PREV NEXT

[Page 131]

Chapter 4. General Procedures

(This item omitted from WebBook edition)

PREV PREV NEXT NEXT

[Page 132]

4.1. Sub Procedures, Part I

Visual Basic has two devices, Sub procedures and Function procedures, that are used to break complex problems into small problems to be solved one at a time. To distinguish them from event procedures, Sub and Function procedures are referred to as general procedures. General procedures also eliminate repetitive code and can be reused in other programs.

In this section, we show how Sub procedures are defined and used. The programs in this section are designed to demonstrate the use of Sub procedures rather than to accomplish sophisticated programming tasks. Later chapters of the book use them for more substantial programming efforts.

A Sub procedure is a part of a program that performs one or more related tasks, has its own name, and is written as a separate part of the program. The simplest sort of Sub procedure has the form

```
Sub ProcedureName()
  statement(s)
End Sub
```

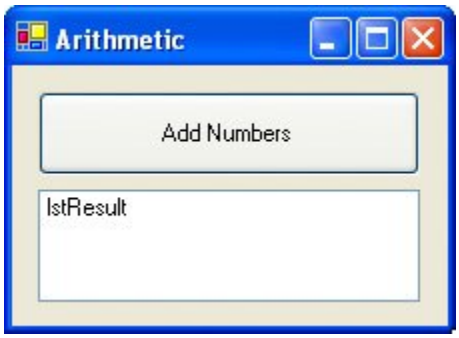
A Sub procedure is invoked with a statement consisting only of the name of the procedure, that is, a statement of the form

```
ProcedureName()
```

Such a statement is said to call the Sub procedure and is referred to as a call statement.

The rules for naming general procedures are identical to the rules for naming variables. The name chosen for a Sub procedure should describe the task it performs. Sub procedures are typed directly into the Code window.

Consider the following program that calculates the sum of two numbers. This program will be revised to incorporate Sub procedures.



Object	Property	Setting
frmAdd	Text	Arithmetic
btnAdd	Text	Add Numbers
lstResult		

```
Private Sub btnAdd_Click(...) Handles btnAdd.Click
  'Display the sum of two numbers
  Dim num1, num2 As Double
  lstResult.Items.Clear()
  lstResult.Items.Add("This program displays a sentence")
  lstResult.Items.Add("identifying two numbers and their sum.")
  lstResult.Items.Add("")
  num1 = 2
  num2 = 3
```

```
  lstResult.Items.Add("The sum of " & num1 & " and " _
                      & num2 & " is " & num1 + num2 & ".")
End Sub
```

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

```
This program displays a sentence
identifying two numbers and their sum.
The sum of 2 and 3 is 5.
```

The tasks performed by this program can be summarized as follows:

Task #1: Explain purpose of program.

Task #2: Display numbers and their sum.

Sub 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.

Example 1.

The following program uses a Sub procedure to accomplish the first task of the preceding program. When the statement `ExplainPurpose()` is reached, execution jumps to the `Sub ExplainPurpose()` statement. The lines between `Sub ExplainPurpose()` and `End Sub` are executed, and then execution continues with the line following the call statement.

```
PrivateSub btnAdd_Click(...) Handles btnAdd.Click
```

```
  'Display the sum of two numbers
  Dim num1, num2 As Double
  lstResult.Items.Clear()
  ExplainPurpose()
  lstResult.Items.Add("")
  num1 = 2
  num2 = 3
  lstResult.Items.Add("The sum of " & num1 & " and " _
                      & num2 & " is " & num1 + num2 & ".")
End Sub

Sub ExplainPurpose()
  'Explain the task performed by the program
  lstResult.Items.Add("This program displays a sentence")
  lstResult.Items.Add("identifying two numbers and their sum.")
End Sub
```

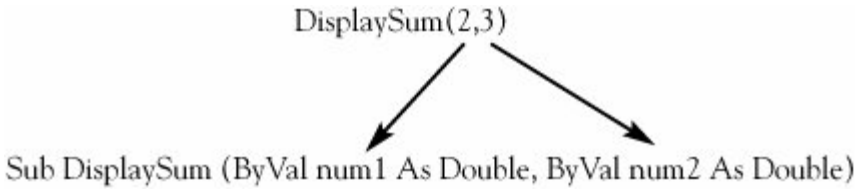
Note: When you type `Sub ExplainPurpose` and then press the Enter key, the editor automatically inserts the parentheses, the line `End Sub`, and a blank line separating the two lines of code. Also, the smart indenting feature of the editor automatically indents all lines in the block of code between the Sub and End Sub statements.

In Example 1, the btnAdd_Click event procedure is referred to as the calling procedure and the ExplainPurpose Sub procedure is referred to as the called procedure. The second task performed by the addition program also can be handled by a Sub procedure. The values of the two numbers, however, must be transmitted to the Sub procedure. This transmission is called passing.

[Page 134]

Example 2.

The following revision of the program in Example 1 uses a Sub procedure to accomplish the second task. The statement `DisplaySum(2, 3)` causes execution to jump to the `Sub DisplaySum(ByVal num1 As Double, ByVal num2 As Double)` statement, which assigns the number 2 to num1 and the number 3 to num2.



After the lines between `Sub DisplaySum(ByVal num1 As Double, ByVal num2 As Double)` and `End Sub` are executed, execution continues with the line following `DisplaySum(2, 3)`, namely, the `End Sub` statement in the event procedure. Note: If you

don't type in the word ByVal from the Sub DisplaySum line, the editor will automatically insert it when you either press the Enter key or move the cursor away from the line. In the next section, we consider an alternative to the keyword ByVal. For now we needn't be concerned with ByVal.

```
Private Sub btnAdd_Click(...) Handles btnAdd.Click
  'Display the sum of two numbers
  lstResult.Items.Clear()
  ExplainPurpose()
  lstResult.Items.Add("")
  DisplaySum(2, 3)
End Sub

Sub DisplaySum(ByVal num1 As Double, ByVal num2 As Double)
  'Display numbers and their sum
  lstResult.Items.Add("The sum of " & num1 & " and " _
                      & num2 & " is " & (num1 + num2) & ".")
End Sub

Sub ExplainPurpose()
  'Explain the task performed by the program
  lstResult.Items.Add("This program displays a sentence")
  lstResult.Items.Add("identifying two numbers and their sum.")
End Sub
```

Sub procedures make a program easy to read, modify, and debug. The event procedure gives a description of what the program does, and the Sub procedures fill in the details. Another benefit of Sub procedures is that they can be called several times during the execution of the program. This feature is especially useful when there are many statements in the Sub procedure.

Example 3.

(This item is displayed on pages 134 - 135 in the print version)

The following extension of the program in Example 2 displays several sums:

```
Private Sub btnAdd_Click(...) Handles btnAdd.Click
  'Display the sum of two numbers
  lstResult.Items.Clear()
```

```
  ExplainPurpose()
  lstResult.Items.Add("")
  DisplaySum(2, 3)
  DisplaySum(4, 6)
  DisplaySum(7, 8)
End Sub

Sub DisplaySum(ByVal num1 As Double, ByVal  num2 As Double)
```

```
  'Display numbers and their sum
  lstResult.Items.Add("The sum of " & num1 & " and "_
                      & num2 & " is " & num1 + num2 & ".")
End Sub

Sub ExplainPurpose()
  'Explain the task performed by the program
  lstResult.Items.Add("This program displays a sentence")
  lstResult.Items.Add("identifying two numbers and their sum.")
End Sub
```

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

```
This program displays sentences
identifying pairs of numbers and their sums.
The sum of 2 and 3 is 5.
The sum of 4 and 6 is 10.
The sum of 7 and 8 is 15.
```

The variables num1 and num2 appearing in the Sub procedure DisplaySum are called parameters. They are merely temporary place holders for the numbers passed to the Sub procedure; their names are not important. The only essentials are their type, quantity, and order. In this DisplaySum Sub procedure, the parameters must be numeric variables of type Double and there must be two of them. For instance, the Sub procedure could have been written

```
Sub DisplaySum(ByVal this As Double, ByVal  that As Double)
  'Display numbers and their sum
  lstResult.Items.Add("The sum of " & this & " and " _
                      & that & " is " & this + that & ".")
End Sub
```

When a parameter is defined in a Sub procedure, it is automatically available to the code between the Sub and End Sub lines. That is, the code "this As Double" that defines a parameter behaves similarly to the "Dim this As Double" code that defines a variable. A string also can be passed to a Sub procedure. In this case, the receiving parameter in the Sub procedure must be followed by the type declaration "As String".

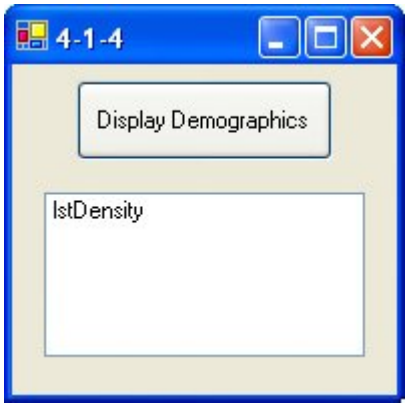
Example 4.

(This item is displayed on pages 135 - 136 in the print version)

The following program passes a string and two numbers to a Sub procedure. When the Sub procedure is first called, the string parameter state is assigned the value "Hawaii", and the numeric parameters pop and area are assigned the values 1257608 and 6471, respectively. The Sub procedure then uses these parameters to carry out the task of calculating the population density of Hawaii. The second call statement assigns different values to the

parameters.

[Page 136]



Object	Property	Setting
frmDensities	Text	4-1-4
btnDisplay	Text	Display Demographics
lstDensity		

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Calculate the population densities of states
  lstDensity.Items.Clear()
  CalculateDensity("Hawaii", 1257608, 6471)
  lstDensity.Items.Add("")
  CalculateDensity("Alaska", 648818, 591000)
End Sub

Sub CalculateDensity(ByVal state As String, _
                     ByVal pop As Double, ByVal area As Double)
  Dim rawDensity, density As Double
  'The density (number of people per square mile)
  'will be displayed rounded one decimal place
  rawDensity = pop / area
  density = Math.Round(rawDensity, 1)      'Round to one decimal place
  lstDensity.Items.Add("The density of " & state & " is " & density)
  lstDensity.Items.Add("people per square mile.")
End Sub
```

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

```
The density of Hawaii is 194.3
people per square mile.
The density of Alaska is 1.1
people per square mile.
```

The parameters in the density program can have any valid variable names, as with the parameters in the addition program of Example 3. The only restriction is that the first parameter be a string variable and that the last two parameters have type Double. For instance, the Sub procedure could have been written

```
Sub CalculateDensity(ByVal x As String, _
                     ByVal y As Double, ByVal z As Double)
  'The density (number of people per square mile)
  'will be displayed rounded to a whole number
```



```
  Dim rawDensity, density As Double
  rawDensity = y / z
  density = Math.Round(rawDensity, 1)     'Round to one decimal place
  lstDensity.Items.Add("The density of " & x & " is " & density)
  lstDensity.Items.Add("people per square mile.")
End Sub
```

When nondescriptive names are used for parameters, the Sub procedure should contain comments giving the meanings of the variables. Possible comments for the preceding program are

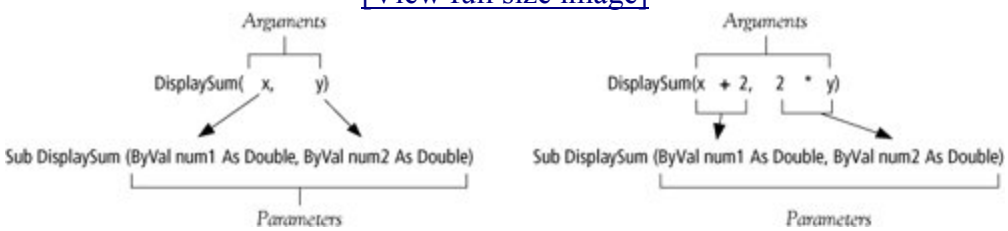
```
'x    name of the state
'y    population of the state
'z    area of the state
```

Variables and Expressions as Arguments

The items appearing in the parentheses of a call statement are called arguments. These should not be confused with parameters, which appear in the header of a Sub procedure. Each parameter defined for a Sub procedure corresponds to an argument passed in a call statement for that procedure. In Example 3, the arguments of the DisplaySum statements were literals. These arguments also could have been variables or expressions. For instance, the event procedure could have been written as follows. See Figure 4.1.

Figure 4.1. Passing arguments to parameters.

[View full size image]




```
Private Sub btnAdd_Click(...) Handles btnAdd.Click
  'Display the sum of two numbers
```

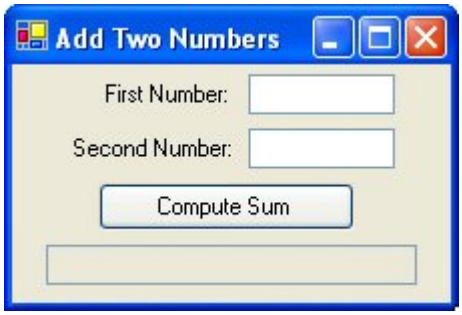
```
  Dim x, y As Double
  lstResult.Items.Clear()
  ExplainPurpose()
  lstResult.Items.Add("")
  x = 2
  y = 3
  DisplaySum(x, y)
  DisplaySum(x + 2, 2 * y)
  z = 7
  DisplaySum(z, z + 1)
End Sub
```

This feature allows values obtained as input from the user to be passed to a Sub procedure.

[Page 138]

Example 5.

The following variation of the addition program requests the two numbers as input from the user. Notice that the names of the arguments, x, and y, are different from the names of the parameters. The names of the arguments and parameters may be the same or different; what matters is that the order, number, and types of the arguments and parameters match.

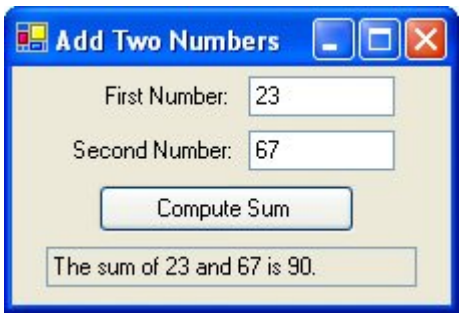


Object	Property	Setting
frmAdd	Text	Add Two Numbers
lblFirstNum	Text	First Number:
txtFirstNum		
lblSecondNum	Text	Second Number:
txtSecondNum		
btnCompute	Text	Compute Sum
txtResult	ReadOnly	True

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  'This program requests two numbers and
  'displays the two numbers and their sum.
  Dim x, y As Double
  x = CDbl(txtFirstNum.Text)
  y = CDbl(txtSecondNum.Text)
  DisplaySum(x, y)
End Sub

Sub DisplaySum(ByVal num1 As Double, ByVal  num2 As Double)
  'Display numbers and their sum
  txtResult.Text = "The sum of " & num1 & " and " & num2 _
                   & " is " & (num1 + num2) & "."
End Sub
```

[Run, type 23 and 67 into the text boxes, and then click the button.]



