. After a Function procedure has been defined, IntelliSense helps you call the function. Word Completion helps type the function's name, and Parameter Info displays the function's parameters. As soon as you type in the left parenthesis preceding the arguments, a Parameter Info banner appears giving information about the number, names, and types of

the parameters required by the function. See Fig. 5.2. A syntax error occurs if the number of arguments in the calling statement is different from the number of parameters in the called Function procedure. Also, having an argument of a data type that cannot be assigned to the corresponding parameter is a syntax error. Parameter Info helps you prevent both of these kinds of syntax errors.

```
Private Sub btnSalary_Click(...) Handles btnSalary.Click
  Dim wage As Double = 15.75
  Dim hrs As Double = 45
  txtSalary.Text = FormatCurrency(Pay(
                                  Pay(wage As Double, hrs As Double) As Double
```

FIGURE 5.2 The Parameter Info help feature.

2. In this text, Function procedure names begin with uppercase letters in order to distinguish them from variable names. Like variable names, however, they can be written with any combination of uppercase and lowercase letters.

Practice Problems 5.1

1. Suppose a program contains the lines

```
Dim n As Double, x As String
lstOutput.Items.Add(Arc(n, x))
```

What types of inputs (numeric or string) and output does the function Arc have?

2. Determine the error in the following program.

```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
  Dim num As Integer = 3
  Dim word As String = "Visual"
  MessageBox.Show("The third letter of the word is " &
                  FindLetter(word, num) & ".")
End Sub

Function FindLetter(ByVal num As Integer, ByVal word As String) As String
  Return word.Substring(num - 1, 1)
End Function
```

EXERCISES 5.1

In Exercises 1 through 10, determine the output displayed when the button is clicked.

1.
```
Private Sub btnConvert_Click(...) Handles btnConvert.Click
  'Convert Celsius to Fahrenheit
  Dim temp As Double = 95
  txtOutput.Text = CStr(CtoF(temp))
End Sub
```

```
Function CtoF(ByVal t As Double) As Double
  Return ((9 / 5) * t) + 32
End Function
```

2.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim acres As Double  'Number of acres in a parking lot
  acres = 5
  txtOutput.Text = "You can park about " & Cars(acres) & " cars."
End Sub

Function Cars(ByVal x As Double) As Double
  'Number of cars that can be parked
  Return 100 * x
End Function
```

3.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Rule of 72
  Dim p As Double
  p = CDbl(txtPopGr.Text) 'Population growth as a percent
  txtOutput.Text = "The population will double in " &
                   DoublingTime(p) & " years."
End Sub

Function DoublingTime(ByVal x As Double) As Double
  'Estimate time required for a population to double
  'at a growth rate of x percent
  Return 72 / x
End Function
```

(Assume the text box txtPopGr contains the number 3.)

4.
```
Private Sub btnDetermine_Click(...) Handles btnDetermine.Click
  Dim numOne, numTwo, numThree, numLowest As Double
  numOne = CDbl(txtOne.Text)
  numTwo = CDbl(txtTwo.Text)
  numThree = CDbl(txtThree.Text)
  numLowest = FindLowest(numOne, numTwo, numThree)
  txtLowest.Text = CStr(numLowest)
End Sub

Function FindLowest(ByVal x As Double, ByVal y As Double,
                    ByVal z As Double) As Double
  'Find the lowest of three numbers denoted by x, y, and z
  Dim lowest As Double
  lowest = x
  If y < lowest Then
    lowest = y
  End If
  If z < lowest Then
    lowest = z
  End If
  Return lowest
End Function
```

(Assume the first three text boxes contain the numbers 7, 4, and 3.)

5.
```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
  Dim num As Integer = 27
  If IsEven(num) Then
    MessageBox.Show(num & " is an even number.")
  Else
    MessageBox.Show(num & " is an odd number.")
  End If
End Sub

Function IsEven(ByVal n As Integer) As Boolean
  If n Mod 2 = 0 Then
    Return True
  Else
    Return False
  End If
End Function
```

6.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim d As Date = #12/4/2011#
  txtOutput.Text = MonthAbbr(d)
End Sub

Function MonthAbbr(ByVal d As Date) As String
  Dim str As String = FormatDateTime(d, DateFormat.LongDate)
  Dim n As Integer = str.IndexOf(" ")
  Return str.Substring(n + 1, 3)
End Function
```

7.
```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
  Dim taxableIncome As Double = 5000
  MessageBox.Show("Your state income tax is " &
                  FormatCurrency(StateTax(taxableIncome)) & ".")
End Sub