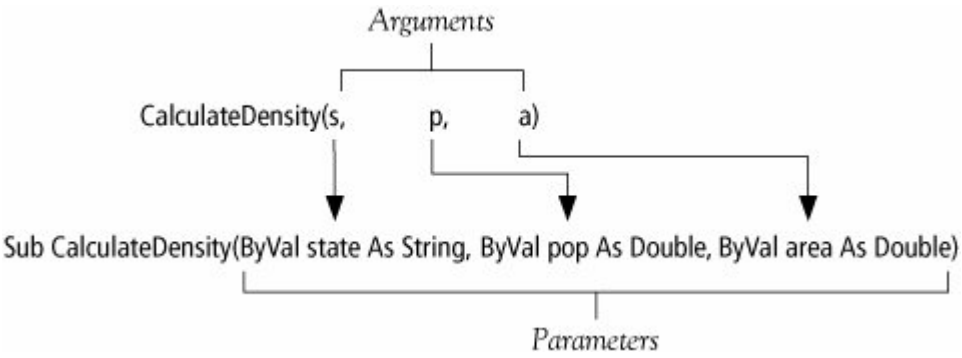
Example 6.

(This item is displayed on pages 138 - 140 in the print version)

The following variation of Example 4 obtains its input from the file DEMOGRAPHICS.TXT. The second call statement uses different variable names for the arguments to show that using the same argument names is not necessary. See Figure 4.2.

[Page 139]

Figure 4.2. Passing arguments to parameters in Example 6.



DEMOGRAPHICS.TXT contains the following lines:

Hawaii

1257608

6471

Alaska

648818

591000

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Calculate the population densities of states
  Dim state As String, pop, area As Double
  Dim s As String, p, a As Double
  Dim sr As IO.StreamReader = IO.File.OpenText("DEMOGRAPHICS.TXT")
  lstDensity.Items.Clear()
  state = sr.ReadLine
  pop = CDbl(sr.ReadLine)
  area = CDbl(sr.ReadLine)
  CalculateDensity(state, pop, area)
  lstDensity.Items.Add("")
  s = sr.ReadLine
  p = CDbl(sr.ReadLine)
  a = CDbl(sr.ReadLine)
  sr.Close()
  CalculateDensity(s, p, a)
End Sub

Sub CalculateDensity(ByVal state As String, _
                     ByVal pop As Double, ByVal area As Double)
  'The density (number of people per square mile)
  'will be displayed rounded to one decimal place
  Dim rawDensity, density As Double
  rawDensity = pop / area
  density = Math.Round(rawDensity, 1)      'Round to one decimal place
  lstDensity.Items.Add("The density of " & state & " is " & density)
  lstDensity.Items.Add("people per square mile.")
```

```
End Sub
```

[Page 140]

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

```
The density of Hawaii is 194.3
people per square mile.
The density of Alaska is 1.1
people per square mile.
```

Sub Procedures Calling Other Sub Procedures

A Sub procedure can call another Sub procedure. If so, after the End Sub of the called Sub procedure is reached, execution continues with the line in the calling Sub procedure that follows the call statement.

Example 7.

In the following program, the Sub procedure FirstPart calls the Sub procedure SecondPart. After the statements in SecondPart are executed, execution continues with the remaining statements in the Sub procedure FirstPart before returning to the event procedure. The form contains a button and a list box.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Demonstrate Sub procedure calling other Sub procedures
  FirstPart()
  lstOutput.Items.Add(4)
End Sub

Sub FirstPart()
  lstOutput.Items.Add(1)
  SecondPart()
  lstOutput.Items.Add(3)
End Sub

Sub SecondPart()
  lstOutput.Items.Add(2)
End Sub
```

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

```
1
2
3
```

```
4
```

Arguments and parameters also can be used to pass values from Sub procedures back to event procedures or other Sub procedures. This important property is explored in detail in the next section.

Comments

1. Sub procedures allow programmers to focus on the main flow of a complex task and defer the details of implementation. Modern programs use them liberally. This method of program construction is known as modular or top-down design. As a rule, a Sub procedure should perform only one task, or several closely related tasks, and should be kept relatively small.

[Page 141]

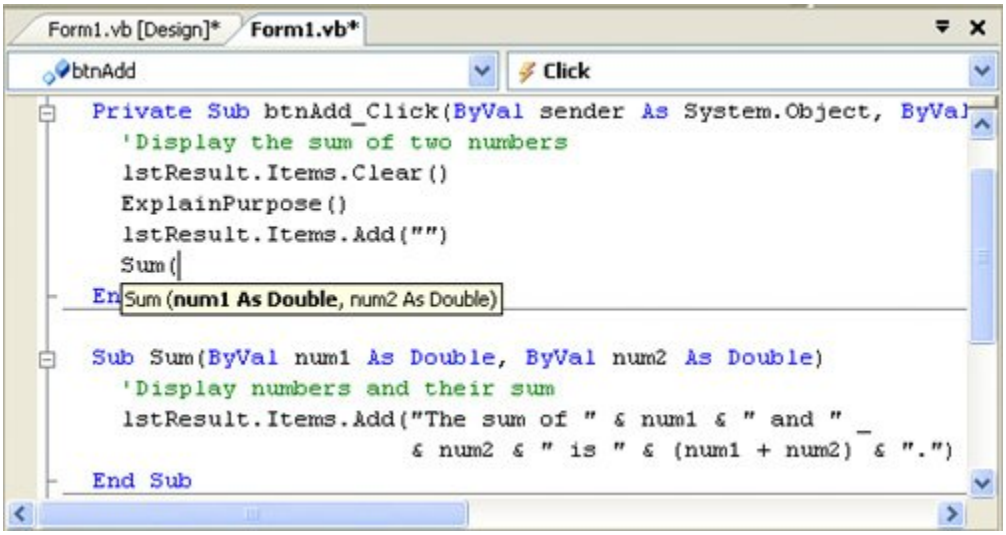
2. In this text, Sub procedure names begin with uppercase letters in order to distinguish them from variable names. Like variable names, however, they can be written with any combination of upper- and lowercase letters. Note: Parameters appearing in a Sub statement are not part of the Sub procedure name.

3. The first line inside a Sub procedure is often a comment statement describing the task performed by the Sub procedure. If necessary, several comment statements are devoted to this purpose. Conventional programming practice also recommends that all variables used by the Sub procedure be listed in comment statements with their meanings. In this text, we give several examples of this practice, but adhere to it only when the variables are especially numerous or lack descriptive names.

4. After a Sub procedure has been defined, Visual Basic automatically reminds you of the Sub procedure's parameters when you type in a call statement. As soon as you type in the left parenthesis of a call statement, a Parameter Info banner appears giving the names and types of the parameters. See Figure 4.3.

Figure 4.3. The Parameter Info help feature.

[View full size image]



Practice Problems 4.1

1. What is the difference between an event procedure and a Sub procedure?
2. What is wrong with the following code?

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim phone As String
  phone = mtxtPhoneNum.Text
  AreaCode(phone)
End Sub

Sub AreaCode()
  txtOutput.Text = "Your area code is " & phone.Substring(1, 3)
End Sub
```

[Page 142]

Exercises 4.1

In Exercises 1 through 34, determine the output displayed when the button is clicked

1.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Quote from Kermit
  Quotation()
  lstOutput.Items.Add ("Kermit the frog")
End Sub

Sub Quotation()
  'Display a quotation
```

```
  lstOutput.Items.Add("Time's fun when you're having flies.")
End Sub
```

2.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  lstOutput.Items.Add("Today")
  WhatDay()
  lstOutput.Items.Add("of the rest of your life.")
End Sub

Sub WhatDay()
  lstOutput.Items.Add("is the first day")
End Sub
```

3.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Question()
  Answer()
End Sub

Sub Answer()
  lstOutput.Items.Add("Since they were invented in the northern")
  lstOutput.Items.Add("hemisphere where sundials go clockwise.")
End Sub

Sub Question()
  lstOutput.Items.Add("Why do clocks run clockwise?")
  lstOutput.Items.Add("")
End Sub
```

4.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  FirstName()
  lstOutput.Items.Add("How are you today?")
End Sub

Sub FirstName()
  Dim name As String
  name = InputBox("What is your first name?", "Name")
  lstOutput.Items.Add("Hello "& name.ToUpper)
End Sub
```

(Assume that the response is George.)

[Page 143]

5.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'The fates of Henry the Eighth's six wives
  CommonFates()
  lstOutput.Items.Add("died,")
```

```
  CommonFates()
  lstOutput.Items.Add("survived")
End Sub

  Sub CommonFates()
  'The most common fates
  lstOutput.Items.Add("divorced")
  lstOutput.Items.Add("beheaded")
End Sub
```

6.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  lstOutput.Items.Add("a rose")
  Rose()
  Rose()
End Sub

Sub Rose()
  lstOutput.Items.Add("is a rose")
End Sub
```

7.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Good advice to follow
  Advice()
End Sub

Sub Advice()
  lstOutput.Items.Add("Keep cool, but don't freeze.")
  Source()
End Sub

Sub Source()
  lstOutput.Items.Add("Source: A jar of mayonnaise.")
End Sub
```

8.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Answer()
  Question()
End Sub

Sub Answer()
  lstOutput.Items.Add("The answer is 9W.")
  lstOutput.Items.Add("What is the question?")
End Sub

Sub Question()
  'Note: "Wagner"is pronounced "Vagner"
  lstOutput.Items.Add("Do you spell your name with a V,")
  lstOutput.Items.Add("Mr. Wagner?")
End Sub
```

[Page 144]

9.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Piano(88)
End Sub

Sub Piano(ByVal num As Integer)
  txtOutput.Text = num & " keys on a piano"
End Sub
```

10.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Opening line of Moby Dick
  FirstLine("Ishmael")
End Sub

Sub FirstLine(ByVal name As String)
  'Display first line
  txtOutput.Text = "Call me "& name & "."
End Sub
```

11.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Beginning of Tale of Two Cities
  Times("best")
  Times("worst")
End Sub

Sub Times(ByVal word As String)
  'Display sentence
  lstOutput.Items.Add("It was the "& word & " of times.")
End Sub
```

12.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Potato(1)
  Potato(2)
  Potato(3)
  lstOutput.Items.Add(4)
End Sub

Sub Potato(ByVal quantity As Integer)
  lstOutput.Items.Add(quantity & " potato")
End Sub
```

13.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Analyze a name
```

```
  Dim name As String = "Gabriel"
  AnalyzeName(name)
End Sub

Sub AnalyzeName(ByVal name As String)
  'Display length and first letter
  lstBox.Items.Add("Your name has "& name.Length & " letters.")
  lstBox.Items.Add("The first letter is "& name.Substring(0, 1))
End Sub
```

[Page 145]

14.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim color As String
  color = InputBox("What is your favorite color?")
  Flattery(color)
End Sub

Sub Flattery(ByVal color As String)
  txtOutput.Text = "You look dashing in "& color & "."
End Sub
```

(Assume that the response is blue.)

15.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim num As Integer
  num = CInt(InputBox("Give a number from 1 to 26."))
  Alphabet(num)
End Sub

Sub Alphabet(ByVal num As Integer)
  txtOutput.Text = "abcdefghijklmnopqrstuvwxyz".Substring(0, num)
End Sub
```

(Assume that the response is 5.)

16.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim size As Double
  size = 435
  House(size)
  lstOutput.Items.Add("of Representatives")
End Sub

Sub House(ByVal size As Double)
  lstOutput.Items.Add(size & " members in the House")
End Sub
```

17.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
```

```
  Dim num As Double
  num = 144
  Gross(num)
End Sub

Sub Gross(ByVal amount As Double)
  txtOutput.Text = amount & " items in a gross"
End Sub
```

18.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim a As String = "mile"
  Acres(a)
End Sub

Sub Acres(ByVal length As String)
  txtOutput.Text = "640 acres in a square "& length
End Sub
```

[Page 146]

19.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim candy As String
  candy = "M&M's Plain Chocolate Candies"
  Brown(candy)
End Sub

Sub Brown(ByVal item As String)
  txtOutput.Text = "30% of "& item & " are brown."
End Sub
```

20.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim annualRate As Double = 0.08
  Balance(annualRate)
End Sub

Sub Balance(ByVal r As Double)
  Dim p As Double
  p = CDbl(InputBox("What is the principal?"))
  txtOutput.Text = "The balance after 1 year is "& (1 + r) * p
End Sub
```

(Assume that the response is 100.)

21.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim hours As Double
  hours = 24
  Minutes(60 * hours)
```

```
End Sub

Sub Minutes(ByVal num As Double)
  txtOutput.Text = num & " minutes in a day"
End Sub
```

22.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim a, b As String
  a = "United States"
  b = "acorn"
  Display(a.Substring(0, 3) & b.Substring(1, 4))
End Sub

Sub Display(ByVal word As String)
  txtOutput.Text = word
End Sub
```

23.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim word As String
  word = InputBox("Enter a word.")
  T(word.IndexOf("t"))
End Sub

Sub T(ByVal num As Integer)
  txtBox.Text = "t is the "& (num + 1) & "th letter of the word."
End Sub
```

(Assume that the response is computer.)

[Page 147]

24.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim states, senators As Double
  states = 50
  senators = 2
  Senate(states * senators)
End Sub

Sub Senate(ByVal num As Double)
  txtBox.Text = "The number of U.S. Senators is "& num
End Sub
```

25.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  DisplaySource()
  Database("Sybase SQL Server", 75633)
  Database("Oracle", 73607)
  Database("Windows CE", 73375)
```

```
  Database("Microsoft SQL Server", 68295)
End Sub

Sub DisplaySource()
  lstOutput.Items.Add("A recent salary survey of readers of")
  lstOutput.Items.Add("Visual Basic Programmer's Journal gave")
  lstOutput.Items.Add("average salaries of database developers")
  lstOutput.Items.Add("according to the database used.")
  lstOutput.Items.Add("")
End Sub

Sub Database(ByVal db As String, ByVal salary As Double)
  lstOutput.Items.Add(db & " programmers earned "& _
             FormatCurrency(salary, 0) & ".")
End Sub
```

26.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Sentence using number, thing, and place
  Sentence(168, "hour", "a week")
  Sentence(76, "trombone", "the big parade")
End Sub

Sub Sentence(ByVal num As Double, ByVal thing As String, _
             ByVal where As String)
  lstOutput.Items.Add(num & " "& thing & "s in "& where)
End Sub
```

27.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim pres, college As String
  Dim sr As IO.StreamReader = IO.File.OpenText("CHIEF.TXT")
  pres = sr.ReadLine
  college = sr.ReadLine
  PresAlmaMater(pres, college)
  pres = sr.ReadLine
  college = sr.ReadLine
```

[Page 148]

```
  PresAlmaMater(pres, college)
  sr.Close()
End Sub

Sub PresAlmaMater(ByVal pres As String, ByVal college As String)
  lstBox.Items.Add("President "& pres & " is a graduate of "_
               & college & ".")
End Sub
```

(Assume that the four lines of the file CHIEF.TXT contain the following data: Clinton, Georgetown University, Bush, Yale University.)

28.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim name As String, yob As Integer
```

```
  name = InputBox("Name?")
  yob = CInt(InputBox("Year of birth?"))
  AgeIn2010(name, yob)
End Sub

Sub AgeIn2010(ByVal name As String, ByVal yob As Integer)
  txtBox.Text = name & ", in the year 2010 your age will be "_
        & (2010  yob) & "."
End Sub
```

(Assume that the responses are Gabriel and 1980.)

29.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim word As String, num As Integer
  word = "Visual Basic"
  num = 6
  FirstPart(word, num)
End Sub

Sub FirstPart(ByVal term As String, ByVal digit As Integer)
  txtOutput.Text = "The first "& digit & " letters are "_
                   & term.Substring(0, digit) & "."
End Sub
```

30.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim statue As String, tons As Double
  statue = "The Statue of Liberty"
  tons = 250
  HowHeavy(statue, tons)
End Sub

Sub HowHeavy(ByVal what As String, ByVal weight As Double)
  txtOutput.Text = what & " weighs "& weight & " tons."
End Sub
```

31.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim word As String
  word = "worldly"
  Negative("un" & word, word)
End Sub
```

[Page 149]

```
Sub Negative(ByVal neg As String, ByVal word As String)
  txtOutput.Text = "The negative of " & word & " is " & neg

End Sub
```

32.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim age, yrs As Integer, major As String
```

```
  age = CInt(InputBox("How old are you?"))
  yrs = CInt(InputBox("In how many years will you graduate?"))
  major = InputBox("What sort of major do you have "& _
                   "(Arts or Sciences)?")
  Graduation(age + yrs, major.Substring(0, 1))
End Sub

Sub Graduation(ByVal num As Integer, ByVal letter As String)
  txtOutput.Text = "You will receive a B"& letter.ToUpper & _
                  " degree at age "& num
End Sub
```

(Assume that the responses are 19, 3, and arts.)

33.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  HowMany(24)
  lstOutput.Items.Add("a pie.")
End Sub

Sub HowMany(ByVal num As Integer)
  What(num)
  lstOutput.Items.Add("baked in")
End Sub

Sub What(ByVal num As Integer)
  lstOutput.Items.Add(num & " blackbirds")
End Sub
```

34.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  txtOutput.Text = "All's"
  PrintWell()
  PrintWords(" that ends")
  PrintWell()
  txtOutput.Text &= "."
End Sub

Sub PrintWell()
  txtOutput.Text &= " well"
End Sub

Sub PrintWords(ByVal words As String)
  txtOutput.Text &= words
End Sub
```

[Page 150]

In Exercises 35 through 38, find the errors.

35.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim n As Integer = 5
  Alphabet()
End Sub

Sub Alphabet(ByVal n As Integer)
  txtOutput.Text = "abcdefghijklmnopqrstuvwxyz".Substring(0, n)
End Sub
```

36.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim word As String, number As Double
  word = "seven"
  number = 7
  Display(word, number)
End Sub

Sub Display(ByVal num As Double, ByVal term As String)
  txtOutput.Text = num & " "& term
End Sub
```

37.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim name As String
  name = InputBox("Name")
  Handles(name)
End Sub

Sub Handles(ByVal moniker As String)
  txtOutput.Text = "Your name is "& moniker
End Sub
```

38.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim num As Integer = 2
  Tea(num)
End Sub

Sub Tea()
  txtOutput.Text = "Tea for "& num
End Sub
```

In Exercises 39 through 42, rewrite the program with the output performed by a call to a Sub procedure.

39.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Display a lucky number
  Dim num As Integer = 7
  txtOutput.Text = num & " is a lucky number."
End Sub
```

40.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Greet a friend
  Dim name As String = "Jack"
  txtOutput.Text = "Hi, "& name
End Sub
```

[Page 151]

41.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Information about trees
  Dim tree As String, ht As Double
  Dim sr As IO.StreamReader = IO.File.OpenText("TREES.TXT")
  lstBox.Items.Clear()
  tree = sr.ReadLine
  ht = CDbl(sr.ReadLine)
  lstBox.Items.Add("The tallest "& tree & " in the U.S. is "_
                   & ht & " feet.")
  tree = sr.ReadLine
  ht = CDbl(sr.ReadLine)
  lstBox.Items.Add("The tallest "& tree & " in the U.S. is "_
                   & ht & " feet.")
  sr.Close()
End Sub
```

(Assume that the four lines of the file TREES.TXT contain the following data: redwood, 362, pine, 223.)

42.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim city As String, salary As Double
  Dim sr As IO.StreamReader = IO.File.OpenText("DATA.TXT")
  lstBox.Items.Clear()
  city = sr.ReadLine
  salary = CDbl(sr.ReadLine)
  lstBox.Items.Add("In 2000, the average salary for "& city & _
                   " residents was "& FormatCurrency(salary, 0))
  city = sr.ReadLine
  salary = CDbl(sr.ReadLine)
  lstBox.Items.Add("In 2000, the average salary for "& city & _
                   " residents was "& FormatCurrency(salary, 0))
  sr.Close()
End Sub
```

(Assume that the four lines of the file DATA.TXT contain the following data: San Jose, 76076, Hartford, 42349.) Note: San Jose is located in Silicon Valley, and Hartford is the center of the insurance industry.

43. Write a program that requests a number as input and displays three times the number. The output should be produced by a call to a Sub procedure named Triple.

44. Write a program that requests a word as input and displays the word followed by the number of letters in the word. The output should be produced by a call to a Sub procedure named HowLong.

45. Write a program that requests a word of at most ten letters and a width from 10 through 20 as input and displays the word right-justified in a zone having the specified width. The output should be produced by a call to a Sub procedure named PlaceNShow.

46. Write a program that requests three numbers as input and displays the average of the three numbers. The output should be produced by a call to a Sub procedure named Average.

[Page 152]

In Exercises 47 through 50, write a program that, when btnDisplay is clicked, will display in lstOutput the output shown. The last two lines of the output should be displayed by one or more Sub procedures using data passed by variables from an event procedure.

47. (Assume that the following is displayed.)

```
According to a 2004 survey of college freshmen
taken by the Higher Educational Research Institute:

16 percent said they intend to major in business.
1.4 percent said they intend to major in computer science.
```

48. (Assume that the current date is 12/31/2006, the label for txtBox reads "What is your year of birth?", and the user types 1980 into txtBox before btnDisplay is clicked.)

```
You are now 26 years old.
You have lived for more than 9490 days.
```

49. (Assume that the label for txtBox reads "What is your favorite number?", and the user types 7 into txtBox before btnDisplay is clicked.)

```
The sum of your favorite number with itself is 14.
The product of your favorite number with itself is 49.
```

50. (Assume that the following is displayed.)

```
In a recent year,
657 thousand college students took a course in Spanish
```

```
199 thousand college students took a course in French
```

51. Write a program to display three verses of "Old McDonald Had a Farm." The primary verse, with variables substituted for the animals and sounds, should be contained in a Sub procedure. The program should use the file FARM.TXT. The eight lines of the file FARM.TXT contain the following data: lamb, baa, firefly, blink, computer, beep.

The first verse of the output should be

```
Old McDonald had a farm. Eyi eyi oh.
And on his farm he had a lamb. Eyi eyi oh.
With a baa baa here, and a baa baa there.
Here a baa, there a baa, everywhere a baa baa.
Old McDonald had a farm. Eyi eyi oh.
```

52. Write a program that displays the word WOW vertically in large letters. Each letter should be drawn in a Sub procedure. For instance, the Sub procedure for the letter W follows. Hint: Use the font Courier New in the list box.

```
Sub DrawW()
  'Draw the letter W
  lstWOW.Items.Add("**          **")
  lstWOW.Items.Add(" **        **")
  lstWOW.Items.Add("  **  **  **")
  lstWOW.Items.Add("   **    **")
  lstWow.Items.Add("")
End Sub
```

[Page 153]

53. Write a program to display the data from Table 4.1. The occupations and numbers of jobs for 2000 and 2012 should be contained in the file GROWTH.TXT. A Sub procedure, to be called four times, should read the first three pieces of data for an occupation, calculate the percent increase from 2000 to 2012, and display the four items. Note: The percent increase is calculated as (2012 value 2000 value)/(2000 value).

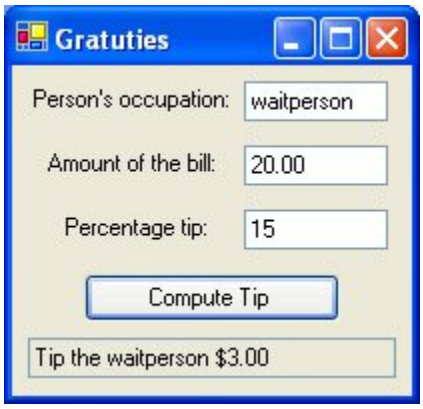
Table 4.1. Occupations projected to experience the largest job growth, 20002012 (numbers in thousands of jobs).

Occupation	2000	2012	Increase
Medical assistant	365	579	59%
Home health aide	580	859	48%
Software engineer	394	573	45%

Systems analyst	468	653	40%

Source: U.S. Department of Labor.

54. Write a program to compute tips for services rendered. The program should request the person's occupation, the amount of the bill, and the percentage tip as input and pass this information to a Sub procedure to display the person and the tip. A sample run is shown in the following figure:



Solutions to Practice Problems 4.1

1. An event procedure always has the two parameters sender and e and ends with a phrase of the form "Handles object.event." It is triggered when the specified object experiences the specified event. On the other hand, a Sub procedure is triggered by a line of code containing the name of the Sub procedure.

2. The statement `Sub AreaCode()` must be replaced by `Sub AreaCode(ByVal phone As String)`. Whenever a value is passed to a Sub procedure, the Sub statement must provide a parameter to receive the value.

PREV PREV NEXT NEXT

[Page 154]

4.2. Sub Procedures, Part II

The previous section introduced the concept of a Sub procedure, but left some questions unanswered. Why can't the value of a variable be passed from an event procedure to a Sub procedure by just using the variable in the Sub procedure? How do Sub procedures pass values back to an event procedure? The answers to these questions provide a deeper understanding of the workings of Sub procedures and reveal their full capabilities.

Passing by Value

In Section 4.1, all parameters appearing in Sub procedures were preceded by the word ByVal, which stands for "By Value." When a variable is passed to such a parameter, we say that the variable is "passed by value." A variable that is passed by value will retain its original value after the Sub procedure terminatesregardless of what was done to the corresponding parameter inside the Sub procedure. Example 1 illustrates this feature.

Example 1.

The following program illustrates the fact that changes to the value of a parameter passed by value have no effect on the value of the calling argument:

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Illustrate that a change in value of parameter does not alter the
  'value of the argument
  Dim amt As Double = 2
  lstResults.Items.Add(amt)
  Triple(amt)
  lstResults.Items.Add(amt)
End Sub
Sub Triple(ByVal num As Double)
  'Triple a number
  lstResults.Items.Add(num)
  num = 3 * num
  lstResults.Items.Add(num)
End Sub
```

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

```
2
2
6
2
```

When a variable is passed by value, two memory locations are involved. At the time the Sub procedure is called, a temporary second memory location for the parameter is set aside for the Sub procedure's use and the value of the argument is copied into that location. After the completion of the Sub procedure, the temporary memory location is released, and the value in it is lost. So, for instance, the outcome in Example 1 would be the same even if the name of the parameter were amt.

[Page 155]

Passing by Reference

Another way to pass a variable to a Sub procedure is "By Reference." In this case the parameter is preceded by the reserved word ByRef. Suppose a variable, call it arg, appears as an argument in a call statement, and its corresponding parameter in the Sub procedure's header, call it par, is preceded by ByRef. After the Sub procedure is executed, arg will have whatever value par had in the Sub procedure. Hence, not only is the value of arg passed to par, but the value of par is passed back to arg.