Function StateTax(ByVal income As Double) As Double
  'Calculate state tax for a single resident of Connecticut
  Select Case income
    Case Is <= 10000
      Return 0.03 * income
    Case Else
      Return 300 + (0.05 * (income - 10000))
  End Select
End Function
```

8.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Triple a number
  Dim num As Double = 5
  lstOutput.Items.Add(Triple(num))
  lstOutput.Items.Add(num)
End Sub
```

```
Function Triple(ByVal x As Double) As Double
  Dim num As Double = 3
  Return num * x
End Function
```

9.
```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
  Dim word1 As String = "beauty"
  Dim word2 As String = "age"
  MessageBox.Show(First(word1, word2) & " before " & Last(word1, word2))
End Sub

Function First(ByVal w1 As String, ByVal w2 As String) As String
  If w1 < w2 Then
    Return w1
  Else
    Return w2
  End If
End Function

Function Last(ByVal w1 As String, ByVal w2 As String) As String
  If w1 > w2 Then
    Return w1
  Else
    Return w2
  End If
End Function
```

10.
```
Private Sub btnOutput_Click(...) Handles btnOutput.Click
  Dim num1 As Integer = 84
  Dim num2 As Integer = 96
  If IsAnA(num1, num2) Then
    MessageBox.Show("A average")
  Else
    MessageBox.Show("not an A average")
  End If
End Sub

Function IsAnA(ByVal n1 As Integer, ByVal n2 As Integer) As Boolean
  If ((n1 + n2) / 2) >= 89.5 Then
    Return True
  Else
    Return False
  End If
End Function
```

In Exercises 11 and 12, identify the errors.

11.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Select a greeting
  Dim answer As Integer
  answer = CInt(InputBox("Enter 1 or 2."))
  txtOutput.Text = CStr(Greeting(answer))
End Sub
```

```
Function Greeting(ByVal x As Integer) As Integer
  Return "hellohi ya".Substring(5 * (x - 1), 5)
End Function
```

12.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim word As String
  word = InputBox("What is your favorite word?")
  txtOutput.Text = "When the word is written twice, "  &
                   Twice(word) & " letters are used."
End Sub

Function Twice(ByVal w As String) As Integer
  'Compute twice the length of a string
  Dim len As Integer
  Return len = 2 * w.Length
End Function
```

In Exercises 13 through 23, construct user-defined functions to carry out the primary task(s) of the program.

13. To determine the number of square centimeters of tin needed to make a tin can, add the square of the radius of the can to the product of the radius and height of the can, and then multiply this sum by 6.283. Write a program that requests the radius and height of a tin can in centimeters as input and displays the number of square centimeters of tin required to make the can.

14. Table 5.2 gives the Saffir-Simpson scale for categorizing hurricanes. Write a program that requests a wind speed in miles/hour and displays the category of the storm.

TABLE 5.2 Rating of hurricanes.

Wind Speed (in mph)	Rating
74 to 95	Category One
96 to 110	Category Two
111 to 130	Category Three
131 to 155	Category Four
Over 155	Category Five

15. The federal government developed the body mass index (BMI) to determine ideal weights. Body mass index is calculated as 703 times the weight in pounds, divided by the square of the height in inches, and then rounded to the nearest whole number. Write a program that accepts a person's weight and height as input and gives the person's body mass index. ***Note:*** A BMI of 19 to 25 corresponds to a healthy weight.

16. In order for exercise to be beneficial to the cardiovascular system, the heart rate (number of heart beats per minute) must exceed a value called the *training heart rate*, THR. A person's THR can be calculated from their age and resting heart rate (pulse rate when first awakening) as follows:

(a) Calculate the maximum heart rate as 220 − age.
(b) Subtract the resting heart rate from the maximum heart rate.
(c) Multiply the result in step (b) by 60%, and then add the resting heart rate.

Write a program to request a person's age and resting heart rate as input and display their THR. (Test the program with an age of 20 and a resting heart rate of 70, and then determine *your* training heart rate.)

17. The three components for a serving of popcorn at a movie theater are popcorn, butter substitute, and a bucket. Write a program that requests the cost of these three items and the price of the serving as input and then displays the profit. (Test the program where popcorn costs 5 cents, butter substitute costs 2 cents, the bucket costs 25 cents, and the selling price is $5.)