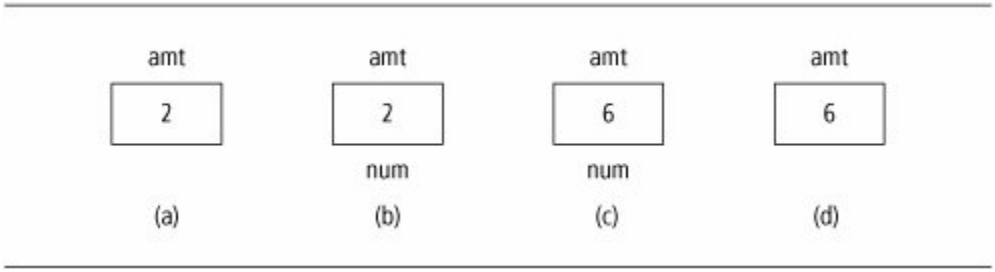
In Example 1, if the first line of the Sub procedure is changed to

```
Sub Triple(ByRef num As Double)
```

then the last number of the output will be 6.

Although this feature may be surprising at first glance, it provides a vehicle for passing values from a Sub procedure back to the place from which the Sub procedure was called. Different names may be used for an argument and its corresponding parameter, but only one memory location is involved. Initially, the btnDisplay_Click() event procedure allocates a memory location to hold the value of amt (Figure 4.4 (a)). When the Sub procedure is called, the parameter num becomes the Sub procedure's name for this memory location (Figure 4.4(b)). When the value of num is tripled, the value in the memory location becomes 6 (Figure 4.4(c)). After the completion of the Sub procedure, the parameter name num is forgotten; however, its value lives on in amt (Figure 4.4(d)). The variable amt is said to be passed by reference.

Figure 4.4. Passing a variable by reference to a Sub procedure.



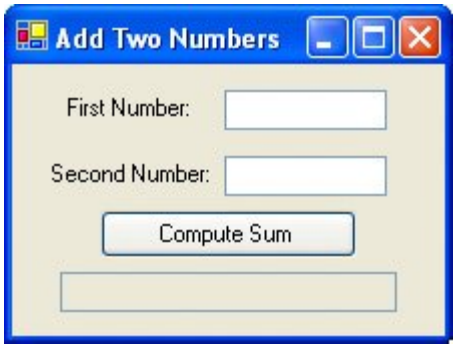
Passing by reference has a wide variety of uses. In the next example, it is used as a vehicle to transport a value from a Sub procedure back to an event procedure.

Example 2.

(This item is displayed on pages 155 - 156 in the print version)

The following variation of Example 5 from the previous section uses a Sub procedure to acquire the input. The variables x and y are not assigned values prior to the execution of the first call statement. Therefore, before the call statement is executed, they have the value 0. After the call statement is executed, however, they have the values entered into the text boxes. These values then are passed by the second call statement to the Sub procedure DisplaySum.

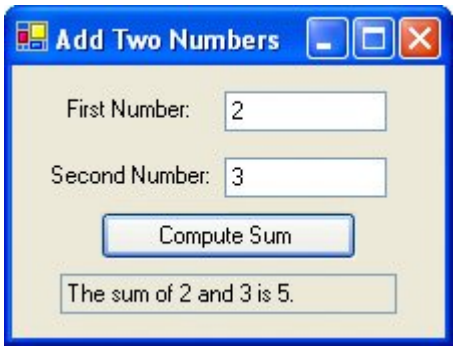
[Page 156]



Object	Property	Setting
frmAdd	Text	Add Two Numbers
lblFirstNum	Text	First Number:
txtFirstNum		
lblSecondNum	Text	Second Number:
txtSecondNum		
btnCompute	Text	Compute Sum
txtResult	ReadOnly	True

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  'This program requests two numbers and
  'displays the two numbers and their sum.
  Dim x, y As Double
  GetNumbers(x, y)
  DisplaySum(x, y)
End Sub
Sub GetNumbers(ByRef x As Double, ByRef y As Double)
  'Record the two numbers in the text boxes
  x = CDbl(txtFirstNum.Text)
  y = CDbl(txtSecondNum.Text)
End Sub
Sub DisplaySum(ByVal num1 As Double, ByVal  num2 As Double)
  'Display numbers and their sum
  txtResult.Text = "The sum of " & num1 & " and " & num2 _
                   & " is " & (num1 + num2) & "."
End Sub
```

[Run, type 2 and 3 into the text boxes, and then click the button.]



In most situations, a variable with no preassigned value is used as an argument of a call statement for the sole purpose of carrying back a value from the Sub procedure.

Example 3.

(This item is displayed on pages 156 - 157 in the print version)

The following variation of Example 2 allows the btnCompute_Click event procedure to be written in the input-process-output style. Just before the call statement `CalculateSum (x, y, t)` is executed, the value of t is 0. After the call, the value of t will be the sum of the two numbers in the text boxes.

[Page 157]

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  'This program requests two numbers and
  'displays the two numbers and their sum.
  Dim  x As Double       'First number
  Dim  y As Double       'Second number
  Dim  t As Double       'Total
  GetNumbers(x, y)
  CalculateSum(x, y, t)
  DisplayResult(x, y, t)
End Sub
Sub GetNumbers(ByRef num1 As Double, ByRef  num2 As Double)
  'Retrieve the two numbers in the text boxes
  num1 = CDbl(txtFirstNum.Text)
  num2 = CDbl(txtSecondNum.Text)
End Sub
Sub CalculateSum(ByVal num1 As Double, ByVal num2 As Double, _
                 ByRef total As Double)
  'Add the values of num1 and num2
  total = num1 + num2
End Sub
Sub DisplayResult(ByVal num1 As Double, ByVal num2 As Double, _
                  ByVal total As Double)
  txtResult.Text = "The sum of " & num1 & " and " & num2 _
                   & " is " & total & "."
End Sub
```

Visual Basic provides a way to override passing by reference, even if the ByRef keyword precedes the parameter. If you enclose the variable in the call statement in an extra pair of parentheses, then the variable will be passed by value.

For instance, in Example 1, if you change the call statement to

```
Triple((amt))
```

then the fourth number of the output will be 2 even if the parameter num is preceded with ByRef.

Local Variables

When a variable is declared in an event or Sub 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 executes, the memory location is freed up; that is, the variable ceases to exist. The variable is said to be local to the procedure.

When variables of the same name are declared with Dim statements in two different procedures (either event or Sub), Visual Basic gives the variables separate identities and treats them as two different variables. A value assigned to a variable in one part of the program will not affect the value of the like-named variable in the other part of the program.

[Page 158]

Example 4.

The following program illustrates the fact that each time a Sub procedure is called, its variables are set to their initial values; that is, numerical variables are set to 0 and string variables are set to the keyword Nothing.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Demonstrate that variables declared in a Sub procedure
  'do not retain their values in subsequent calls
  Three()
  Three()
End Sub
Sub Three()
  'Display the value of num and assign it the value 3
  Dim num As Double
  lstResults.Items.Add(num)
  num = 3
End Sub
```

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

```
0
0
```

Example 5.

(This item is displayed on pages 158 - 159 in the print version)

The following program illustrates the fact that variables are local to the part of the program in which they reside. The variable x in the event procedure and the variable x in the Sub procedure are treated as different variables. Visual Basic handles them as if their names were separate, such as xbtnDisplay_Click and xTrivial. Also, each time the Sub procedure is called, the value of variable x inside the Sub procedure is reset to 0.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Demonstrate the local nature of variables
  Dim x As Double = 2
  lstResults.Items.Clear()
  lstResults.Items.Add(x)
  Trivial()
  lstResults.Items.Add(x)
  Trivial()
  lstResults.Items.Add(x)
End Sub
Sub Trivial()
  'Do something trivial
  Dim x As Double
  lstResults.Items.Add(x)
  x = 3
  lstResults.Items.Add(x)
End Sub
```

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

[Page 159]

```
2
0
3
2
0
3
2
```

Class-Level Variables

Visual Basic provides a way to make a variable visible to every procedure in a form's code without being passed. Such a variable is called a class-level variable. 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 a 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 window.) A class-level variable is visible to every procedure. When a class-level variable has its value changed by a procedure, the value persists even after the procedure has finished executing. We say that such a variable has class-level scope. Variables declared inside a procedure are said to have local scope.

In general, the scope of a variable is the portion of the program that can refer to it. Class-level scope also is referred to as module-level scope, and local scope also is referred to as procedure-level scope. If a 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 6.

(This item is displayed on pages 159 - 160 in the print version)

The following program contains the class-level variables num1 and num2. Their Dim statement does not appear inside a procedure. It appears immediately following the statement `Public Class frmAdd`.

```
Dim num1, num2 As Double     'Class-level variables
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Display the sum of two numbers
  num1 = 2
  num2 = 3
  lstResults.Items.Clear()
  AddAndIncrement()
  lstResults.Items.Add("")
  lstResults.Items.Add("num1 = " & num1)
  lstResults.Items.Add("num2 = " & num2)
End Sub
Sub AddAndIncrement()
  'Display numbers and their sum
  lstResults.Items.Add("The sum of " & num1 & " and " &
                       num2 & " is " & (num1 + num2) & ".")
  num1 += 1       'Add 1 to the value of num1
  num2 += 1       'Add 1 to the value of num2
End Sub
```

[Page 160]

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

```
The sum of 2 and 3 is 5.
num1 = 3
num2 = 4
```

In the preceding example, we had to click a button to assign values to the class-level variables. In some situations, we want to assign a value immediately to a class-level variable, without requiring the user to perform some specific action. This can be accomplished by declaring each class-level variable with a statement of the type

```
Dim variableName As varType = value
```

Example 7.

The following program assigns a value to a class-level variable as soon as it is created:

```
Dim pi As Double = 3.14159
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  'Display the area of a circle of radius 5
  txtArea.Text = "The area of a circle of radius 5 is " & (pi * 5 * 5)
End Sub
```

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

```
The area of a circle of radius 5 is 78.53975
```

Debugging

Programs with Sub procedures are easier to debug. Each Sub procedure can be checked individually before being placed into the program.

In Appendix D, the section "Stepping through a Program Containing a General Procedure: Chapter 4" uses the Visual Basic debugger to trace the flow through a program and observe the interplay between arguments and parameters.

Practice Problems 4.2

1. What does the following code display in the list box when the button is clicked?

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
```

```
  Dim b As Integer = 1, c As Integer = 2
  Rhyme()
  lstOutput.Items.Add(b & " " & c)
End Sub
Sub Rhyme()
  Dim b, c As Integer
  lstOutput.Items.Add(b & " " & c & " buckle my shoe.")
  b = 3
End Sub
```

[Page 161]

2. Determine the output displayed in the list box when the button is clicked.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim amt1 As Integer = 1, amt2 As Integer = 2
  lstOutput.Items.Add(amt1 & " "& amt2)
  Swap(amt1, amt2)
  lstOutput.Items.Add(amt1 & " "& amt2)
End Sub
Sub Swap(ByRef num1 As Integer, ByRef num2 As Integer)
  'Interchange the values of num1 and num2
  Dim temp As Integer
  temp = num1
  num1 = num2
  num2 = temp
  lstOutput.Items.Add(num1 & " "& num2)
End Sub
```

3. In Problem 2, change the Sub statement to

```
Sub Swap(ByRef num1 As Integer, ByVal num2 As Integer)
```

and determine the output.

4. In Problem 2, change the calling statement to

```
Swap((amt1), (amt2))
```

and determine the output.

Exercises 4.2

In Exercises 1 through 18, determine the output displayed when the button is clicked.

1.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    Dim num As Double = 7
    AddTwo(num)
    txtOutput.Text = CStr(num)
  End Sub
  Sub AddTwo(ByRef num As Double)
    'Add 2 to the value of num
    num += 2
  End Sub
```

2.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim term As String
  term = "Fall"
  Plural(term)
  txtOutput.Text = term
End Sub
```

[Page 162]

```
Sub Plural(ByRef term As String)
  'Concatenate the letter "s"to the value of term
  term &= "s"
End Sub
```

3.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim dance As String
  dance = "Can "
  Twice(dance)
  txtOutput.Text = dance
End Sub

Sub Twice(ByRef dance As String)
  'Concatenate the value of dance to itself
  dance &= dance
End Sub
```

4.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim a As Integer = 1, b As Integer = 3
  lstOutput.Items.Add(a & " "& b)
  Combine(a, b)
  lstOutput.Items.Add(a & " "& b)
  Combine((a), b)
  lstOutput.Items.Add(a & " "& b)
End Sub

Sub Combine(ByRef x As Integer, ByVal y As Integer)
  x = y - x
  y = x + y
  lstOutput.Items.Add(x & " "& y)
End Sub
```

5.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim a As Double = 5
  Square(a)
  txtOutput.Text = CStr(a)
End Sub

Sub Square(ByRef num As Double)
  num = num * num
End Sub
```

6.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim state As String = "NEBRASKA"
  Abbreviate(state)
  txtOutput.Text = state
End Sub

Sub Abbreviate(ByRef a As String)
  a = a.SubString(0, 2)
End Sub
```

[Page 163]

7.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim word As String = " "
  GetWord(word)
  txtOutput.Text = "Less is "& word & "."
End Sub

Sub GetWord(ByRef w As String)
  w = "more"
End Sub
```

8.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim hourlyWage, annualWage As Double
  hourlyWage = 10
  CalcAnnualWage(hourlyWage, annualWage)
  txtOutput.Text = "Approximate Annual Wage: "& _
                   FormatCurrency(annualWage)
End Sub

Sub CalcAnnualWage(ByVal hWage As Double, ByRef aWage As Double)
  aWage = 2000 * hWage
End Sub
```

9.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim name As String = "", yob As Integer
  GetVita(name, yob)
  txtOutput.Text = name & " was born in the year "& yob
End Sub

Sub GetVita(ByRef name As String, ByRef yob As Integer)
  name = "Gabriel"
  yob = 1980   'Year of birth
End Sub
```

10.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim word1, word2 As String
  word1 = "fail"
  word2 = "plan"
  txtOutput.Text = "If you "
  Sentence(word1, word2)
  txtOutput.Text &= ","
  Exchange(word1, word2)
  txtOutput.Text &= " then you "
  Sentence(word1, word2)
  txtOutput.Text &= "."
End Sub

Sub Exchange(ByRef word1 As String, ByRef word2 As String)
  Dim temp As String
  temp = word1
  word1 = word2
  word2 = temp
End Sub
```

[Page 164]

```
Sub Sentence(ByVal word1 As String, ByVal word2 As String)
  txtOutput.Text &= word1 & " to "& word2
End Sub
```

11.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim state As String = "Ohio "
  Team()
End Sub

Sub Team()
  Dim state As String
  txtOutput.Text = state
  vtxtOutput.Text &= "Buckeyes"
End Sub
```

12.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim a As Double = 5
```

```
  Multiply(7)
  lstOutput.Items.Add(a * 7)
End Sub

Sub Multiply(ByRef num As Double)
  Dim a As Double
  a = 11
  lstOutput.Items.Add(a * num)
End Sub
```

13.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim a As Double = 5
  Multiply(7)
End Sub

Sub Multiply(ByVal num As Double)
  Dim a As Double
  txtOutput.Text = CStr(a * num)
End Sub
```

14.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim name, n As String
  name = "Ray"
  Hello(name)
  lstOutput.Items.Add(n & " and "& name)
End Sub

Sub Hello(ByRef name As String)
  Dim n As String
  n = name
  name = "Bob"
  lstOutput.Items.Add("Hello "& n & " and "& name)
End Sub
```

[Page 165]

15.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim num As Double = 1
  Amount(num)
  Amount(num)
End Sub

Sub Amount(ByVal num As Double)
  Dim total As Double
  total += num 'Add the value of num to the value of total
  lstOutput.Items.Add(total)
End Sub
```

16.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim river As String
  river = "Wabash"
  Another()
  lstOutput.Items.Add(river & " River")
  Another()
End Sub

Sub Another()
  Dim river As String
  lstOutput.Items.Add(river & " River")
  river = "Yukon"
End Sub
```

17.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim n As Integer = 4, word As String = "overwhelming"
  lstOutput.Items.Add(n & " "& word)
  Crop(n, word)
  lstOutput.Items.Add(n & " "& word)
  Crop(n, (word))
  lstOutput.Items.Add(n & " "& word)
End Sub

Sub Crop(ByVal n As Integer, ByRef word As String)
  n = word.Length - n
  word = word.Substring(word.Length - n)
  lstOutput.Items.Add(n & " "& word)
End Sub
```

18.
```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  Dim tax, price, total As Double
  tax = 0.05
  GetPrice("bicycle", price)
  ProcessItem(price, tax, total)
  DisplayResult(total)
End Sub

Sub DisplayResult(ByVal total As Double)
  txtOutput.Text = "With tax, price is "& FormatCurrency(total)
End Sub
```

[Page 166]

```
Sub GetPrice(ByVal item As String, ByRef price As Double)
  Dim strVar As String
  strVar = InputBox("What is the price of a "& item & "?")
  price = CDbl(strVar)
End Sub

Sub ProcessItem(ByVal price As Double, ByVal tax As Double, _
                ByRef total As Double)
  total = (1 + tax) * price
End Sub
```

(Assume that the cost of the bicycle is $200.)

In Exercises 19 and 20, find the errors.

19.
```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  Dim a, b, c As Double
  a = 1
  b = 2
  Sum(a, b, c)
  txtOutput.Text = "The sum is "& c
End Sub

Sub Sum(ByVal x As Double, ByVal y As Double)
  Dim c As Double
  c = x + y
End Sub
```

20.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim ano As String = ""
  GetYear(ano)
  txtOutput.Text = ano
End Sub

Sub GetYear(ByRef yr As Double)
  yr = 2006
End Sub
```

In Exercises 21 through 24, rewrite the program so input, processing, and output are each performed by calls to Sub procedures.

21.
```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  'Calculate sales tax
  Dim price, tax, cost As Double
  lstOutput.Items.Clear()
  price = CDbl(InputBox("Enter the price of the item:"))
  tax = .05 * price
  cost = price + tax
  lstOutput.Items.Add("Price:  "& price)
  lstOutput.Items.Add("Tax:    "& tax)
  lstOutput.Items.Add("-------")
  lstOutput.Items.Add("Cost:  "& cost)
End Sub
```

[Page 167]

22.
```
Private Sub btnDisplay_Click(...) Handles bnDisplay.Click
  'Letter of acceptance
  Dim name, firstName As String, n As Integer
  lstOutput.Items.Clear()
  name = InputBox("What is your full name?")
  n = name.IndexOf(" ")
  firstName = name.Substring(0, n)
  lstOutput.Items.Add("Dear "& firstName & ",")
  lstOutput.Items.Add("")
  lstOutput.Items.Add("We are proud to accept you to Yale.")
End Sub
```

23.
```
Private Sub btnDisplay_Click(...) Handles bnDisplay.Click
  'Determine the area of a rectangle
  Dim length, width, area As Double
  length = CDbl(txtLength.Text)
  width = CDbl(txtWidth.Text)
  area = length * width
  txtOutput.Text = "The area of the rectangle is "& area
End Sub
```

24.
```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  'Convert feet and inches to centimeters
  Dim str As String
  Dim feet, inches, totalInches, centimeters As Double
  str = "Give the length in feet and inches. "
  feet = CDbl(InputBox(str & "Enter the number of feet."))
  inches = CDbl(InputBox(str & "Enter the number of inches. "))
  totalInches = 12 * feet + inches
  centimeters = 2.54 * totalInches
  txtOutput.Text = "The length in centimeters is "& centimeters
End Sub
```

In Exercises 25 and 26, write a line of code to carry out the task. Specify where in the program the line of code would occur.

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

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

In Exercises 27 through 32, write a program to perform the stated task. The input, processing, and output should be performed by calls to Sub procedures.

27. Request a person's first name and last name as input and display the corresponding initials.

28. Request the amount of a restaurant bill as input and display the amount, the tip (15 percent), and the total amount.

29. Request the cost and selling price of an item of merchandise as input and display the percentage markup. Test the program with a cost of $4 and a selling price of $6. Note: The percentage markup is (sellingprice cost) / cost.

[Page 168]

30. Read the number of students in public colleges (12.1 million) and private colleges (3.7 million) from a file, and display the percentage of college students attending public colleges.

31. Read a baseball player's name (Sheffield), times at bat (557), and hits (184) from a file and display his name and batting average. Note: Batting average is calculated as (hits)/ (times at bat).

32. Request three numbers as input, and then calculate and display the average of the three numbers.

33. The Hat Rack is considering locating its new branch store in one of three malls. The following file gives the monthly rent per square foot and the total square feet available at each of the three locations. Write a program to display a table exhibiting this information along with the total monthly rent for each mall.

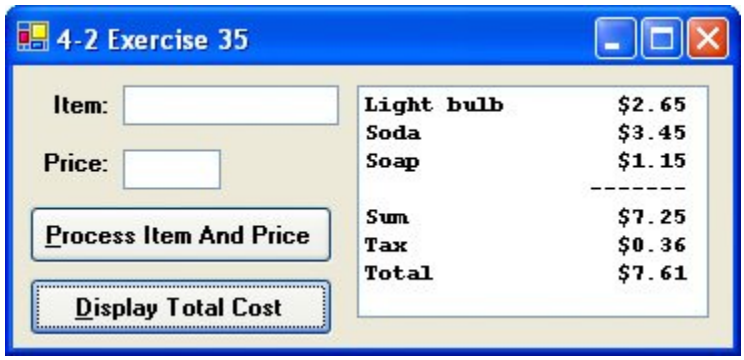
(Assume the nine lines of the file MALLS.TXT contain the following data: Green Mall, 6.50, 583, Red Mall, 7.25, 426, Blue Mall, 5.00, 823.)

34. Write a program that uses the data in the file CHARGES.TXT to display the end-of-month credit-card balances of three people. (Each set of four lines gives a person's name, beginning balance, purchases during month, and payment for the month.) The end-of-month balance is calculated as [finance charges] + [beginning-of-month balance] + [purchases] - [payment], where the finance charge is 1.5 percent of the beginning-of-month balance.

(Assume the 12 lines of the file CHARGES.TXT contain the following data: John Adams, 125.00, 60.00, 110.00, Sue Jones, 0, 117.25, 117.25, John Smith, 350.00, 200.50, 300.00.)

35. Write a program to produce a sales receipt. Each time the user clicks on a button, an item and its price should be read from a pair of text boxes and displayed in a list box. Use a class-level variable to track the sum of the prices. When the user clicks on a second button (after all the entries have been made), the program should display the sum of the prices, the sales tax (5 percent of total), and the total amount to be paid. Figure 4.5 shows a sample output of the program.

Figure 4.5. Sample output for Exercise 35.



[Page 169]

Solutions to Practice Problems 4.2

1.
```
0 0 buckle my shoe.
1 2
```

This program illustrates the local nature of the variables in a Sub procedure. Notice that the variables b and c appearing in the Sub procedure have no relationship whatsoever to the variables of the same name in the event procedure. In a certain sense, the variables inside the Sub procedure can be thought of as having alternate names, such as bRhyme and cRhyme.

2.
```
1 2
2 1
2 1
```

Both variables are passed by reference and so have their values changed by the Sub procedure.

3.
```
1 2
2 1
2 2
```

Here amt1 is passed by reference and amt2 is passed by value. Therefore, only amt1 has its value changed by the Sub procedure.

4.
```
1 2
2 1
1 2
```

Both variables are passed by value, so their values are not changed by the Sub procedure.

PREV PREV NEXT NEXT

[Page 169 (continued)]

4.3. Function Procedures

Visual Basic has many built-in functions. In one respect, functions are like miniature programs. They use input, they process the input, and they have output. Some functions we encountered earlier are listed in Table 4.2.

Table 4.2. Some Visual Basic built-in functions.

Function	Example	Input	Ouput
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 involve several values, the output always consists of a single value. The items inside the parentheses can be literals (as in Table 4.2), variables, or expressions.

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 defined in much the same way as Sub procedures and 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 can be used in expressions in exactly the same way as built-in functions. Programs refer to them as if they were literals, variables, or expressions. Function procedures are defined by function blocks of the form

---

[Page 170]

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

The variables appearing in the top line are called parameters. Variables declared by statements inside the function block have local scope. Function names should be suggestive of the role performed and must conform to the rules for naming variables. The type DataType, which specifies the type of the output, will be one of String, Integer, Double, and so on. In the preceding general code, the next-to-last line specifies the output, which must be of type DataType. Like Sub procedures, Function procedures are typed directly into the Code window. (The last line, End Function, will appear automatically after the

first line is entered into the Code window.) A variable passed to a Function procedure is normally passed by value. It can also be passed by reference and thereby possibly have its value changed by the Function procedure. However, passing a variable by reference violates good design principles, since a function is intended to only create a single result and not cause any other changes.

Two examples of Function procedures are as follows:

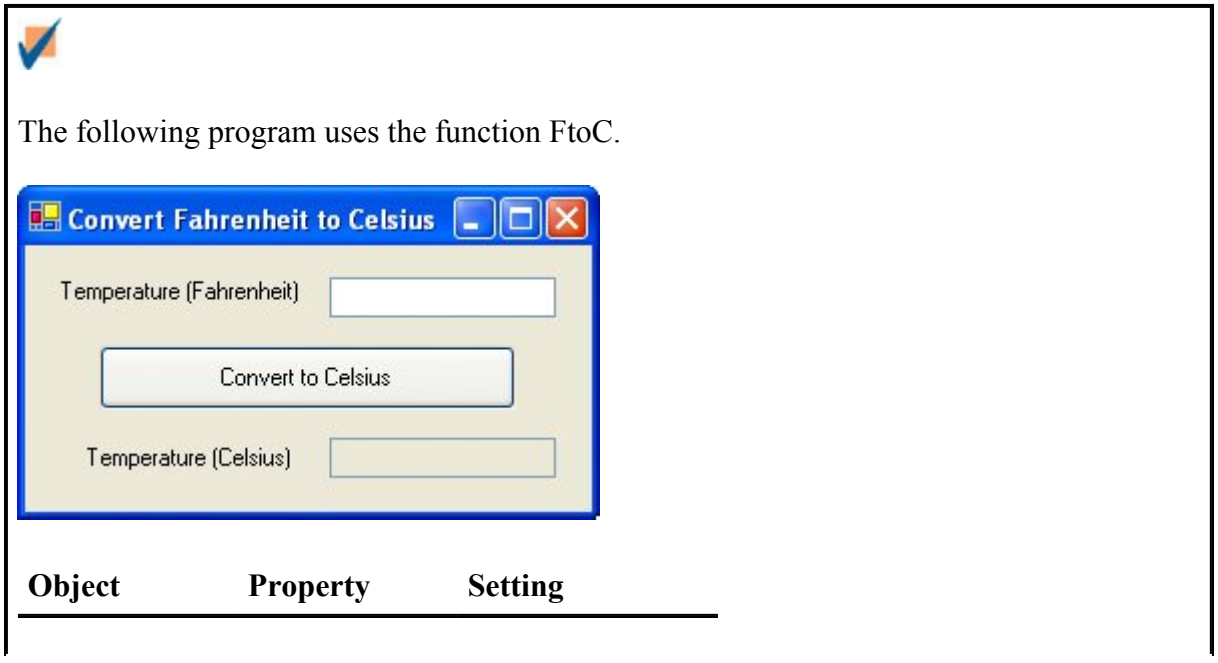
```
Function FtoC(ByVal t As Double) As Double
  'Convert Fahrenheit temperature to Celsius
  Return (5 / 9) * (t - 32)
End Function
Function FirstName(ByVal name As String) As String
  'Extract the first name from a full name
  Dim firstSpace As Integer
  firstSpace = name.IndexOf(" ")
  Return name.Substring(0, firstSpace)
End Function
```

The value of each of the preceding functions is assigned by a statement of the form `Return expression`. The variables t and name appearing in the preceding functions are parameters. They can be replaced with any variable of the same type without affecting the function definition. For instance, the function FtoC could have been defined as

```
Function FtoC(ByVal temp As Double) As Double
  'Convert Fahrenheit temperature to Celsius
  Return (5 / 9) * (temp - 32)
End Function
```

[Page 171]

Example 1.