18. Write a program that requests the numeric grades on a midterm and a final exam and then uses a Function procedure to assign a semester grade (A, B, C, D, or F). The final exam should count twice as much as the midterm exam, the semester average should be rounded up to the nearest whole number, and the semester grade should be assigned by the following criteria: 90–100 (A), 80–89 (B), ... Use a function called Ceil that rounds noninteger numbers up to the next integer. The function Ceil can be defined by Ceil(x) = −Int(−x).

19. The original cost of airmail letters was 5 cents for the first ounce and 10 cents for each additional ounce. Write a program to compute the cost of a letter whose weight is given by the user in a text box. Use a function called Ceil that rounds noninteger numbers up to the next integer. The function Ceil can be defined by Ceil(x) = −Int(−x). (Test the program with the weights 4, 1, 2.5, and .5 ounces.)

20. Suppose a fixed amount of money is deposited at the beginning of each month into a savings account paying 6% interest compounded monthly. After each deposit is made, [new balance] = 1.005 * [previous balance one month ago] + [fixed amount]. Write a program that requests the fixed amount of the deposits as input and displays the balance after each of the first four deposits. Shown below is the outcome when 1000 is typed into the text box for the amount deposited each month.

```
Month 1:  $1,000.00
Month 2:  $2,005.00
Month 3:  $3,015.03
Month 4:  $4,030.10
```

21. Write a program to request the name of a United States senator as input and display the address and salutation for a letter to the senator. Assume the name has two parts, and use a function to determine the senator's last name. The outcome when Robert Smith is typed into the input dialog box requesting the senator's name follows.

```
The Honorable Robert Smith
United States Senate
Washington, DC 20001


Dear Senator Smith,
```

22. Write a program that requests a date as input and then gives the spelled-out day of the week for that date as output. The program should use a function with the following header.

```
Function DayOfWeek(ByVal d As Date) As String
```

23. Write a program that requests a year as input and then tells whether or not the year is a leap year. The program should use a Boolean-value function named IsLeapYear. ***Hint:*** Use the DateDiff function.

Solutions to Practice Problems 5.1

1. The first argument, *n* takes a value of type Double and the second argument, *x*, takes a String value; therefore, the input consists of a number and a string. From the two lines shown here, there is no way to determine the type of the output. This can be determined only by looking at the definition of the function.

2. The two arguments in **`FindLetter(word, num)`** are in the wrong order. Since the two parameters in the header for the Function procedure have types Integer and String, in that order, the arguments must have the same types and order when the Function procedure is called. The function call should be **`FindLetter(num, word)`**. Visual Basic matches arguments to parameters based on their positions, not on their names.

5.2 Sub Procedures, Part I

Sub procedures share several features with Function procedures.

- Both are written as a separate block of code that can be called to perform a specific task.
- Both are used to break complex problems into small problems.
- Both are used to eliminate repetitive code.
- Both can be reused in other programs.
- Both make a program easier to read by separating it into logical units.
- Both have parameters that are declared in a header.

Sub procedures, however, do not return a value associated with their name. The most common uses of Sub procedures are to receive input, process input, or display output.

Defining and Calling Sub Procedures

VideoNote
Sub procedures

Sub procedures are defined by blocks of the form

```
Sub ProcedureName(ByVal par1 As Type1,
                  ByVal par2 As Type2,
                    ⋮
                  ByVal parN As TypeN)
  statement(s)
End Sub
```

In the block above, one or more of the ByVals can be replaced with the keyword ByRef. (The use of ByRef will be discussed in the next section.)

Like Function procedure names, the names of Sub procedures must conform to the rules for naming variables. By convention, Sub procedure names begin with an uppercase letter and describe its purpose. In this section all parameters will be preceded by the keyword ByVal. The primary difference will be that Sub procedures will perform some task (such as displaying output) rather than return a value.

Sub procedures are called by statements of the form

```
ProcedureName(arg1, arg2, ..., argN)
```

When a Sub procedure is called, the value of each argument is assigned to the corresponding parameter, the statement(s) inside the procedure block are carried out, and execution continues with the statement following the calling statement.

Here is an example of a Sub procedure.

```
Sub DisplaySum(ByVal num1 As Double, ByVal num2 As Double)
  Dim z As Double
  z = num1 + num2
  lstOutput.Items.Add(z)
End Sub
```

When a statement such as

```
DisplaySum(3, 4)
```