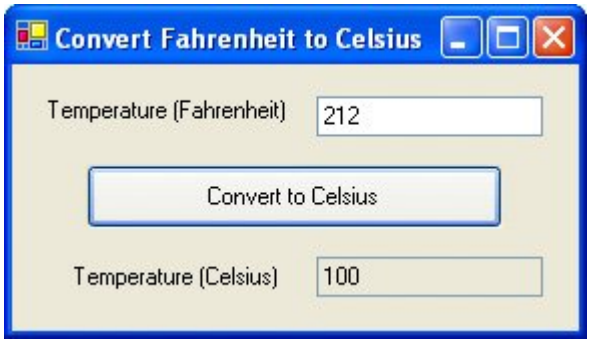
The following program uses the function FtoC.

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 the button.]



Example 2.

(This item is displayed on pages 171 - 172 in the print version)

The following program uses the function FirstName.

Object	Property	Setting
frmFirstName	Text	Extract First Name
lblName	Text	Name
txtFullName		
btnDetermine	Text	Determine First Name
txtFirstName	ReadOnly	True

[Page 172]

```
Private Sub btnDetermine_Click(...) Handles btnDetermine.Click
  'Determine a person's first name
  Dim name As String
  name = txtFullName.Text
  txtFirstname.Text = FirstName(name)
End Sub
Function FirstName(ByVal name As String) As String
  'Extract the first name from a full name
  Dim firstSpace As Integer
  firstSpace = name.IndexOf(" ")
  Return name.Substring(0, firstSpace)
End Function
```

[Run, type Thomas Woodrow Wilson into the text box, and then click the button.]

User-Defined Functions Having Several Parameters

The input to a user-defined function can consist of one or more values. Two examples of functions with several parameters follow. One-letter variable names have been used so the mathematical formulas will look familiar and be readable. Because the names are not descriptive, the meanings of these variables are

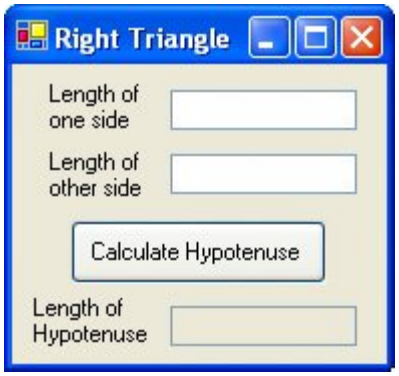
carefully spelled out in comment statements.

```
Function Hypotenuse(ByVal a As Double, ByVal b As Double) As Double
  'Calculate the hypotenuse of a right triangle
  'having sides of lengths a and b
  Return Math.Sqrt(a ^ 2 + b ^ 2)
End Function
Function FutureValue(ByVal p As Double, ByVal r As Double, _
            ByVal c As Double, ByVal n As Double) As Double
  'Find the future value of a bank savings account
  'p principal, the amount deposited
  'r annual rate of interest
  'c number of times interest is compounded per year
  'n number of years
  Dim i As Double  'interest rate per period
  Dim m As Double  'total number of times interest is compounded
  i = r / c
  m = c * n
  Return p * ((1 + i) ^ m)
End Function
```

[Page 173]

Example 3.

The following program uses the Hypotenuse function.

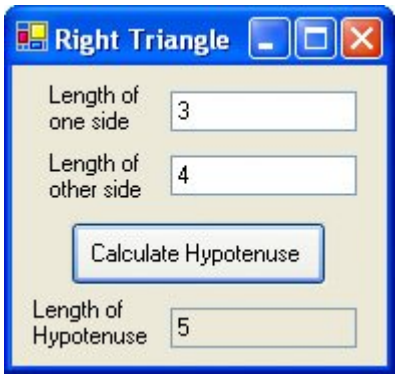


Object	Property	Setting
frmPythagoras	Text	Right Triangle
lblSideOne	AutoSize	False
	Text	Length of one side
txtSideOne		
lblSideTwo	AutoSize	False

	Text	Length of other side
txtSideTwo		
btnCalculate	Text	Calculate Hypotenuse
lblHyp	AutoSize	False
	Text	Length of Hypotenuse
txtHyp	ReadOnly	True

```
Private Sub btnCalculate_Click(...) Handles btnCalculate.Click
  'Calculate the length of the hypotenuse of a right triangle
  Dim a, b As Double
  a = CDbl(txtSideOne.Text)
  b = CDbl(txtSideTwo.Text)
  txtHyp.Text = CStr(Hypotenuse(a, b))
End Sub
Function Hypotenuse(ByVal a As Double, ByVal b As Double) As Double
  'Calculate the hypotenuse of a right triangle
  'having sides of lengths a and b
  Return Math.Sqrt(a ^ 2 + b ^ 2)
End Function
```

[Run, type 3 and 4 into the text boxes, and then click the button.]



Example 4.

(This item is displayed on pages 173 - 175 in the print version)

The following program uses the future value function. With the responses shown, 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 4 * 5, or 20, interest periods.

[Page 174]

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   'principal, the amount deposited
  Dim  r As Double   'annual rate of interest
```

```
  Dim  c As Double   'number of times interest is compounded per year
  Dim  n As Double   'number of years
  InputData(p, r, c, n)
  DisplayBalance(p, r, c, n)
End Sub
Sub InputData(ByRef p As Double, ByRef r As Double, _
              ByRef c As Double, ByRef n As Double)
      'Get the four values from the text boxes
      p = CDbl(txtAmount.Text)
      r = CDbl(txtRate.Text)
      c = CDbl(txtNumComp.Text)
      n = CDbl(txtNumYrs.Text)
End Sub

Sub DisplayBalance(ByVal p As Double, ByVal  r As Double, _
                   ByVal c As Double, ByVal n As Double)
  'Display the balance in a text box
  Dim balance As Double
  balance = FutureValue(p, r, c, n)
  txtbalance.Text = FormatCurrency(balance)
End Sub
Function FutureValue(ByVal p As Double, ByVal r As Double, _
                     ByVal c As Double, ByVal n As Double) As Double
  'Find the future value of a bank savings account
  'p  principal, the amount deposited
  'r  annual rate of interest
  'c  number of times interest is compounded per year
  'n  number of years
```

[Page 175]

```
  Dim i As Double   'interest rate per period
  Dim m As Double   '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, then click the button.]

User-Defined Functions Having No Parameters

Function procedures, like Sub procedures, need not have any parameters.

Example 5.

The following program uses a parameterless function.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Request and display a saying
  txtBox.Text = Saying()
End Sub
Function Saying() As String
  'Retrieve a saying from the user
  Return InputBox("What is your favorite saying?")
End Function
```

[Run, click the button, and then type "Less is more." into the message box.]

The saying "Less is more." is displayed in the text box.

Comparing Function Procedures with Sub Procedures

Function procedures differ from Sub procedures in the way they are accessed. Sub procedures are invoked with call statements, whereas functions are invoked by placing them where you would otherwise expect to find a literal, variable, or expression. Unlike a Function procedure, a Sub procedure can't be used in an expression.

Function procedures can perform the same tasks as Sub procedures. For instance, they can request input and display text. However, Function procedures are primarily used to calculate a single value. Normally, Sub procedures are used to carry out other tasks.

[Page 176]

The Sub procedures considered in this book terminate only when End Sub is reached. On the other hand, Function procedures terminate as soon as the first Return statement is executed. For instance, if a Return statement is followed by a sequence of statements and the Return statement is executed, then the sequence of statements will not be executed.

Collapsing a Procedure with a Region Directive

A group of procedures or class-level variables can be collapsed behind a captioned rectangle. This task is carried out with a so-called Region directive. To specify a region, precede the code to be collapsed with a line of the form

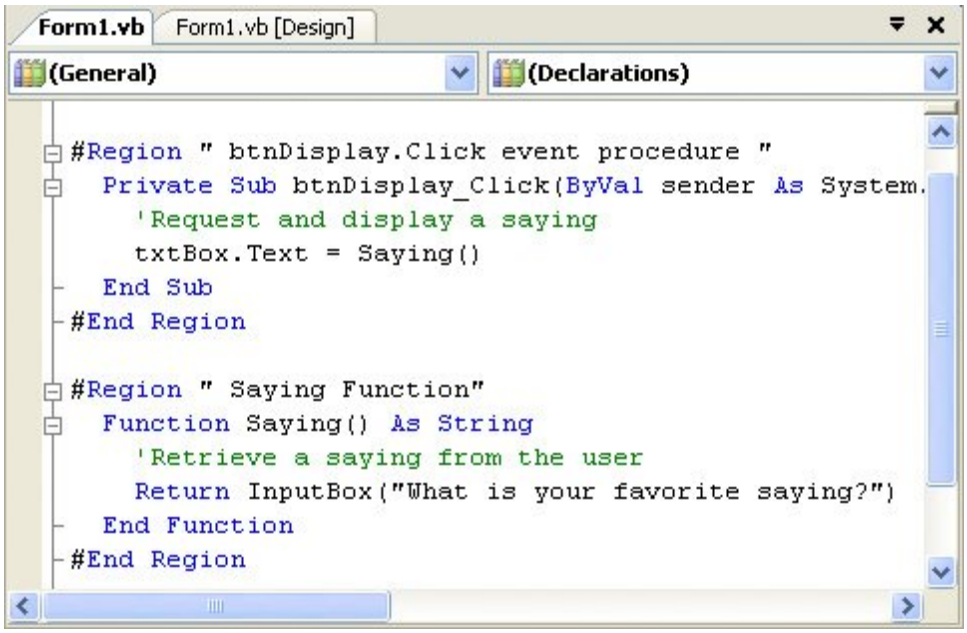
```
#Region "Text to be displayed in the rectangle."
```

and follow the code with the line

```
#End Region
```

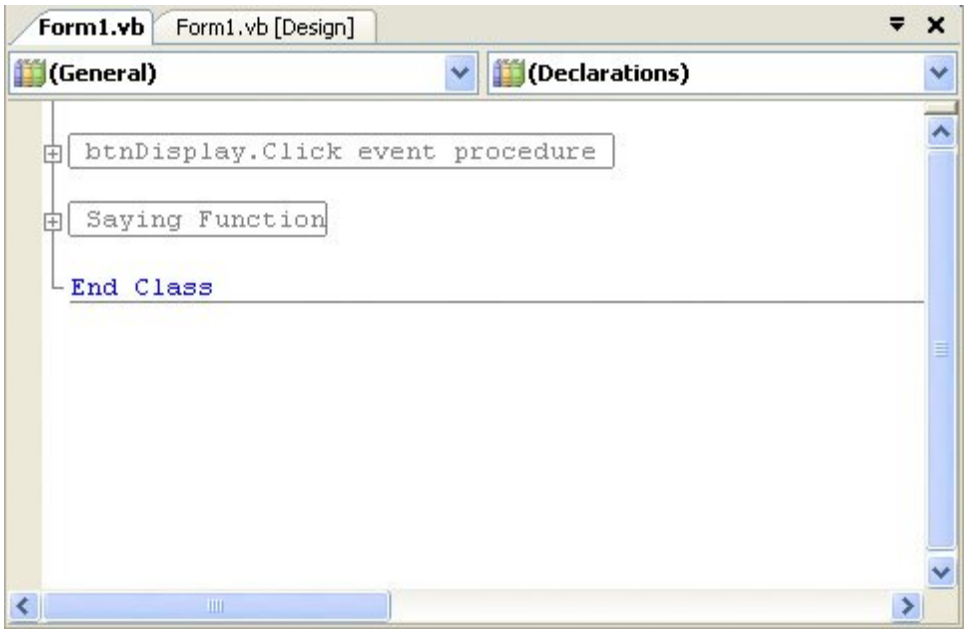
A tiny box holding a minus sign will appear to the left of the #Region line. To collapse the code, click on the minus sign. The code will be hidden behind a rectangle captioned with the text you specified and the minus sign will be replaced by a plus sign. Click on the plus sign to expand the region. The Region directive is used to make a program more readable or to create an outline for a program. In Figure 4.6(a), Region directives have been specified for each procedure in Example 5. In Figure 4.6(b), these two regions have been collapsed.

Figure 4.6(a). Region directives.



[Page 177]

Figure 4.6(b). Collapsed regions.



Practice Problems 4.3

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. What is displayed in the text box when btnCompute is clicked?

```
Private Sub btnCompute_Click(...) Handles btnCompute.Click
  'How many gallons of apple cider can we make?
  Dim gallonsPerBushel, apples As Double
  GetData(gallonsPerBushel, apples)
  DisplayNumOfGallons(gallonsPerBushel, apples)
End Sub
Function Cider(ByVal g As Double, ByVal x As Double) As Double
  Return g * x
End Function
Sub DisplayNumOfGallons(ByVal galPerBu As Double, _
                        ByVal apples As Double)
  txtOutput.Text = "You can make "& Cider(galPerBu, apples) _
                  & " gallons of cider."
End Sub
Sub GetData(ByRef gallonsPerBushel As Double, _
            ByRef apples As Double)
  'gallonsPerBushel Number of gallons of cider one bushel
  'of apples makes
  'apples       Number of bushels of apples available
  gallonsPerBushel = 3
  apples = 9
```

```
End Sub
```

[Page 178]

Exercises 4.3

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 btnDisplay_Click(...) Handles btnDisplay.Click
  'Calculate max. ht. of a ball thrown straight up in the air
  Dim initVel, initHt As Double
  initVel = CDbl(txtVel.Text) 'Initial velocity of ball
  initHt = CDbl(txtHt.Text)   'Initial height of ball
  txtOutput.Text = CStr(MaximumHeight(initVel, initHt))
End Sub
```

[Page 179]

```
Function MaximumHeight(ByVal v As Double, ByVal h As Double) _
                    As Double
  Return h + (v ^ 2 / 64)
End Function
```

(Assume the text boxes contain the values 96 and 256.)

5.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Compute volume of a cylinder
  Dim r As Double = 1   'Radius
  Dim h As Double = 2   'Height
  DisplayVolume(r, h)
  r = 3
  h = 4
  DisplayVolume(r, h)
End Sub

Function Area(ByVal r As Double) As Double
  'Compute area of a circle of radius r
  Return 3.14159 * r ^ 2
End Function

Sub DisplayVolume(ByVal r As Double, ByVal h As Double)
  lstBox.Items.Add("Volume of cylinder having base area "& _
        Area(r) & " and height "& h & " is "& (h * Area(r)))
End Sub
```

6.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Determine the day of the week from its number
  Dim days As String, num As Integer
  days = "SunMonTueWedThuFriSat"
  num = CInt(InputBox("Enter the number of the day."))
  txtOutput.Text = "The day is "& DayOfWk(days, num) & "."
End Sub

Function DayOfWk(ByVal x As String, ByVal n As Integer) As String
  'x String containing 3-letter abbreviations of days
  'n The number of the day
  Dim position As Integer
  position = 3 * n - 3
  Return x.Substring(position, 3)
End Function
```

(Assume the response is 4.)

7.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Demonstrate local variables
  Dim word As String = "Choo "
  txtOutput.Text = TypeOfTrain()
End Sub
```

[Page 180]

```
Function TypeOfTrain() As String
  'Concatenate the value of word with itself
  Dim word As String
  word &= word
  Return word & "train"
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 btnDisplay_Click(...) Handles btnDisplay.Click
  Dim word As String
  word = "moral"
  Negative(word)
  word = "political"
  Negative(word)
End Sub

Function AddA(ByVal word As String) As String
  Return "a"& word
  End Function

Sub Negative(ByVal word As String)
  lstOutput.Items.Add(word & " has the negative "& AddA(word))
End Sub
```

10.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim city As String, pop, shrinks As Double
  Dim sr As IO.StreamReader = IO.File.OpenText("DOCS.TXT")
  city = sr.ReadLine
```

```
  pop = CDbl(sr.ReadLine)
  shrinks = CDbl(sr.ReadLine)
  DisplayData(city, pop, shrinks)
  city = sr.ReadLine
  pop = CDbl(sr.ReadLine)
  shrinks = CDbl(sr.ReadLine)
  sr.Close()
  DisplayData(city, pop, shrinks)
End Sub

Sub DisplayData(ByVal city As String, ByVal pop As Double, _
                ByVal shrinks As Double)
  lstBox.Items.Add(city & " has "& ShrinkDensity(pop, shrinks) _
                   & " psychiatrists per 100,000 people.")
End Sub
```

[Page 181]

```
Function ShrinkDensity(ByVal pop As Double, _
                       ByVal shrinks As Double) As Double
  Return Int(100000 * (shrinks / pop))
End Function
```

(Assume the six lines of the file DOCS.TXT contain the following data: Boston, 2824000, 8602, Denver, 1633000, 3217.)

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 21, 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 required to make the can.

14. According to Plato, a man should marry a woman whose age is half his age plus seven years. Write a program that requests a man's age as input and gives the ideal age of his wife.

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.

[Page 182]

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 his age and resting heart rate (pulse 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 ingredients 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. Rewrite the population-density program from Example 4 of Section 4.1 using a function to calculate the population density.

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. A sample outcome when 800 is typed into the text box for the amount deposited each month follows.

```
Month 1     800.00
Month 2   1,604.00
Month 3   2,412.02
Month 4   3,224.08
```

21. Write a program to request the name of a United States senator as input and display the address and greeting for a letter to the senator. Assume the name has two parts, and use a function to determine the senator's last name. A sample 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,
```

[Page 183]

Solutions to Practice Problems 4.3

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. `You can make 27 gallons of cider` In this program, the function was called by a Sub procedure rather than by an event procedure.

PREV PREV NEXT NEXT

[Page 183 (continued)]

4.4. Modular Design

Top-Down Design

Full-featured software usually requires large programs. Writing the code for an event procedure in such a Visual Basic program might pose a complicated problem. One method programmers use to make a complicated problem more understandable is to divide it into smaller, less complex subproblems. Repeatedly using a "divide-and-conquer" approach to break up a large problem into smaller subproblems is called stepwise refinement. Stepwise refinement is part of a larger methodology of writing programs known as top-down design. The term top-down refers to the fact that the more general tasks occur near the top of the design and tasks representing their refinement occur below. Top-down design and structured programming emerged as techniques to enhance programming productivity. Their use leads to programs that are easier to read and maintain. They also produce programs containing fewer initial errors, with these errors being easier to find and correct. When such programs are later modified, there is a much smaller likelihood of introducing new errors.

The goal of top-down design is to break a problem into individual tasks, or modules, that can easily be transcribed into pseudocode, flowcharts, or a program. First, a problem is restated as several simpler problems depicted as modules. Any modules that remain too complex are broken down further. The process of refining modules continues until the smallest modules can be coded directly. Each stage of refinement adds a more complete specification of what tasks must be performed. The main idea in top-down design is to go from the general to the specific. This process of dividing and organizing a problem into tasks can be pictured using a hierarchy chart. When using top-down design, certain criteria should be met:

1. The design should be easily readable and emphasize small module size.
2. Modules proceed from general to specific as you read down the chart.
3. The modules, as much as possible, should be single minded. That is, they should only perform a single well-defined task.
4. Modules should be independent of each other as much as possible, and any relationships among modules should be specified.

This process is illustrated with the following example.

[Page 184]

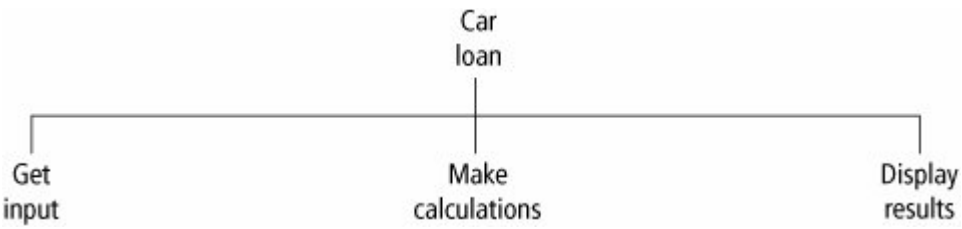
Example 1.

(This item is displayed on pages 184 - 185 in the print version)

The chart in Figure 4.7 is a hierarchy chart for a program that gives certain information about a car loan. The inputs are the amount of the loan, the duration (in years), and the interest rate. The output consists of the monthly payment and the amount of interest paid during the first month. In the broadest sense, the program calls for obtaining the input,

making calculations, and displaying the output. Figure 4.7 shows these tasks as the first row of a hierarchy chart.

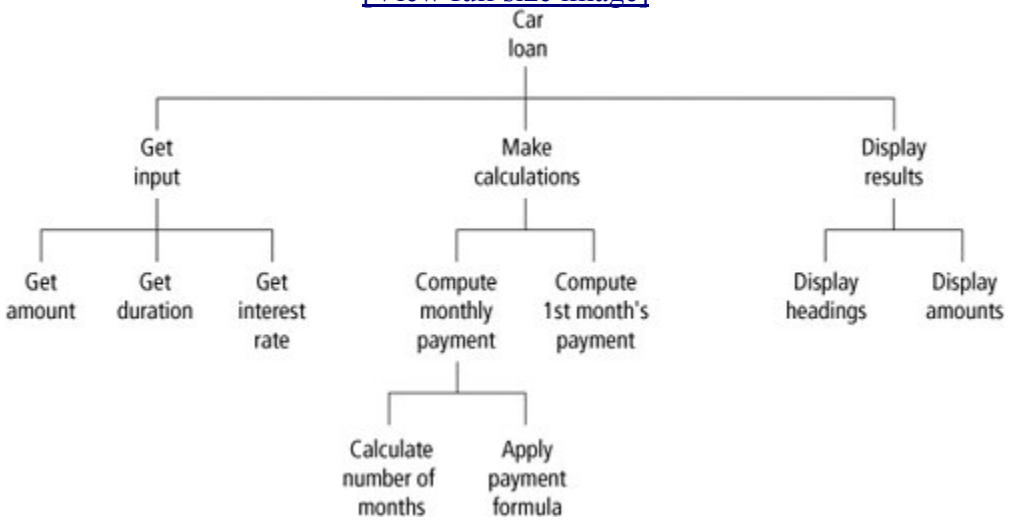
Figure 4.7. Beginning of a hierarchy chart for the car loan program.



Each of these tasks can be refined into more specific subtasks. (See Figure 4.8 for the final hierarchy chart.) Most of the subtasks in the third row are straightforward and so do not require further refinement. For instance, the first month's interest is computed by multiplying the amount of the loan by one-twelfth of the annual rate of interest. The most complicated subtask, the computation of the monthly payment, has been broken down further. This task is carried out by applying a standard formula found in finance books; however, the formula requires the number of payments.

Figure 4.8. Hierarchy chart for the car loan program.

[View full size image]



[Page 185]

It is clear from the hierarchy chart that the top modules manipulate the modules beneath them. While the higher-level modules control the flow of the program, the lower-level modules do the actual work. By designing the top modules first, specific processing decisions can be delayed.

Structured Programming

A program is said to be structured if it meets modern standards of program design. Although there is no formal definition of the term structured program, computer scientists are in uniform agreement that such programs should have modular design and use only the three types of logical structures discussed in Chapter 2: sequences, decisions, and loops.

Sequences: Statements are executed one after another.

Decisions: One of several blocks of program code is executed based on a test for some condition.

Loops (iteration): One or more statements are executed repeatedly as long as a specified condition is true.

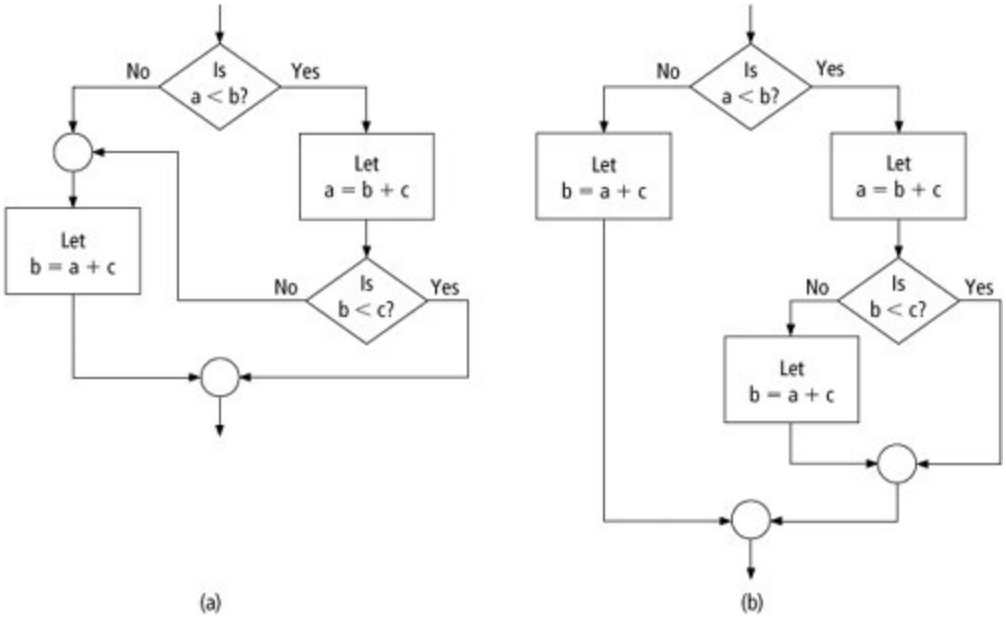
Chapters 5 and 6 are devoted to decisions and loops, respectively.

One major shortcoming of the earliest programming languages was their reliance on the GoTo statement. This statement was used to branch (that is, jump) from one line of a program to another. It was common for a program to be composed of a convoluted tangle of branchings that produced confusing code referred to as spaghetti code. At the heart of structured programming is the assertion of E. W. Dijkstra that GoTo statements should be eliminated entirely because they lead to complex and confusing programs. Two Italians, C. Bohm and G. Jacopini, were able to prove that GoTo statements are not needed and that any program can be written using only the three types of logic structures discussed before.

Structured programming requires that all programs be written using sequences, decisions, and loops. Nesting of such statements is allowed. All other logical constructs, such as GoTos, are not allowed. The logic of a structured program can be pictured using a flowchart that flows smoothly from top to bottom without unstructured branching (GoTos). The portion of a flowchart shown in Figure 4.9(a) (on the next page) contains the equivalent of a GoTo statement and, therefore, is not structured. A correctly structured version of the flowchart in which the logic flows from the top to the bottom appears in Figure 4.9(b).

Figure 4.9. Flowcharts illustrating the removal of a GoTo statement.
(This item is displayed on page 186 in the print version)

[View full size image]



Advantages of Structured Programming

The goal of structured programming is to create correct programs that are easy to write, understand, and change. Let us now take a closer look at the way modular design, along with a limited number of logical structures, contributes to attaining these goals.

1. Easy to write.

   Modular design increases the programmer's productivity by allowing him or her to look at the big picture first and focus on the details later. During the actual coding, the programmer works with a manageable chunk of the program and does not have to think about an entire complex program.

[Page 186]

   Several programmers can work on a single large program, each taking responsibility for a specific module.

   Studies have shown structured programs require significantly less time to write than standard programs.

   Often, procedures written for one program can be reused in other programs requiring the same task. Not only is time saved in writing a program, but reliability is enhanced, because reused procedures will already be tested and debugged. A procedure that can be used in many programs is said to be reusable.

2. Easy to debug.

   Because each procedure is specialized to perform just one task or several related tasks, a procedure can be checked individually to determine its reliability. A dummy program, called a driver, is set up to test the procedure. The driver contains the minimum definitions needed to call

the procedure to be tested. For instance, if the procedure to be tested is a function, the driver program assigns diverse values to the arguments and then examines the corresponding function return values. The arguments should contain both typical and special-case values.

The program can be tested and debugged as it is being designed with a technique known as stub programming. In this technique, the key event procedures and perhaps some of the smaller procedures are coded first. Dummy procedures, or stubs, are written for the remaining procedures. Initially, a stub procedure might consist of a message box to indicate that the procedure has been called, and thereby confirm that the procedure was called at the right time. Later, a stub might simply display values passed to it in order to confirm not only that the procedure was called, but also that it received the correct values from the calling procedure. A stub also can assign new values to one or more of its parameters to simulate either input or computation. This provides greater control of the conditions being tested. The stub procedure is always simpler than the actual procedure it represents. Although the stub program is only a skeleton of the final program, the program's structure can still be debugged and tested. (The stub program consists of some coded procedures and the stub procedures.)

[Page 187]

Old-fashioned unstructured programs consist of a sequence of instructions that are not grouped for specific tasks. The logic of such a program is cluttered with details and therefore difficult to follow. Needed tasks are easily left out and crucial details easily neglected. Tricky parts of the program cannot be isolated and examined. Bugs are difficult to locate because they might be present in any part of the program.

3. Easy to understand.

   The interconnections of the procedures reveal the modular design of the program.

   The meaningful procedure names, along with relevant comments, identify the tasks performed by the modules.

   The meaningful variable names help the programmer to recall the purpose of each variable.

4. Easy to change.

   Because a structured program is self-documenting, it can easily be deciphered by another programmer.

   Modifying a structured program often amounts to inserting or altering a few procedures rather than revising an entire complex program. The programmer does not even have to look at most of the program.

Object-Oriented Programming

An object is an encapsulation of data and code that operates on the data. Like controls, objects have properties, respond to methods, and raise events. The most effective type of programming for complex problems is called object-oriented design. An object-oriented program can be viewed as a collection of cooperating objects. Most modern programmers use a blend of traditional structured programming along

with object-oriented design.

Visual Basic.NET was the first version of Visual Basic that was truly object-oriented; in fact, every element such as a control or a String is actually an object. This book illustrates the building blocks of Visual Basic in the initial chapters and then puts them together using object-oriented techniques in Chapter 11. Throughout the book, an object-oriented approach is taken whenever feasible.

[Page 188]

A Relevant Quote

We end this section with a few paragraphs from Dirk Gently's Holistic Detective Agency, by Douglas Adams, Pocketbooks, 1987:

> "What really is the point of trying to teach anything to anybody?"
>
> This question seemed to provoke a murmur of sympathetic approval from up and down the table.
>
> Richard continued, "What I mean is that if you really want to understand something, the best way is to try and explain it to someone else. That forces you to sort it out in your own mind. And the more slow and dim-witted your pupil, the more you have to break things down into more and more simple ideas. And that's really the essence of programming. By the time you've sorted out a complicated idea into little steps that even a stupid machine can deal with, you've certainly learned something about it yourself. The teacher usually learns more than the pupil. Isn't that true?"

PREV PREV NEXT NEXT

[Page 188 (continued)]

Chapter 4 Summary

1. A general procedure is a portion of a program that is accessed by event procedures or other general procedures. The two types of general procedures are Sub procedures and Function procedures.

2. Sub procedures are defined in blocks beginning with Sub statements and ending with End Sub statements. A Sub procedure is accessed (called) by a statement consisting of the name of the procedure.

3. Function procedures are defined in blocks beginning with Function statements and ending with End Function statements. A function is invoked by a reference in an expression and returns a value.

4. In any procedure, the arguments appearing in the calling statement must match the parameters of the Sub or Function statement in number, type, and order. They need not match in name.

5. A variable declared in the Declarations section of the Code window is class-level. Such a variable is available to every procedure in the form's code and retains its value from one procedure invocation to the next.

6. Variables declared with a Dim statement inside a procedure are local to the procedure. The values of these variables are reinitialized each time the procedure is called. A variable with the same name appearing in another part of the program is treated as a different variable.

7. Structured programming uses modular design to refine large problems into smaller subproblems. Programs are coded using the three logical structures of sequences, decisions, and loops.

PREV PREV NEXT NEXT

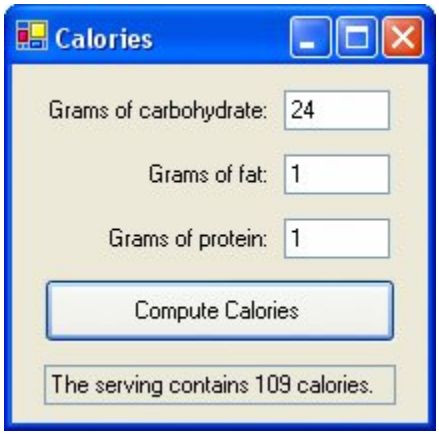
[Page 188 (continued)]

Chapter 4 Programming Projects

1. The numbers of calories per gram of carbohydrate, fat, and protein are 4, 9, and 4, respectively. Write a program that requests the nutritional content of a serving of food and displays the number of calories in the serving. The input and output should be handled by Sub procedures and the calories computed by a function. A sample run for a typical breakfast cereal is shown in Figure 4.10.

[Page 189]

Figure 4.10. Sample run for Programming Project 1.



2. Annually, 1.7 billion dollars worth of toothpaste are sold each year in the United States. Table 4.3 gives the market share for the four top brands. Write a program that displays the annual sales (in millions of dollars) for each of the top four. The input and output should be handled by Sub procedures and the annual sales calculated by a Function procedure.

Table 4.3. 2001 market shares of the top-selling toothpaste brands.

Company	Market Share
Crest	19.4%
Colgate	17.2%
Aquafresh	8.0%
Colgate Total	6.6%

Source: The World Almanac and Book of Facts 2002.

3. Table 4.4 gives the research and development budgets (in billions of dollars) for four departments of the federal government for the years 2004 and 2005. Write a program that displays the percentage change in the budget for each department. Sub procedures should be used for input and output, and the percentage change should be computed with a Function procedure. Note: The percentage change is ([2005 budget] [2004 budget])/[2004 budget].

Table 4.4. Research and development budget for several departments.

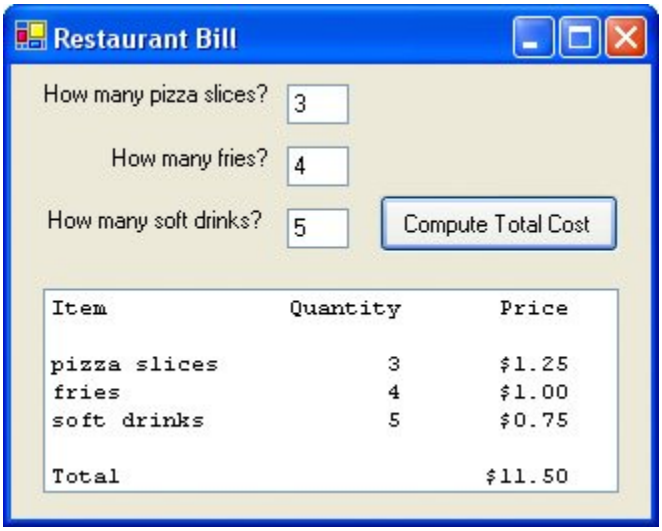
Department	2004 Budget	2005 Budget
Defense	65.6	70.3
Health and Human Services	28.5	29.1
NASA	10.9	11.1
Homeland Security	1.0	1.2

Source: American Association for the Advancement of Science.

[Page 190]

4. A fast-food vendor sells pizza slices ($1.25), fries ($1.00), and soft drinks ($.75). Write a program to compute a customer's bill. The program should request the quantity of each item ordered in a Sub procedure, calculate the total cost with a Function procedure, and use a Sub procedure to display an itemized bill. A sample output is shown in Figure 4.11.

Figure 4.11. Sample run for Programming Project 4.



5. Write a program to generate a business travel expense attachment for an income-tax return. The program should request as input the name of the organization visited, the date and location of the visit, and the expenses for meals and entertainment, airplane fare, lodging, and taxi fares. (Only 50% of the expenses for meals and entertainment are deductible.) A possible form layout and run are shown in Figures 4.12 and 4.13, respectively. The output is displayed in a list box that becomes visible when the button is clicked. Sub procedures should be used for the input and output.

[Page 191]

Figure 4.12. Form with sample data for Programming Project 5.

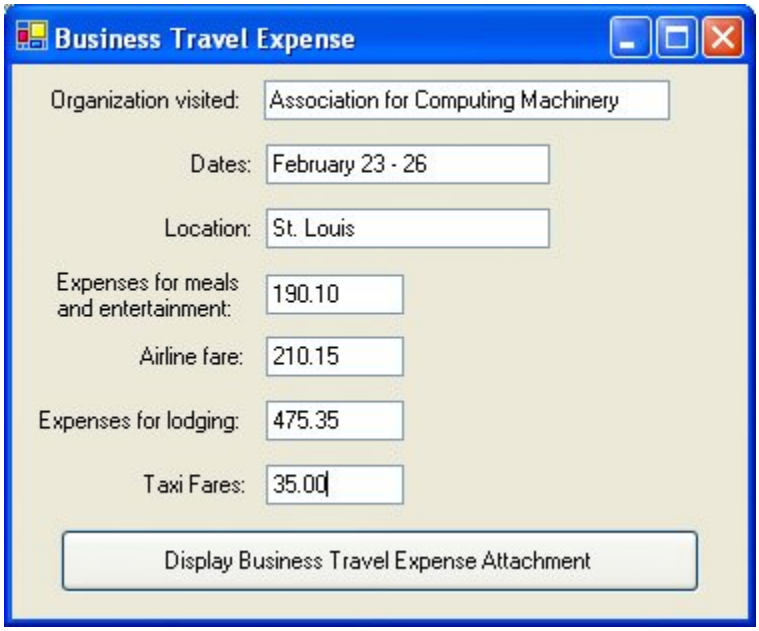
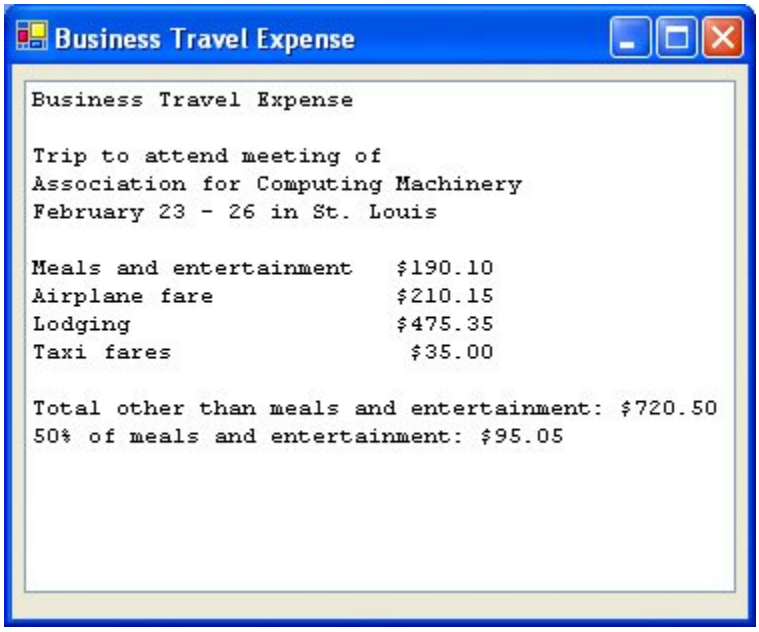


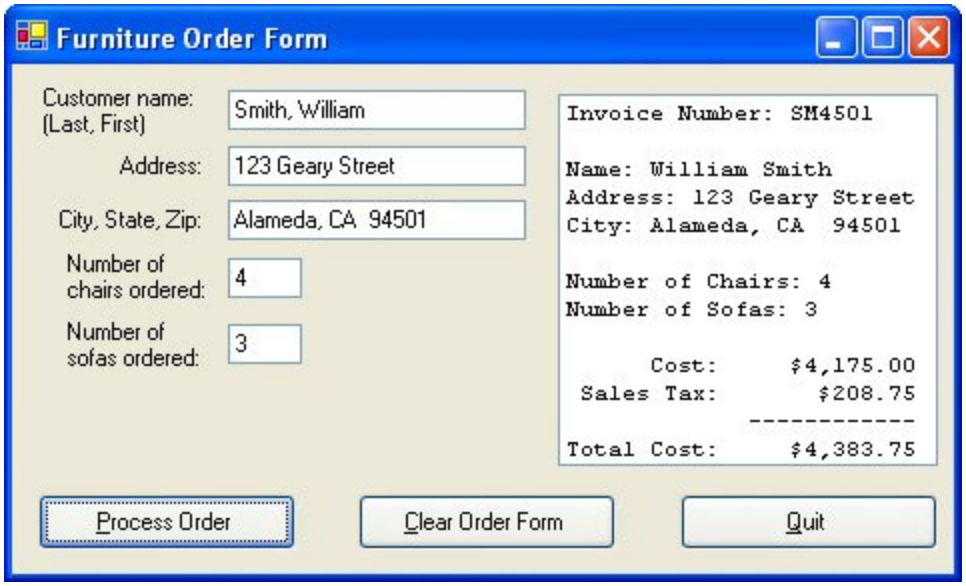
Figure 4.13. Output in list box for sample run of Programming Project 5.



[Page 192]

6. A furniture manufacturer makes two types of furniturechairs and sofas. The file PRICE&TAXDATA.TXT contains three numbers giving the cost per chair, cost per sofa, and sales tax rate. Write a program to create an invoice form for an order. See Figure 4.14. After the data on the left side of Figure 4.14 are entered, you can display an invoice in a list box by pressing the Process Order button. You can press the Clear Order Form button to clear all text boxes and the list box, and you can press the Quit button to exit the program. The invoice number consists of the capitalized first two letters of the customer's last name, followed by the last four digits of the zip code. The customer name is input with the last name first, followed by a comma, a space, and the first name. However, the name is displayed in the invoice in the proper order. The generation of the invoice number and the reorder of the first and last names should be carried out in Function procedures.

Figure 4.14. Sample run for Programming Project 6.



PREV NEXT